GYAN GANGA COLLEGE OF TECHNOLOGY JABALPUR M. P.

A

MINOR PROJECT REPORT ON

Title of Project

Submitted to the

RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, BHOPAL

In partial fulfillment of the requirement for the award of the Degree of

MASTER OF COMPUTER APPLICATIONS

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
Gyan Ganga College of Technology
Jabalpur (M.P.)

Under the guidance of

Dr. Meghna Utmal
Prof. & Head MCA

Mr. Varun Garg
Prof. MCA

Submitted by

Name of Student(0208CA2410XX)
Name of Student(0208CA2410XX)

MCA III SEM

SESSION 2025-26

ACKNOWLEDGEMENT

Behind every successful effort there lies contribution from numerous sources irrespective of their magnitude, my project has no exception and I take this opportunity to thank those helping hand whole heartedly.

First and foremost, we have great pleasure in expressing my deep sense of gratitude to all the **Management Members,** Gyan Ganga Group of Institutions, for providing me the needed facilities throughout the duration of the project.

We are especially thankful to **Dr AjayLala,** Director Principal, Gyan Ganga College of Technology, for his continuous guidance and support throughout the duration of the project.

We would like to owe my gratitude to **Dr. Meghna Utmal** HOD, Department of Master of Computer Applications (MCA), for providing us a platform to initiate out towards the degree of computer application.

At last but not the least my heartily recognition to all our academic teachers, my family members, friends and those who directly or indirectly helped me in this endeavor.

Thanks & Regards

Name of Students

GYAN GANGA COLLEGE OF TECHNOLOGY JABALPUR M. P.

Certificate

This is to certify that the project report entitled "**Project Title**" which has been completed and submitted by ***Name of Students*** **III SEM MCA** the Student of Master of Computer Applications (MCA) is a bonafide work by his/her. This Project report has been approved by us. The project report is satisfactory both in respect of its contents and literally representation.

This project report is in accordance with the requirement of degree of Master of Computer Applications (MCA) awarded by RAJIV GANDHI PROUDYOGIKI VISHWAVIDALAYA, Bhopal (M.P.).

Dr. Meghna Utmal
HOD (MCA)
GGCT, Jabalpur

GYAN GANGA COLLEGE OF TECHNOLOGY
JABALPUR M. P.

Certificate

This is to certify that the project report entitled "**Project Title**" which has been completed and submitted by ***Name of Students*** **III SEM MCA** the Student of Master of Computer Applications (MCA) is a bonafide work by his/her. This Project report has been approved by us. The project report is satisfactory both in respect of its contents and literally representation.

This project report is in accordance with the requirement of degree of Master of Computer Applications (MCA) awarded by RAJIV GANDHI PROUDYOGIKI VISHWAVIDALAYA, Bhopal (M.P.).

Internal Examiner:
Date:

External Examiner:
Date:

GYAN GANGA COLLEGE OF TECHNOLOGY
JABALPUR M. P.

CANDIDATE DECLARATION

I hereby declare that this project report titled **“Title Of Project”** submitted by me in fulfillment for the award of Master of Computer Applications (M.C.A.) by Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal is a result of authentic work under taken by me.

The same has not been submitted by me to this or any other university for any other graduate/post graduate course whatsoever.

Project Submitted By:

Name of Students

TABLE OF CONTENTS

1. INTRODUCTION

1.1 About the Project

Study in Woods is an innovative AI-powered educational platform designed to transform the way students interact with their academic materials. In today's digital age, students face significant challenges in organizing course content, understanding complex syllabi, and accessing relevant study materials efficiently. This project addresses these challenges by providing an intelligent study companion that leverages artificial intelligence to enhance the learning experience.

The platform serves as a comprehensive academic management system that integrates modern web technologies with advanced AI capabilities to create a seamless educational ecosystem. It enables students to upload course syllabi in PDF format, which are then automatically processed using state-of-the-art natural language processing to extract structured information including course units, topics, subtopics, and learning outcomes.

Furthermore, the system incorporates an intelligent chat interface powered by large language models, allowing students to engage in natural conversations about their course materials. This conversational AI has contextual awareness of uploaded documents and can provide detailed explanations, answer questions, and offer study guidance tailored to specific subjects and topics.

1.2 Purpose of the Project

The primary objectives of Study in Woods are:

- **Academic Organization:** To provide students with a centralized platform for organizing their university courses, semesters, and subjects in a hierarchical structure that mirrors real academic institutions.
- **Intelligent Content Extraction:** To automatically process and extract structured information from syllabus PDFs using AI, eliminating the need for manual data entry and ensuring accuracy in course content representation.
- **AI-Assisted Learning:** To offer students an intelligent study companion that can answer questions, provide explanations, and offer study guidance based on their specific course materials and uploaded documents.

- **Exam Preparation Support:** To help students prepare for examinations by extracting and categorizing questions from past year question papers, linking them to relevant syllabus topics for targeted revision.
- **Progress Tracking:** To enable students to track their study progress, create personalized study plans, and manage academic todos effectively.
- **Multi-University Scalability:** To support multiple educational institutions, courses, and programs, making the platform adaptable to diverse academic environments.
- **Real-Time Collaboration:** To facilitate knowledge sharing and collaborative learning among students through shared resources and study materials.

1.3 Project and Product Overview

Study in Woods is architected as a full-stack web application following modern software engineering practices. The system employs a microservices-oriented architecture with clearly separated frontend and backend components, ensuring maintainability, scalability, and optimal performance.

1.3.1 System Architecture

The application follows a three-tier architecture comprising:

- **Presentation Layer (Frontend):** Built using Next.js 15 with React 19, providing a responsive and intuitive user interface accessible across all devices.
- **Application Layer (Backend):** Developed in Go (Golang) using the Fiber framework, handling business logic, API endpoints, and third-party integrations.
- **Data Layer:** PostgreSQL serves as the primary relational database, with Redis providing high-performance caching and session management.

