Analysis and Design Document: Computing Research Projects Repository

JHUB Africa | April 2025

Group Members:

Dr. Lawrence Nderu, *lawrence_nderu@live.com*
Gikuru Joseph Nderitu, *gikurujoseph53@gmail.com*
Alex Maina, *alemaina867@gmail.com*
Boniface Mwangi, *bmwh6475@gmail.com*
Loren Odhiambo, *lorencepoppa823@gmail.com*
Felix Ouma, *felixouma4859@gmail.com*

Table of Contents

1. Introduction

Computing Research Projects Repository (CRPR) is a dynamic, collaborative platform designed to showcase, manage, and share research work within the computing domain. Built under the umbrella of JHUB Africa and the **Department of Computing** at **Jomo Kenyatta University of Agriculture and Technology (JKUAT)**, CRPR empowers students, researchers, and faculty to not only archive their academic projects but also foster interdisciplinary collaboration, resource sharing, and innovation.

Recognizing the need for a centralized system that accommodates the evolving landscape of computing research, CRPR provides a seamless interface for project submission, exploration, and management. From undergraduate research to high-impact doctoral and master's level work, the platform offers flexible access through affordable monthly subscriptions—ensuring a broader audience benefits from high-end academic resources. These include but are not limited to audio files, video presentations, research papers (PDFs, DOCX), slide decks (PPT), and more. Where applicable, projects are enhanced with demo links and GitHub repositories to promote transparency and reproducibility.

This document captures the current stage of CRPR's development journey—from conceptualization and user-centered requirements gathering to structured analysis and thoughtful design. It highlights the platform's architectural decisions, functional and non-functional specifications, UI/UX considerations, and overall system modelling.

The goal is to bridge visionary academic objectives with technical execution—translating the real-world needs of computing scholars into robust system components, intuitive user experiences, and scalable workflows, all while adhering to modern software engineering standards.

2. Functional Requirements

Our system is centered around key functions designed to meet both operational and commercial goals:

I. User Authentication and Account Management

- FR1.1: The platform allows users to create accounts using Django's built-in authentication system.
- FR1.2: Users can log in using third-party authentication via GitHub and LinkedIn.
- FR1.3: Users securely log in with their registered credentials.
- FR1.4: A password reset feature is available for users who forget their credentials.
- FR1.5: Users can manage and update their profile information after logging in.

II. Project Creation and Management

- FR2.1: Authenticated users can create and publish research projects on the platform.
- FR2.2: Users upload various resource files related to their projects, including PDFs, DOCX, PPT, audio, and video files.
- FR2.3: Project creators can include demo links and GitHub repositories where available.
- FR2.4: Users have full CRUD (Create, Read, Update, Delete) privileges over the projects they create.
- FR2.5: The platform simplifies project creation with a streamlined and user-friendly workflow.
- FR2.6: All users can view project details, including resources and contributor information.
- FR2.7: Projects are categorized and can be tagged by topic, academic level, or computing field.

III. Collaboration Features

- FR3.1: Multiple users can collaborate on a single project.
- FR3.2: Collaborators are visibly listed on the project page.
- FR3.3: Users can engage with projects through comments or discussions (if enabled).

IV. Resource Access and Downloads

- FR4.1: Visitors and authenticated users can browse and download project resources.
- FR4.2: Access to high-end academic projects (e.g., Master's and Doctorate level) is gated behind a subscription paywall.

V. Subscription and Premium Access

- FR5.1: Users can subscribe to a monthly plan to access premium project content.
- FR5.2: Subscriptions unlock advanced academic work created by professors, PhD holders, and postgraduate students.
- FR5.3: The system enforces access controls based on the user's subscription status.

VI. Notification and Alert System

- FR6.1: Subscribed users receive real-time alerts about newly posted or updated projects.
- FR6.2: The system also sends alerts about upcoming events or site announcements.
- FR6.3: Notifications are delivered based on the user's alert preferences and subscription settings.

VII. Admin Dashboard and System Monitoring

- FR7.1: A real-time monitoring dashboard is available for administrators.
- FR7.2: The dashboard provides insights into:
    - Active users and projects
    - Uploaded resources
    - Subscribed users
    - System health and activity trends
- FR7.3: Admins can manage users, projects, subscriptions, and overall site content.