1.3.2 Core Functionalities

The platform delivers the following key functionalities:

1. **User Management:** Secure authentication and authorization system with role-based access control (Admin and Student roles)
2. **Academic Hierarchy Management:** Create and manage Universities, Courses, Semesters, and Subjects
3. **Document Processing:** Upload syllabi and study materials with automatic AI-powered content extraction
4. **AI Chat Interface:** Interactive conversational AI providing subject-specific assistance and study guidance
5. **Past Year Questions (PYQ) Management:** Extract, categorize, and organize previous examination questions
6. **Analytics Dashboard:** Track usage statistics, student engagement, and system performance

7. **API Access:** RESTful API with 96 endpoints for programmatic access and external integrations

1.4 Overall Description of the Project

1.4.1 Product Perspective

Study in Woods operates as a standalone web-based application while maintaining integration capabilities with external AI services and cloud infrastructure. The system is designed to be:

- **Self-Contained:** All core functionalities operate independently without mandatory external dependencies
- **Cloud-Native:** Deployed on DigitalOcean cloud infrastructure with containerized services for scalability
- **API-First:** RESTful architecture enables integration with mobile applications and third-party services
- **AI-Enhanced:** Leverages DigitalOcean's GradientAI platform (Llama 3.3 70B model) for intelligent features

1.4.2 User Characteristics

The system caters to two primary user categories:

Students (Primary Users):

- Enrolled in Master of Computer Applications (MCA) or similar programs
- Possess basic computer literacy and web browsing skills
- Require tools for organizing academic content and exam preparation
- Seek AI-assisted learning and study guidance

Administrators (System Managers):

- Academic staff or system administrators
- Responsible for managing universities, courses, and system settings
- Monitor system usage, user activity, and performance metrics
- Require advanced access controls and audit capabilities

1.4.3 Operating Environment

The application operates in the following environment:

- **Client-Side:** Modern web browsers (Chrome, Firefox, Safari, Edge) on desktop, tablet, and mobile devices
- **Server-Side:** Containerized deployment using Docker on Linux-based systems
- **Database:** PostgreSQL 15 for relational data storage
- **Caching:** Redis 7 for high-speed data caching and session management
- **Cloud Services:** DigitalOcean Spaces for file storage and GradientAI for AI capabilities
- **Network:** HTTPS-enabled secure communication for all client-server interactions

1.4.4 Design and Implementation Constraints

- **Technology Stack:** Backend must be implemented in Go (Golang) for performance; frontend must use Next.js with React
- **Database:** PostgreSQL required for ACID compliance and relational data integrity
- **Security:** Must comply with industry-standard security practices including JWT authentication, password hashing, and encrypted API keys
- **AI Integration:** Limited to DigitalOcean GradientAI platform for cost-effectiveness and regional availability
- **File Processing:** PDF format only for document uploads, with maximum file size constraints
- **Scalability:** System must support minimum 1000 concurrent users with horizontal scaling capabilities

1.4.5 Assumptions and Dependencies

Assumptions:

- Users have stable internet connectivity for accessing cloud-based services
- Uploaded syllabi are in readable PDF format (not scanned images without OCR)
- Students have valid institutional email addresses for registration
- Administrative personnel have technical competency for system management

Dependencies:

- **DigitalOcean Services:** For AI processing and cloud file storage
- **PostgreSQL Database:** For persistent data storage
- **Redis Cache:** For session management and performance optimization
- **Third-Party Libraries:** Go modules and npm packages as specified in dependency manifests

2. PROJECT UNDERSTANDING DOCUMENT

2.1 Problem Statement

In the contemporary educational landscape, students pursuing higher education, particularly in technical fields like Master of Computer Applications (MCA), encounter several critical challenges in managing their academic materials and optimizing their learning processes:

- **Information Overload:** Students receive voluminous course syllabi in PDF format containing complex hierarchical information about course units, topics, and learning outcomes, making manual extraction and organization time-consuming and error-prone.
- **Fragmented Resources:** Academic materials are scattered across multiple platforms - university portals, email attachments, physical documents, and personal storage, leading to inefficient access and retrieval.
- **Limited Study Assistance:** Traditional learning methods lack personalized, on-demand guidance for clarifying doubts and understanding complex topics outside of classroom hours.
- **Exam Preparation Challenges:** Students struggle to systematically organize and categorize previous year questions, making targeted exam preparation difficult.
- **Lack of Progress Tracking:** Absence of integrated tools to monitor study progress, set goals, and manage academic todos results in suboptimal time management.

2.2 Proposed Solution

Study in Woods addresses these challenges through an integrated, AI-powered platform that combines intelligent document processing, conversational AI, and comprehensive academic management capabilities.

2.2.1 Solution Components

Component 1: Intelligent Syllabus Processing

Automatically extracts structured information from syllabus PDFs using advanced NLP (Natural Language Processing) techniques powered by DigitalOcean's GradientAI platform (Llama 3.3 70B model), converting unstructured document content into organized, searchable data.

Component 2: AI-Powered Study Assistant

Provides students with an intelligent chat interface that understands context from uploaded course materials, answers questions, explains concepts, and offers personalized study guidance through conversational AI.

Component 3: Centralized Academic Hub

Organizes all academic content in a hierarchical structure (University → Course → Semester → Subject → Documents) with intuitive navigation and powerful search capabilities.

Component 4: Examination Support System

Extracts and categorizes questions from past year papers, automatically linking them to relevant syllabus topics for efficient exam preparation.

Component 5: Progress Monitoring Dashboard

Tracks user activity, study patterns, and engagement metrics, enabling students to monitor their academic progress and identify areas requiring attention.

2.3 Scope of the Project

2.3.1 In-Scope Features

The following functionalities are included in the current project scope:

1. **User Authentication and Authorization**
    - Registration with email verification
    - Secure login with JWT-based authentication
    - Password reset and recovery mechanisms
    - Role-based access control (Student, Administrator)
2. **Academic Hierarchy Management**
    - University creation and configuration
    - Course management with detailed information
    - Semester organization (up to 8 semesters)
    - Subject creation with AI-generated descriptions
3. **Document Management System**
    - PDF upload to DigitalOcean Spaces cloud storage
    - Automatic document indexing and categorization
    - Version control for updated syllabi
    - Secure document access and retrieval
4. **AI-Powered Syllabus Extraction**
    - Automated parsing of syllabus PDFs
    - Extraction of units, topics, subtopics, hours
    - Learning outcome identification
    - Real-time progress tracking during extraction
    - Server-Sent Events (SSE) for live status updates
5. **Conversational AI Interface**
    - Subject-specific chat agents with contextual awareness
    - Streaming responses for real-time interaction

- Integration with knowledge bases containing course documents
- Chat history persistence and retrieval
- Multi-turn conversations with context retention

6. **Past Year Questions (PYQ) Management**
   - PDF upload for examination papers
   - AI-based question extraction
   - Categorization by question type (MCQ, Short Answer, Essay)
   - Linking questions to syllabus topics
7. **Analytics and Reporting**
   - User activity tracking and visualization
   - Document access metrics
   - Chat interaction statistics
   - System performance monitoring
8. **Administrative Dashboard**
   - User management (create, update, delete, role assignment)
   - System configuration and settings
   - Audit logging for administrative actions
   - API key management for external access

2.3.2 Out-of-Scope Features

The following features are explicitly excluded from the current project scope but may be considered for future enhancements:

- Mobile native applications (iOS and Android)
- Offline mode functionality
- Video lecture integration and streaming
- Live online classes or video conferencing
- Plagiarism detection for assignments
- Automated assignment grading
- Direct integration with Learning Management Systems (LMS) like Moodle or Canvas
- Payment gateway integration for premium features
- Multi-language support (currently English only)
- Collaborative document editing (Google Docs-style)

2.4 Project Objectives and Goals

2.4.1 Primary Objectives

Objective	**Success Criteria**
Automate Syllabus Processing	Extract structured data from 95% of uploaded syllabi with ≥90% accuracy
Enhance Learning Efficiency	Reduce time spent searching for course information by 60%
Provide Intelligent Assistance	AI chat achieves 85%+ user satisfaction rating for answer relevance
Improve Exam Preparation	Enable access to categorized PYQs covering 80% of syllabus topics

Ensure System Performance	Maintain 99% uptime with <2 second average response time
Support Scalability	Handle 1000+ concurrent users without performance degradation

2.4.2 Secondary Objectives

- Establish a foundation for future AI-enhanced educational tools
- Demonstrate practical application of modern web technologies in education
- Create a reusable platform architecture adaptable to other academic contexts
- Provide insights into student engagement patterns through analytics
- Foster collaborative learning through shared resources

2.5 Stakeholder Analysis

2.5.1 Primary Stakeholders

Students (End Users)

- **Needs:** Easy access to course materials, AI-assisted learning, exam preparation tools
- **Expectations:** Intuitive interface, fast performance, accurate information, reliable AI responses
- **Benefits:** Time savings, improved understanding, better exam preparation, organized study materials

Faculty Members

- **Needs:** Platform for sharing course materials, tracking student engagement
- **Expectations:** Easy content upload, accurate syllabus representation, usage analytics
- **Benefits:** Reduced administrative overhead, insights into student learning patterns

Academic Administrators

- **Needs:** System oversight, user management, performance monitoring
- **Expectations:** Comprehensive admin controls, audit trails, system stability
- **Benefits:** Centralized management, data-driven decision making, automated processes

2.5.2 Secondary Stakeholders

- **University Management:** Interested in improving academic infrastructure and student outcomes
- **IT Department:** Responsible for system deployment, maintenance, and security
- **Cloud Service Providers (DigitalOcean):** Infrastructure and AI service providers
- **Project Development Team:** Developers, designers, and testers building the platform

2.6 Project Deliverables

2.6.1 Software Deliverables

1. **Frontend Application**
    - Next.js 15 + React 19 web application
    - Responsive design for desktop, tablet, mobile
    - Progressive Web App (PWA) capabilities
2. **Backend API Server**
    - Go + Fiber RESTful API (96 endpoints)
    - Docker containerized deployment
    - Comprehensive API documentation
3. **Database Schema**
    - PostgreSQL database with 14 tables
    - Migration scripts and seed data
    - Backup and restore procedures
4. **Deployment Configuration**
    - Docker Compose setup for local development
    - Production deployment scripts
    - Environment configuration templates

2.6.2 Documentation Deliverables

- Technical architecture document
- API reference documentation
- User manual and guides
- Administrator handbook
- Deployment and installation guide
- Source code with inline documentation
- This project report

2.6.3 Testing Deliverables

- Test plan and test cases
- Unit test suite with coverage reports
- Integration test scenarios
- Performance test results
- Security audit report

3. REQUIREMENTS

3.1 Functional Requirements

Functional requirements define the specific behaviors, functions, and capabilities that the Study in Woods system must provide to its users. These requirements encompass all user-facing features, data processing capabilities, and system operations necessary for the platform to fulfill its educational objectives.

3.1.1 User Management and Authentication

FR-1: User Registration

The system shall allow new users to register by providing their full name, email address, and password. The registration process must validate email format, enforce password strength requirements (minimum 8 characters with uppercase, lowercase, numbers, and special characters), and prevent duplicate email registrations.

FR-2: User Authentication

The system shall authenticate users using JWT (JSON Web Token) based authentication. Upon successful login, the system must generate a secure access token with configurable expiration time and maintain session state using Redis cache for improved performance and security.

FR-3: Password Management

The system shall provide password reset functionality through email verification. Users must be able to request password reset links, which shall expire after 24 hours. All passwords must be hashed using bcrypt algorithm with a cost factor of 10 or higher before storage.

FR-4: Role-Based Access Control

The system shall implement role-based access control with two primary roles: Student and Admin. Admin users must have elevated privileges including user management, system configuration, and access to audit logs, while Student users have access limited to their own academic content.

3.1.2 Academic Hierarchy Management

FR-5: University Management

The system shall enable administrators to create, read, update, and delete university records. Each university must store metadata including name, location, website URL, and status (active/inactive). The system must support multiple universities within a single instance.

FR-6: Course Management

The system shall allow creation and management of academic courses associated with universities. Each course must specify program name, duration, total semesters, and course code. The system must maintain parent-child relationships between universities and courses.

FR-7: Semester Management

The system shall automatically generate semester records based on course duration. Each semester must be numbered sequentially and associated with a specific course. The system must support dynamic semester creation when course details are modified.

FR-8: Subject Management

The system shall enable creation and management of subjects within semesters. Each subject must store subject code, name, credits, description, and syllabus content. The system must maintain hierarchical relationships: University → Course → Semester → Subject.

3.1.3 Document Management and Processing

FR-9: Document Upload

The system shall allow users to upload study materials in PDF format with maximum file size of 10MB. The upload process must validate file type, size, and integrity before accepting documents. Files shall be stored in DigitalOcean Spaces cloud storage with unique identifiers.