VIII. Search, Filter, and Navigation

- FR8.1: Users can search for projects using keywords such as title, author, tags, or topic.
- FR8.2: Projects can be filtered by academic level, category, or resource type.
- FR8.3: Navigation throughout the platform is intuitive, clean, and responsive across devices.

3. Non-Functional Requirements

I. Performance Requirements

- **NFR1.1**: The platform responds to user actions (e.g., login, project upload, resource download) within 2 seconds under normal load.
- **NFR1.2**: The system supports concurrent access by multiple users without noticeable degradation in performance.
- **NFR1.3**: Project resources (PDFs, videos, etc.) load or download within acceptable time frames, even under peak usage.

II. Scalability

- **NFR2.1**: The system is designed to scale horizontally to support a growing number of users, projects, and uploaded resources.
- **NFR2.2**: The database architecture supports large data volumes, including heavy resource files like video and audio content.

III. Security

- **NFR3.1**: All user authentication and authorization processes follow industry security standards (e.g., Django authentication, OAuth2 for GitHub/LinkedIn).
- **NFR3.2**: User passwords are securely hashed and stored.
- **NFR3.3**: The system uses HTTPS to encrypt all data transmission between clients and the server.

- **NFR3.4**: Users can reset passwords through secure email verification mechanisms.
- **NFR3.5**: Role-based access controls ensure users, subscribers, and admins have access only to authorized features and data.

IV. Usability

- **NFR4.1**: The platform provides a clean, intuitive, and responsive UI/UX design for both desktop and mobile users.
- **NFR4.2**: Project creation and resource upload workflows are optimized to minimize user effort and confusion.
- **NFR4.3**: Documentation or tooltips guide users through key actions such as posting a project or subscribing.

V. Maintainability

- **NFR5.1**: The codebase follows modular design principles to make updates and feature additions easier.
- **NFR5.2**: All features are documented to aid future developers in system maintenance and debugging.
- **NFR5.3**: The system uses standard Django conventions, making it easier for developers familiar with the framework to contribute.

VI. Reliability and Availability

- **NFR6.1**: The system maintains 99.9% uptime, excluding scheduled maintenance periods.
- **NFR6.2**: Regular backups of the database and resource files are performed to prevent data loss.
- **NFR6.3**: In the event of failure, the system recovers gracefully with minimal downtime.

VII. Portability and Compatibility

- **NFR7.1**: The platform is compatible with major web browsers (Chrome, Firefox, Safari, Edge).

- **NFR7.2**: The UI is responsive and accessible on various devices including desktops, tablets, and smartphones.

VIII. Auditability and Logging

- **NFR8.1**: The system logs critical user and admin actions (e.g., project creation, login attempts, subscription updates).
- **NFR8.2**: Admins can review logs through the dashboard for monitoring or security purposes.

4. Software and Hardware Requirements Software

🔧Software Stack:

- **Frontend**: HTML, CSS, JavaScript, Bootstrap, Font Awesome
- **Backend**: Django (Python Framework)
- **Database**: PostgreSQL
- **Authentication**: Django Auth, GitHub OAuth, LinkedIn OAuth
- **CI/CD**: GitHub Actions, Docker
- **Deployment**:
  - **Development & Testing**: Heroku, PythonAnywhere
  - **Production**: JHUB Africa Cloud Infrastructure
- **File Handling**: Built-in Django storage with support for uploading DOCX, PPT, PDF, audio, and video files
- **Notification System**: Real-time threshold-based alerts and updates for subscribers
- **Admin Dashboard**: Real-time monitoring and statistics interface for user, project, and resource management

💻 Hardware Requirements (Server/Hosting Environment):

- **Server Specifications** (for deployment under JHUB or equivalent hosting):
  - Linux-based OS (Ubuntu preferred)
  - 2+ vCPUs

    - 4GB+ RAM
    - PostgreSQL-compatible storage engine
    - SSD storage with backup support
- **Client-Side Requirements**:
    - Internet-enabled device (Desktop, Laptop, Mobile)
    - Modern Web Browser (Chrome, Firefox, Safari, Edge)

5. System Architecture

I. Overview

The architecture of CRPR follows a modern web application design, built on a **client-server** model, with clear separation between the **frontend**, **backend**, and **database layers**. The system also leverages CI/CD pipelines for smooth development, testing, and deployment workflows. The core functionality revolves around user authentication, project management, resource handling, and real-time notifications for subscribers.