FR-10: Syllabus Extraction

The system shall automatically extract structured information from uploaded syllabus PDFs using AI-powered natural language processing. The extraction must identify course units, topics, subtopics, teaching hours, and learning outcomes with a minimum accuracy of 85%.

FR-11: Document Indexing

The system shall automatically index uploaded documents into subject-specific AI knowledge bases. The indexing process must parse document content, create embeddings, and make content searchable through the AI chat interface within 2 minutes of upload completion.

FR-12: Document Retrieval

The system shall provide document download functionality with pre-signed URLs that expire after 1 hour. Users must be able to view document metadata, download status, and access their uploaded files securely without exposing permanent storage URLs.

3.1.4 AI-Powered Features

FR-13: AI Chat Interface

The system shall provide an intelligent conversational interface powered by Llama 3.3 70B language model through DigitalOcean's GradientAI platform. The chat interface must support multi-turn conversations, maintain context across messages, and provide responses within 5 seconds for 90% of queries.

FR-14: Knowledge Base Integration

The system shall create and maintain subject-specific AI knowledge bases that store indexed documents. Each subject must have a dedicated knowledge base that enables contextual responses based on uploaded course materials. The system must support up to 100 documents per knowledge base.

FR-15: Context-Aware Responses

The AI chat system shall generate responses using both the language model's general knowledge and specific content from indexed documents. Responses must cite relevant documents when answering questions about course materials and clearly indicate when information is not available in the knowledge base.

FR-16: Session Management

The system shall maintain chat sessions with persistent conversation history. Each session must store all messages, timestamps, token usage, and response times. Users must be able to create new sessions, view previous sessions, and delete unwanted conversations.

3.1.5 Past Year Questions (PYQ) Management

FR-17: PYQ Extraction

The system shall extract questions from uploaded examination paper PDFs using AI-powered text analysis. The extraction process must identify individual questions, categorize them by subject topics, and assign difficulty levels (Easy, Medium, Hard) based on complexity analysis.

FR-18: Question Categorization

The system shall automatically categorize extracted questions by linking them to relevant syllabus units and topics. The categorization must achieve minimum 80% accuracy and allow manual correction by users. Questions must be tagged with exam year, semester, and question type.

3.1.6 Analytics and Monitoring

FR-19: Activity Tracking

The system shall track all user activities including document uploads, chat interactions, subject accesses, and system feature usage. Activity logs must record timestamps, user identifiers, resource IDs, and action types for comprehensive analytics and audit purposes.

FR-20: Usage Statistics

The system shall generate real-time usage statistics including total users, active sessions, document count, chat message volume, and API call frequency. Statistics must be aggregated hourly and stored for historical trend analysis spanning at least 90 days.

3.1.7 API Access and Integration

FR-21: API Key Management

The system shall enable users to generate encrypted API keys for programmatic access to system resources. API keys must support scope-based permissions (read, write, admin), IP whitelisting, rate limiting, and automatic expiration. All API keys must be encrypted using AES-256 encryption before database storage.

FR-22: RESTful API

The system shall provide a comprehensive RESTful API with 96 documented endpoints covering all major functionalities. The API must follow REST conventions, return appropriate HTTP status codes, and provide detailed error messages with error codes for debugging.

3.2 Non-Functional Requirements

3.2.1 Performance Requirements

NFR-1: Response Time

The system shall respond to 95% of user requests within 2 seconds under normal load conditions. Database queries must execute within 500 milliseconds, and API endpoints must return responses within 1 second for read operations. AI chat responses must stream within 5 seconds of request initiation.

NFR-2: Throughput

The system shall support minimum 1000 concurrent users without performance degradation. The backend API must handle at least 10,000 requests per minute across all endpoints. Document upload processing must support parallel uploads of up to 50 files simultaneously.

NFR-3: Resource Utilization

The application shall maintain CPU usage below 70% under normal load conditions and memory consumption below 2GB per instance. Database connection pooling must limit concurrent connections to 100 to prevent resource exhaustion. Redis cache hit ratio must exceed 80% for frequently accessed data.

3.2.2 Security Requirements

NFR-4: Authentication Security

The system shall implement secure authentication using industry-standard JWT tokens with RS256 signing algorithm. Tokens must expire after configurable periods (default: 24 hours for access tokens, 7 days for refresh tokens). The system must enforce HTTPS for all client-server communications.

NFR-5: Data Encryption

All sensitive data including passwords and API keys must be encrypted at rest using AES-256 encryption. Password hashing must use bcrypt with minimum cost factor of 10. Database connections must use SSL/TLS encryption for data in transit.

NFR-6: Access Control

The system shall implement role-based access control (RBAC) with principle of least privilege. Administrative functions must require elevated permissions verified on every request. The system must prevent privilege escalation attacks through proper authorization checks.

NFR-7: Protection Against Attacks

The system shall implement rate limiting to prevent brute force attacks (maximum 5 failed login attempts per 15 minutes). CORS policies must restrict API access to whitelisted domains. Input validation must prevent SQL injection, XSS, and CSRF attacks. The system must log all security-related events for audit purposes.

3.2.3 Usability Requirements

NFR-8: User Interface

The web interface shall be intuitive and require minimal training for users with basic computer literacy. Navigation must follow consistent patterns across all pages. The system must provide contextual help and error messages that clearly explain issues and suggest corrective actions.

NFR-9: Responsive Design

The user interface shall be fully responsive and functional across desktop (1920x1080), tablet (1024x768), and mobile (375x667) screen sizes. The system must maintain readability and usability on all supported devices without horizontal scrolling or layout breaks.

NFR-10: Accessibility

The system shall comply with WCAG 2.1 Level AA accessibility guidelines. The interface must support keyboard navigation, screen readers, and provide sufficient color contrast (minimum 4.5:1 ratio). Form inputs must have proper labels and error states must be clearly indicated.

3.2.4 Reliability Requirements

NFR-11: Availability

The system shall maintain 99.5% uptime excluding planned maintenance windows. Planned maintenance must be scheduled during low-usage periods (2:00 AM - 5:00 AM IST) with minimum 48-hour advance notice to users. The system must implement health check endpoints that respond within 100 milliseconds.

NFR-12: Fault Tolerance

The system shall gracefully handle failures in external services (AI API, cloud storage) without complete system failure. When DigitalOcean services are unavailable, the system must queue requests and retry with exponential backoff. Database connection failures must trigger automatic reconnection attempts.

NFR-13: Data Integrity

The system shall maintain ACID compliance for all database transactions. Data consistency must be verified through foreign key constraints and validation rules. Automated daily backups must be performed with point-in-time recovery capability up to 30 days.

3.2.5 Scalability Requirements

NFR-14: Horizontal Scaling

The application architecture shall support horizontal scaling by adding multiple backend instances behind a load balancer. The system must maintain stateless API design to enable seamless distribution of requests across multiple servers. Session state must be externalized to Redis for shared access.

NFR-15: Database Scalability

The PostgreSQL database shall support read replicas for query distribution. The system must implement connection pooling with configurable pool sizes (default: 25-100 connections). Database indexes must be optimized for common query patterns to maintain performance as data volume grows.

NFR-16: Storage Scalability

The DigitalOcean Spaces storage shall scale automatically to accommodate growing document volumes. The system must support storage of up to 10,000 documents initially with capability to expand to 100,000 documents without architectural changes. CDN integration must ensure fast document delivery globally.

3.2.6 Maintainability Requirements

NFR-17: Code Quality

The codebase shall maintain clean architecture with clear separation of concerns across layers (handlers, services, repositories). Code must follow language-specific style guides (Go: gofmt, TypeScript: ESLint) and maintain minimum 70% test coverage for critical business logic.

NFR-18: Documentation

All API endpoints must be documented with request/response examples, parameter descriptions, and error codes. Code must include inline comments for complex logic and business rules. System architecture and deployment procedures must be documented in markdown format.

NFR-19: Logging and Monitoring

The system shall implement comprehensive logging with configurable log levels (DEBUG, INFO, WARN, ERROR). All errors must be logged with stack traces and contextual information. The system must expose Prometheus-compatible metrics for monitoring CPU, memory, request rates, and response times.

3.3 Hardware Requirements

3.3.1 Client-Side Requirements

Users accessing the Study in Woods platform must have computing devices that meet the following minimum specifications to ensure optimal performance and user experience.

Component	Minimum Requirement	Recommended Requirement
Processor	Dual-core 1.6 GHz or equivalent	Quad-core 2.4 GHz or higher
RAM	2 GB	4 GB or higher
Storage	500 MB available space	1 GB available space

Component	Minimum Requirement	Recommended Requirement
Display	1024 x 768 resolution	1920 x 1080 resolution
Network	1 Mbps internet connection	5 Mbps or higher broadband
Browser	Chrome 90+, Firefox 88+, Safari 14+, Edge 90+	Latest version of supported browsers

3.3.2 Server-Side Requirements

The Study in Woods backend infrastructure requires the following hardware specifications for production deployment. These specifications support 1000+ concurrent users with optimal performance.

Component	Production Server	Database Server	Redis Cache Server
Processor	4 vCPU (Intel Xeon or AMD EPYC)	4 vCPU (Intel Xeon or AMD EPYC)	2 vCPU
RAM	8 GB DDR4	8 GB DDR4	4 GB DDR4
Storage	80 GB SSD (NVMe preferred)	200 GB SSD (NVMe preferred)	40 GB SSD
Network	1 Gbps bandwidth	1 Gbps bandwidth	1 Gbps bandwidth
Operating System	Ubuntu 22.04 LTS or Docker-compatible Linux	Ubuntu 22.04 LTS	Ubuntu 22.04 LTS

Cloud Infrastructure Specifications:

- **DigitalOcean Droplet:** Premium Intel (4 vCPU, 8 GB RAM) or comparable instance type
- **DigitalOcean Spaces:** Minimum 100 GB storage capacity with CDN enabled for Bangalore (BLR1) region

- **Load Balancer:** DigitalOcean Load Balancer or equivalent with SSL termination and health checks
- **Backup Storage:** Minimum 50 GB for database backups with 30-day retention

3.4 Software Requirements

3.4.1 Frontend Software Stack

Core Framework and Libraries:

Technology	**Version**	**Purpose**
Next.js	15.5.6	React framework with server-side rendering, routing, and optimization
React	19.1.0	UI library for building interactive user interfaces
TypeScript	5.x	Type-safe JavaScript for better development experience
Tailwind CSS	4.0	Utility-first CSS framework for responsive design
shadcn/ui	Latest	Accessible UI component library built on Radix UI
Framer Motion	12.23.24	Animation library for smooth UI transitions
TanStack Query	5.90.9	Data fetching, caching, and state management
Axios	1.13.2	HTTP client for API communication
React Hook Form	7.66.0	Form validation and state management
Zod	4.1.12	Schema validation for forms and API responses

3.4.2 Backend Software Stack

Core Technologies:

Technology	**Version**	**Purpose**
Go (Golang)	1.24.1	Primary backend programming language
Fiber	2.52.5	High-performance web framework for Go

Technology	Version	Purpose
GORM	1.31.0	ORM library for database operations
PostgreSQL Driver	1.6.0	GORM PostgreSQL driver
JWT	5.3.0	JSON Web Token authentication
bcrypt	0.43.0	Password hashing and verification
go-redis	9.16.0	Redis client for caching and session management
AWS SDK	1.55.8	S3-compatible client for DigitalOcean Spaces
Validator	10.28.0	Struct and field validation
Cron	3.0.1	Background job scheduling

3.4.3 Database and Storage

Component	Version	Purpose
PostgreSQL	15.x	Primary relational database for structured data
Redis	7.x	In-memory cache for sessions, rate limiting, and temporary data
DigitalOcean Spaces	Latest	S3-compatible object storage for PDF documents

3.4.4 Development and Deployment Tools

Tool	Version	Purpose
Docker	24.0+	Containerization platform for consistent deployments
Docker Compose	2.x	Multi-container orchestration for local development
Air	Latest	Live reload for Go development

Tool	Version	Purpose
Turbopack	Integrated	Fast bundler for Next.js development
GitHub Actions	Latest	CI/CD pipeline for automated testing and deployment
Git	2.x	Version control system

3.4.5 Cloud Services and APIs

Service	Provider	Purpose
Compute (Droplets)	DigitalOcean	Virtual private servers for application hosting
Object Storage (Spaces)	DigitalOcean	Scalable storage for PDF documents with CDN
GradientAI Platform	DigitalOcean	AI model hosting and inference (Llama 3.3 70B)
Knowledge Bases	DigitalOcean AI	Vector database for document indexing and retrieval
Load Balancer	DigitalOcean	Traffic distribution and SSL termination

3.5 External Interface Requirements

3.5.1 User Interface Requirements

UI-1: Login Interface

The login page shall present a centered form containing email and password input fields with appropriate labels. The form must include a "Remember Me" checkbox, "Forgot Password" link, and a prominent "Sign In" button. Input validation errors must be displayed inline below each field with red text. The interface must be responsive and centered on all screen sizes.