II. System Components and Flow

1. **Frontend (User Interface Layer)**:

- **Technologies**: HTML, CSS, JavaScript, Bootstrap, Font Awesome
- **Functionality**:
    - Provides a clean, responsive, and interactive user interface.
    - Handles user authentication (login, signup, password reset).
    - Displays project listings, resources, and allows CRUD operations on projects.
    - Provides real-time notifications for subscribers.

2. **Backend (Application Logic Layer)**:

- **Technologies**: Django (Python), PostgreSQL

- **Functionality**:
    - **Django Framework**: Handles routing, user authentication, database queries, and application logic.
    - **User Management**: Manages user accounts (signup, login via Django, GitHub, and LinkedIn OAuth).
    - **Project Management**: Manages project creation, editing, deleting, and CRUD functionalities.
    - **File Storage**: Handles uploading, storing, and serving resources such as PDF, DOCX, PPT, video, audio, etc.
    - **Real-time Notifications**: Sends threshold-based alerts about new projects, updates, and upcoming events.

3. **Database Layer**:

- **Technology**: PostgreSQL
- **Functionality**:
    - Stores user data, project information, uploaded resources, subscription details, and notification preferences.
    - Ensures data integrity with proper relationships between users, projects, resources, and subscriptions.

4. **Authentication Layer**:

- **Technologies**: Django Auth, GitHub OAuth, LinkedIn OAuth
- **Functionality**:
    - Handles secure authentication and authorization processes, including third-party logins.

5. **CI/CD and DevOps Layer**:

- **Technologies**: GitHub Actions, Docker
- **Functionality**:
    - **GitHub Actions**: Automates testing, building, and deployment workflows.
    - **Docker**: Ensures consistent environments for development, testing, and production.

6. **Deployment Layer**:

- **Platforms**: JHUB (Production), Heroku, PythonAnywhere (Testing)
- **Functionality**:
    - **JHUB**: The platform is deployed under JHUB for production, ensuring scalability and availability.
    - **Heroku/PythonAnywhere**: Used during development and testing for quick deployment and iteration.

7. **Admin Monitoring and Dashboard**:

- **Functionality**:
    - Provides real-time monitoring of system activity, including active users, uploaded projects, and resources.
    - Admins can manage users, projects, resources, and monitor system health from a clean, intuitive UI.

III. Data Flow Diagram (DFD)

1. **User Authentication**:
    - User visits the platform and signs up/logs in using Django Auth or third-party OAuth.
    - If login is successful, the user is redirected to the dashboard or project creation page.
2. **Project Management**:
    - Authenticated users create new projects by filling in project details (title, description, resources).
    - The frontend sends a POST request to the Django backend with the project data.
    - The backend processes the request, stores the data in PostgreSQL, and returns a confirmation.
    - The project is displayed on the user's profile page and is publicly available (if applicable).
3. **File Handling and Resource Upload**:

- Users upload resources (e.g., PDF, DOCX, video) via the frontend.
- The backend processes the file, stores it in the storage system, and associates it with the correct project in the database.

4. **Subscription and Notification**:
   - Users subscribe to receive notifications about project updates or new events.
   - The system sends threshold-based notifications via email or on-site alerts whenever relevant changes occur (e.g., new project posted, project updated).
5. **Admin Monitoring**:
   - Admins access the dashboard for real-time monitoring of user activity, project status, and subscription trends.

IV. Key System Components Interaction

1. **User** interacts with the **Frontend** to either create, view, or manage projects and subscribe to notifications.
2. **Frontend** sends requests to the **Backend** (Django) to perform CRUD operations and authenticate users.
3. **Backend** interacts with the **Database** (PostgreSQL) to fetch and store project data, user details, and resources.
4. The system sends **real-time notifications** (via Django or other methods) to users based on their subscriptions.
5. **Admin** uses the **Admin Dashboard** to monitor system activities and manage the platform.
6. CI/CD pipeline (through **GitHub Actions** and **Docker**) ensures smooth deployment and version control, automatically deploying to **Heroku/PythonAnywhere** for testing and **JHUB** for production.