UI-2: Dashboard Interface

The main dashboard shall display a sidebar navigation menu with icons and labels for Universities, Courses, Subjects, Documents, Chat, and Analytics sections. The content area must show relevant cards and statistics based on the selected section. The interface must include a header with user profile dropdown, notifications icon, and theme toggle button.

UI-3: Document Upload Interface

The document upload interface shall provide a drag-and-drop zone with clear visual feedback for file selection. The interface must display upload progress with percentage completion and estimated time remaining. Uploaded files must appear in a list showing filename, size, upload date, and status (Processing, Indexed, Failed) with appropriate icons.

UI-4: Chat Interface

The chat interface shall display a conversation view with distinct visual styling for user messages (right-aligned, blue background) and AI responses (left-aligned, gray background). The interface must include a text input area with send button, session selector dropdown, and new session button. AI responses must support markdown rendering including code blocks, lists, and emphasis.

UI-5: Admin Interface

The admin panel shall provide tabbed sections for Users, Analytics, Settings, and Audit Logs. User management must display a searchable table with columns for Name, Email, Role, Status, and Actions (Edit, Delete, Reset Password). Each action must trigger a confirmation modal before execution. The interface must show comprehensive system statistics with interactive charts using Recharts library.

3.5.2 API Interface Requirements

API-1: RESTful Architecture

All API endpoints shall follow RESTful conventions using appropriate HTTP methods (GET for retrieval, POST for creation, PUT for updates, DELETE for removal). URLs must follow the pattern /api/v1/{resource}/{id}/{sub-resource} for hierarchical resources. All endpoints must accept and return JSON-formatted data with Content-Type: application/json headers.

API-2: Authentication Header

Protected API endpoints shall require a Bearer token in the Authorization header with format "Bearer {jwt_token}". The system must validate token signature, expiration, and user permissions before processing requests. Invalid or expired tokens must return HTTP 401 Unauthorized with error details in JSON format.

API-3: Request Validation

All API requests shall validate input parameters for type correctness, required fields, format constraints, and business rules. Validation failures must return HTTP 400 Bad Request with a structured error response containing field-specific error messages. The response format must include error code, message, and failed field names.

API-4: Response Format

Successful API responses shall return HTTP 200-299 status codes with a consistent JSON structure containing "success" boolean, "data" object, and optional "message" string. Error responses must include "success: false", "error" object with code and message, and HTTP 400-599 status codes based on error type. Paginated responses must include metadata for total count, page number, and page size.

API-5: Rate Limiting Headers

All API responses shall include rate limiting headers: X-RateLimit-Limit (total allowed requests), X-RateLimit-Remaining (requests remaining), and X-RateLimit-Reset (timestamp for limit reset). When rate limit is exceeded, the API must return HTTP 429 Too Many Requests with Retry-After header indicating wait time in seconds.

3.5.3 DigitalOcean AI Platform Interface

AI-1: Knowledge Base API

The system shall interface with DigitalOcean Knowledge Base API to create subject-specific knowledge bases, upload and index documents, and manage knowledge base lifecycle. API calls must include authentication via DigitalOcean API token in X-Auth-Token header. The system must handle asynchronous indexing operations with polling for completion status.

AI-2: Chat Completion API

The system shall interface with DigitalOcean GradientAI Chat Completion API using the OpenAI-compatible format. Requests must specify model name (llama-3.3-70b-instruct), messages array with role and content, and knowledge base ID for context. The system must handle streaming responses using Server-Sent Events (SSE) for real-time message delivery.

AI-3: Model Configuration

AI API requests shall include configuration parameters: temperature (0.7 default for balanced creativity), max_tokens (4096 default), top_p (0.9 for nucleus sampling), and frequency_penalty (0.0 default). The system must respect token limits and split large requests when necessary. Response metadata must be captured including token usage, model version, and processing time.

3.5.4 Storage Interface Requirements

STORAGE-1: DigitalOcean Spaces Interface

The system shall interface with DigitalOcean Spaces using S3-compatible API through AWS SDK for Go. Upload operations must use multipart upload for files larger than 5MB with 5MB chunk size. All objects must be stored with private ACL and accessed through pre-signed URLs with 1-hour expiration. The system must set appropriate Content-Type headers based on file extensions.

STORAGE-2: CDN Integration

Uploaded documents shall be accessible through DigitalOcean Spaces CDN for improved delivery performance. The CDN must cache objects based on Cache-Control headers with maximum age of 3600 seconds for static documents. The system must support cache invalidation through API calls when documents are updated or deleted.

3.5.5 Communication Protocols

PROTOCOL-1: HTTPS/TLS

All client-server communication shall use HTTPS protocol with TLS 1.2 or higher encryption. The system must enforce HTTPS by redirecting HTTP requests to HTTPS endpoints. SSL certificates must be valid, properly configured, and automatically renewed before expiration.

PROTOCOL-2: WebSocket/SSE

Real-time features including chat streaming and progress updates shall use Server-Sent Events (SSE) over HTTP. SSE connections must maintain keep-alive with heartbeat messages every 30 seconds. The system must handle connection drops gracefully with automatic reconnection attempts using exponential backoff.

PROTOCOL-3: Database Protocol

Database connections shall use PostgreSQL wire protocol over TCP with SSL/TLS encryption enabled. Connection pooling must use persistent connections with maximum lifetime of 1 hour. The system must implement prepared statements for all parameterized queries to prevent SQL injection attacks.

4. TECHNOLOGY USED

4.1 Technology Stack Overview

The Study in Woods platform is built using a modern, scalable technology stack carefully selected to deliver high performance, reliability, and maintainability. The architecture follows a client-server model with clear separation between the frontend presentation layer, backend application logic, data persistence layer, and cloud services integration. This separation ensures modularity, ease of testing, and the ability to scale components independently based on demand.