6. Design Patterns

CRPR implements modern software engineering principles to ensure scalability, maintainability, security, and extensibility. Below are the key design patterns used in its architecture and development:

I. Model-View-Template (MVT) Pattern – Core Application Structure

- **Used In**: Django backend
- **Description**: Django's built-in **MVT** design pattern separates concerns effectively:
    - **Model**: Handles database schema and ORM (e.g., User, Project, Resource, Subscription)
    - **View**: Processes user requests, interacts with models, and returns rendered templates or JSON responses.
    - **Template**: HTML/CSS pages rendered with dynamic data for the frontend.

☑ **Benefits**: Clean separation of business logic, data access, and presentation. Makes the application easier to maintain and test.

II. Factory Pattern – User Authentication Providers

- **Used In**: Social login integrations (GitHub, LinkedIn, Django Auth)
- **Description**: The factory pattern helps create authentication handler objects dynamically based on the user's selected method (OAuth or Django).
- **Example**: A factory method decides whether to initiate GitHub OAuth, LinkedIn OAuth, or default Django Auth during login.

☑ **Benefits**: Simplifies authentication logic, promotes modularity and code reuse.

III. Observer Pattern – Real-Time Notifications & Subscriptions

- **Used In**: Notification system for project updates, new posts, and upcoming events.
- **Description**: Subscribed users are **observers** watching for changes in specific entities (projects, events, updates). When a change occurs, all observers are notified instantly.

☑ **Benefits**: Decouples the notification logic from core project update logic, making it easier to extend in the future.

IV. Singleton Pattern – Admin Dashboard State

- **Used In**: Admin dashboard's global state tracking for monitoring active users, resource uploads, and project counts.
- **Description**: Ensures a single instance of monitoring tools or resource counters is shared across the admin system.

☑ **Benefits**: Avoids duplication of stateful monitoring logic and ensures consistency.

V. Strategy Pattern – File Upload and Validation

- **Used In**: Resource upload system (DOCX, PDF, PPT, Audio, Video, etc.)
- **Description**: Different file types require different handling strategies (e.g., validation, storage paths, metadata extraction). The strategy pattern encapsulates this logic into interchangeable upload strategies.

☑ **Benefits**: Makes the system easily extensible for new file types or upload rules without modifying core logic.

VI. Decorator Pattern – Permission and Access Control

- **Used In**: Role-based access in views (e.g., only project owners can edit/delete).
- **Description**: Django's `@login_required`, `@permission_required`, and custom decorators are used to wrap view functions to enforce access control.

☑ **Benefits**: Keeps views clean and readable while enforcing security checks externally.

VII. Builder Pattern – Project Creation Wizard (if applicable)

- **Used In**: Multi-step project creation flow
- **Description**: Guides users through structured steps to build a project submission including title, description, files, tags, and GitHub/demo links.

☑ **Benefits**: Simplifies the user experience and ensures complete, structured data submission.

VIII. Proxy Pattern – Admin Monitoring of System Activity

- **Used In**: Admin dashboard or metrics API
- **Description**: The proxy handles access to performance stats, allowing monitoring tools to interface with the data layer indirectly.

☑ **Benefits**: Adds logging, caching, or rate-limiting capabilities without modifying core components.

🧠 Summary Table

Pattern	Purpose	Applied In
MVT (MVC variant)	App structure	Django core (models, views, templates)
Factory	Dynamic auth selection	Social login (GitHub, LinkedIn, Django)
Observer	Real-time user notifications	Project/events update alerts
Singleton	Global admin state	Dashboard stats & metrics
Strategy	Handling various file uploads	PDF, DOCX, Video, Audio resources
Decorator	Access control	Login/permission wrappers
Builder	Guided data entry	Project creation flow

Pattern	Purpose	Applied In
Proxy	Admin monitoring abstraction	Dashboard system usage interface

7. User Interface and UX Design

UI/UX design for the Computing Research Projects Repository (CRPR) is guided by clarity, simplicity, responsiveness, and accessibility. Interfaces were built using Bootstrap and styled with Font Awesome, ensuring consistency and ease of navigation. Key UI components include:

- **Login/Signup Page**: Clean forms with validation, password reset, and social logins (GitHub, LinkedIn, Django).
- **Home Page**: Searchable project listings with categories, filters, and featured research highlights.
- **Project Upload Page**: Step-by-step form supporting multiple file types (PDF, DOCX, PPT, audio, video) with real-time validation.
- **Project View**: Detailed view showing abstracts, downloads, demo links, and GitHub repositories.
- **Admin Dashboard**: Real-time monitoring of users, projects, uploads, and subscriptions with alerts and threshold notifications.

The layout uses a mobile-first design and ensures accessibility for all users, including support for screen readers and keyboard navigation.

8. Database Design

Database modelling for the Computing Research Projects Repository (CRPR) followed a **normalization-first approach**, ensuring minimal data redundancy, optimized indexing, and

efficient query performance. The system is powered by **PostgreSQL**, chosen for its robustness, scalability, and support for complex relationships. Key tables include:

- **users**: Stores user credentials, profile information, authentication method (Django, GitHub, LinkedIn), and user role (admin, contributor, subscriber).
- **projects**: Contains metadata on submitted projects including title, description, authors, supervisor, department, and project level (e.g., BSc, MSc, PhD).
- **resources**: Linked files such as PDF, DOCX, PPT, audio, or video formats uploaded per project.
- **subscriptions**: Tracks active user subscriptions, subscription type, and expiry dates.
- **notifications**: Captures real-time alerts sent to subscribers about new uploads, updates, and events.
- **events**: Stores upcoming academic events, seminars, or news tied to computing research.
- **logs**: Records user activity, system events, and administrative actions for audit purposes.

An **Entity-Relationship Diagram (ERD)** has been developed, capturing:

- **One-to-many** relationship between users and projects.
- **One-to-many** between projects and resources.
- **Many-to-many** between users and projects via subscriptions and collaboration.
- **One-to-one** between users and roles (if roles are stored separately).

This relational structure supports scalability, enhances data integrity, and aligns with CRPR's collaborative and research-focused mission.

9. Security Architecture

Security is integral to the Computing Research Projects Repository (CRPR). All communication between the frontend, backend, and database is encrypted using HTTPS to ensure data privacy and integrity. The platform supports secure authentication through

Django's built-in auth system as well as OAuth integration with GitHub and LinkedIn. Passwords are hashed using PBKDF2 with salt, and users can securely reset their credentials via token-based email links. Role-based access control (RBAC) restricts sensitive operations to authorized users, such as admins and project owners. Input validation is enforced throughout the application to prevent XSS, SQL injection, and malicious file uploads. All sessions are managed using secure, HttpOnly cookies, and Django's CSRF protection is enabled by default. Real-time monitoring and audit logging capture system activities for accountability, while threshold-based alerts notify users of key events like new project uploads or updates. CRPR's CI/CD pipeline, powered by GitHub Actions and Docker, ensures secure and consistent deployments, with secrets and credentials securely managed in the environment.

10. Scalability and Deployment Strategy

The Computing Research Projects Repository (CRPR) is built with a scalable and cloud-ready architecture, ensuring efficient performance and smooth deployment as user demand grows:

- **Docker containers** encapsulate the Django backend and PostgreSQL database, enabling consistent development and production environments with minimal configuration overhead.
- **GitHub Actions** automate continuous integration and deployment workflows, handling code testing, builds, and deployment seamlessly across environments.
- **Heroku** and **PythonAnywhere** served as staging and testing platforms during development, offering fast iterations and simplified debugging before production deployment.
- In production, CRPR is hosted under **JHUB's infrastructure**, with flexibility to scale up on cloud environments such as **AWS EC2 and RDS** if needed in the future.
- **Static files** are efficiently served using **Whitenoise**, eliminating the need for external CDN services and ensuring fast, reliable asset delivery.

- The **admin dashboard**, designed using the **Django Jazzmin** template, provides a responsive and user-friendly UI for real-time system monitoring, including user activity, project uploads, and resource management.
- **Threshold alerts and smart notifications** enhance observability by notifying subscribers and admins of new projects, updates, and upcoming events.