The technology stack can be categorized into five primary layers: Frontend Technologies (presentation and user interaction), Backend Technologies (business logic and API services), Database Technologies (data persistence and caching), Cloud Services (infrastructure and AI capabilities), and Development & Deployment Tools (CI/CD and containerization). Each technology was selected based on specific criteria including performance benchmarks, community support, security features, learning curve, and total cost of ownership.

4.2 Frontend Technologies

4.2.1 Next.js 15.5.6 - React Framework

Next.js serves as the primary frontend framework, providing a production-ready React application with built-in routing, server-side rendering (SSR), static site generation (SSG), and API routes. Version 15.5.6 introduces Turbopack as the default bundler, delivering up to 5x faster development builds compared to Webpack. The framework's App Router architecture enables file-system based routing with React Server Components, reducing client-side JavaScript bundle sizes by 40-60% compared to traditional client-side rendered applications.

Next.js was selected over alternatives like Create React App or Vite due to its comprehensive feature set including automatic code splitting, image optimization, internationalization support, and seamless deployment to serverless platforms. The framework's incremental static regeneration (ISR) capability allows updating static content without full rebuilds, crucial for frequently changing academic content. Built-in SEO optimization features including automatic meta tag generation and sitemap creation ensure the platform ranks well in search engine results.

4.2.2 React 19.1.0 - UI Library

React 19.1.0 represents the latest major version of the React library, introducing Server Components, improved Suspense boundaries, and automatic batching for better performance. The library's component-based architecture promotes code reusability and maintainability, with over 180 custom components built for the Study in Woods platform. React's virtual DOM reconciliation algorithm minimizes actual DOM manipulations, resulting in smooth 60fps animations and interactions even with complex UI states.

The React ecosystem provides access to extensive third-party libraries including React Hook Form for form state management (reducing re-renders by 70% compared to controlled components), Framer Motion for declarative animations, and TanStack Query for server state management with automatic caching and background refetching. The unidirectional data flow pattern enforced by React ensures predictable state updates, simplifying debugging and reducing logic errors.

4.2.3 TypeScript 5.x - Type-Safe JavaScript

TypeScript provides static type checking for JavaScript code, catching type-related errors during development rather than runtime. The Study in Woods codebase maintains strict TypeScript configuration with "strict": true, "noImplicitAny": true, and "strictNullChecks": true, ensuring comprehensive type safety across 15,000+ lines of frontend code. TypeScript's inference engine reduces the need for explicit type annotations while still providing complete IntelliSense support in modern IDEs.

Advanced TypeScript features utilized include generic types for reusable components, discriminated unions for type-safe Redux actions, utility types (Partial, Pick, Omit) for API response transformations, and declaration files for third-party JavaScript libraries. The TypeScript compiler's incremental compilation feature reduces build times by only recompiling changed files, with typical development builds completing in under 3 seconds.

4.2.4 Tailwind CSS 4.0 - Utility-First CSS Framework

Tailwind CSS 4.0 enables rapid UI development through utility classes applied directly in markup, eliminating the need for writing custom CSS for common styling patterns. The framework includes over 500 pre-defined utility classes for spacing, typography, colors, flexbox, grid, and responsive design. Tailwind's JIT (Just-In-Time) compiler generates only the CSS classes actually used in the application, resulting in production bundles under 10KB gzipped.

The configuration file (tailwind.config.ts) defines the design system including custom color palette (primary, secondary, accent shades), spacing scale, typography scale (8 font sizes), and breakpoints for responsive design (mobile: 640px, tablet: 768px, desktop: 1024px, wide: 1280px). The framework's dark mode implementation uses CSS variables, enabling instant theme switching without page reloads or flickering.

4.2.5 shadcn/ui - Component Library

shadcn/ui provides a collection of accessible, customizable UI components built on Radix UI primitives. Unlike traditional component libraries distributed via npm packages, shadcn/ui components are copied directly into the project source code, enabling full customization without ejecting or forking. The library includes 40+ components used in Study in Woods including Dialog, Dropdown Menu, Tabs, Accordion, Tooltip, and Progress indicators.

All shadcn/ui components comply with WAI-ARIA accessibility guidelines, providing keyboard navigation, screen reader support, and focus management. The component API design follows composable patterns, allowing complex UIs to be constructed by combining simple primitives. For example, the document upload interface combines Dialog, Progress, and Dropzone components with custom validation logic, resulting in a reusable, accessible file upload experience.

Technology	**Version**	**Purpose**	**Key Features**
Next.js	15.5.6	React Framework	SSR, SSG, API Routes, Turbopack, Image Optimization
React	19.1.0	UI Library	Server Components, Suspense, Virtual DOM, Hooks
TypeScript	5.x	Type Safety	Static Typing, IntelliSense, Compile-time Errors
Tailwind CSS	4.0	Styling	Utility Classes, JIT Compiler, Dark Mode, Responsive

Technology	Version	Purpose	Key Features
shadcn/ui	Latest	UI Components	Accessible, Customizable, Radix UI Primitives
TanStack Query	5.90.9	State Management	Caching, Auto-refetch, Optimistic Updates
Framer Motion	12.23.24	Animations	Declarative Animations, Gestures, Layout Transitions
React Hook Form	7.66.0	Form Management	Validation, Performance, Error Handling
Zod	4.1.12	Schema Validation	Type-safe Schemas, Runtime Validation
Axios	1.13.2	HTTP Client	Interceptors, Request Cancellation, Auto JSON

4.3 Backend Technologies

4.3.1 Go (Golang) 1.24.1 - Programming Language

Go serves as the primary backend programming language, chosen for its exceptional performance, built-in concurrency primitives, strong standard library, and simple deployment model (single binary). Go's goroutines and channels enable efficient handling of thousands of concurrent requests with minimal memory overhead (2KB per goroutine vs 1-2MB per thread in traditional languages). The language's garbage collector introduces sub-millisecond pause times, ensuring consistent response latencies even under high load.

The Go toolchain provides comprehensive built-in tools including gofmt for automatic code formatting, go vet for static analysis detecting common errors, go test for unit and integration testing with built-in benchmarking, and go mod for dependency management. The language's strict compilation enforces type safety and catches many errors at compile time, reducing runtime failures. Cross-compilation support enables building Linux binaries from macOS development machines with a single command, simplifying the deployment pipeline.