11. Agile Development Plan

The development of the Computing Research Projects Repository (CRPR) follows an agile methodology with bi-weekly sprints to ensure steady progress, continuous feedback, and adaptability to changing requirements. Work is divided into well-defined sprints, with key milestones outlined below:

- **Sprint 1**: **Repo Setup, Wireframes, and Project Layout**
  - Set up the repository and version control structure (GitHub).
  - Design and finalize wireframes for key pages (login, project submissions, admin dashboard).
  - Implement initial project structure and build basic UI components.
- **Sprint 2**: **Authentication System and Database Setup**
  - Implement user authentication using Django's auth system and OAuth integrations (GitHub, LinkedIn).
  - Design and implement the PostgreSQL database schema for users, projects, resources, and subscriptions.
  - Develop basic user registration, login, and password reset functionalities.
- **Sprint 3**: **Project Submission System and Admin Dashboard**
  - Build out the project submission system, allowing users to upload resources (PDF, DOCX, etc.).
  - Develop the admin dashboard using the Django Jazzmin template for managing users, projects, and resources.
  - Integrate notification system for project updates and new submissions.
- **Sprint 4**: **Alerts, Reports, and Additional Admin Features**

    - Implement threshold alerts for project updates, new uploads, and system events.
    - Develop reporting features for admin to monitor user activity, subscription status, and project statistics.
    - Add user role management and permissions for different access levels.
- **Sprint 5**: **CI/CD Setup, Security, and Documentation**
    - Set up continuous integration and deployment using GitHub Actions and Docker for automated builds and deployment.
    - Implement security features, including password hashing, CSRF protection, and session management.
    - Document the codebase, API, and deployment process to ensure easy onboarding for future developers.
- **Sprint 6**: **Final Testing, Deployment, and Presentation**
    - Conduct thorough testing (unit tests, integration tests, user acceptance tests).
    - Deploy the application to the production environment under JHUB, ensuring scalability with Docker and Whitenoise.
    - Final presentation, including demonstrations of key features and system performance.

Development Process:

- **Daily Stand-ups**: 15-minute meetings to review progress, identify roadblocks, and plan daily tasks.
- **Bi-weekly Retrospectives**: Review the sprint's progress, discuss what went well, what can be improved, and plan adjustments for the next sprint.
- **Tools**:
    - **GitHub Projects** is used for managing the repository and tracking development progress through issues and milestones.

This Agile plan ensures continuous delivery of value, promotes adaptability, and allows for close collaboration and feedback throughout the development lifecycle.

12. Risk Analysis

Risk	Mitigation Plan
Hardware Failure	Pre-test sensors; maintain spare kits and ensure backup hardware for critical components.
Integration with External Systems (e.g., GitHub, LinkedIn)	Use mock services and sandbox environments for testing integration before going live.
Performance Bottlenecks	Perform load testing, optimize database queries, and implement caching for heavy analytics.
User Adoption Challenges	Provide multilingual support, create user-friendly training materials, and offer user assistance through help desks.
Security Vulnerabilities	Implement robust security measures, including password hashing, input validation, and role-based access control. Regular vulnerability scans to identify weaknesses.
Data Loss	Regular backups and use of high-availability database setups to minimize data loss risks.
Compliance with Academic Standards	Ensure data storage and processing complies with relevant academic and legal standards, including GDPR and other data protection regulations.

13. Collaboration Tools and Workflow

The development team for CRPR uses a suite of tools designed to streamline collaboration, enhance productivity, and ensure effective communication:

- **GitHub**: For code versioning, issue tracking, and managing documentation. GitHub ensures that the development process is transparent, organized, and collaborative.

- **Trello**: For task management and sprint planning. The team uses Trello boards to manage backlogs, track tasks for each sprint, and monitor progress across features and bugs.
- **Slack**: For daily coordination and status updates. Slack serves as the primary communication tool for quick discussions, daily stand-ups, and resolving any immediate issues or questions.
- **Google Drive**: For documentation storage and sharing. Google Drive is used for collaborative document creation, including API documentation, project specifications, and sprint retrospectives.
- **Figma**: For UI/UX collaboration. Figma allows the team to collaboratively design, prototype, and iterate on the user interface, ensuring that the design aligns with user needs and project goals.

14. Conclusion

The CRPR design is grounded in a clear, user-centered approach and has been developed with scalability, maintainability, and ease of collaboration in mind. By utilizing modern web technologies, a robust backend, and cloud-native tools, CRPR is poised to provide a transformative platform for computing research projects. With a well-structured architecture and streamlined workflows, CRPR is positioned to successfully address the needs of researchers, students, and academics.