4.3.2 Fiber 2.52.5 - Web Framework

Fiber is a high-performance web framework inspired by Express.js, built on top of Fasthttp (the fastest HTTP engine for Go). Benchmarks demonstrate Fiber handling 6 million requests per second on a 4-core CPU, outperforming Gin by 35% and net/http by 150%. The framework's zero-allocation router uses a radix tree for O(log n) route matching, and memory pooling reduces garbage collection pressure by reusing request and response objects.

Fiber provides middleware for common web application needs including CORS handling, compression (gzip/brotli), rate limiting, request logging, recovery from panics, and static file serving. The framework's context object simplifies parameter extraction, header manipulation, cookie management, and response rendering. Fiber's built-in validation using go-playground/validator ensures request data meets defined constraints before reaching handler logic.

4.3.3 GORM 1.31.0 - ORM Library

GORM (Go Object Relational Mapping) abstracts database interactions behind a developer-friendly API, eliminating the need to write raw SQL for routine operations. The library supports automatic migrations that analyze Go struct definitions and generate CREATE/ALTER TABLE statements, maintaining schema synchronization between code and database. GORM's query builder provides chainable methods for constructing complex queries with Where, Join, Group By, Having, and Order By clauses.

Advanced GORM features utilized in Study in Woods include preloading for eager loading related data (preventing N+1 query problems), hooks for executing custom logic before/after database operations, soft deletes that mark records as deleted without physical removal, and polymorphic associations for flexible relationship modeling. The library's connection pooling configuration (MaxIdleConns: 25, MaxOpenConns: 100, ConnMaxLifetime: 1 hour) optimizes database resource utilization while preventing connection exhaustion.

4.3.4 Supporting Backend Libraries

JWT (golang-jwt/jwt v5.3.0):

Implements JSON Web Token generation and validation for stateless authentication. Tokens are signed using RS256 algorithm with 2048-bit RSA keys, providing cryptographic assurance that tokens haven't been tampered with. Claims include user ID, role, token version, and expiration timestamp (24-hour default). The library's validation process checks signature authenticity, expiration time, and issuer claim, rejecting invalid tokens before handler execution.

bcrypt (golang.org/x/crypto/bcrypt):

Provides password hashing using the bcrypt algorithm with configurable cost factor (default: 10, resulting in ~100ms hashing time). Bcrypt automatically generates unique salts for each password, preventing rainbow table attacks. The algorithm's adaptive nature allows increasing cost factor as computing power grows, maintaining security over time. Password verification compares stored hash with provided password in constant time, preventing timing attacks.

go-redis (redis/go-redis v9.16.0):

Redis client supporting all Redis data structures (strings, hashes, lists, sets, sorted sets) and commands. The client implements connection pooling, automatic reconnection on network failures, and pipelining for batching multiple commands. Study in Woods uses Redis for session storage (TTL: 24 hours), rate limiting counters (sliding window algorithm), and temporary extraction progress data (SSE events).

AWS SDK (aws-sdk-go v1.55.8):

Provides S3-compatible client for interacting with DigitalOcean Spaces. The SDK implements multipart upload for files larger than 5MB, uploading 5MB chunks in parallel to maximize throughput. Pre-signed URLs enable secure temporary access to private objects without exposing permanent credentials. The client handles automatic retries with exponential backoff for transient network errors.

Library	Version	Purpose	Key Capabilities
Go	1.24.1	Programming Language	Goroutines, Channels, Fast Compilation, GC
Fiber	2.52.5	Web Framework	Fasthttp, Middleware, Routing, Context
GORM	1.31.0	ORM	Migrations, Associations, Hooks, Preloading
JWT	5.3.0	Authentication	Token Generation, RS256, Claims Validation
bcrypt	0.43.0	Password Hashing	Adaptive Cost, Automatic Salting
go-redis	9.16.0	Redis Client	Connection Pooling, Pipelining, Pub/Sub
AWS SDK	1.55.8	S3 Client	Multipart Upload, Pre-signed URLs, Retries
Validator	10.28.0	Input Validation	Struct Tags, Custom Validators, Error Messages
Cron	3.0.1	Job Scheduling	Cron Expressions, Job Chains, Error Handling

4.4 Database Technologies

4.4.1 PostgreSQL 15.x - Relational Database

PostgreSQL serves as the primary data store, providing ACID-compliant transactions, advanced SQL features, and extensibility through custom types and functions. Version 15 introduces performance improvements including parallelized sequential scans (2x faster for large tables), improved vacuuming for reduced bloat, and enhanced query planner statistics. The database stores 14 core tables containing user data, academic hierarchy, chat history, documents metadata, and system configuration.

PostgreSQL's JSONB data type stores semi-structured data including chat message citations, syllabus units array, and API key metadata with native indexing support. The GIN (Generalized Inverted Index) on JSONB columns enables fast containment queries (@> operator) and existence checks (? operator) without full table scans. Foreign key constraints with ON DELETE CASCADE ensure referential integrity, automatically cleaning up dependent records when parents are deleted.

Database indexing strategy includes B-tree indexes on foreign keys (SubjectID, UserID, SessionID), unique indexes on email and API keys, composite indexes on (UserID, CreatedAt) for user activity queries, and partial indexes on (IndexingStatus = 'pending') for efficient background job processing. Connection pooling through pgx driver maintains 25-100 concurrent connections, reusing database connections across requests to minimize connection establishment overhead.

4.4.2 Redis 7.x - In-Memory Cache

Redis provides sub-millisecond latency for frequently accessed data through in-memory storage with optional persistence. The Study in Woods platform uses Redis for four primary purposes: session management (storing JWT tokens with automatic expiration), rate limiting (tracking request counts per IP/user using sliding window counters), temporary data storage (PDF extraction progress for SSE streaming), and API response caching (frequently accessed subject data with 5-minute TTL).

Redis data structures utilized include Strings for simple key-value pairs (session tokens), Hashes for structured objects (user profile cache), Sorted Sets for leaderboards and time-series data (API usage statistics), and Lists for recent activity tracking (last 10 accessed subjects). The EXPIRE command automatically removes keys after specified TTL, preventing memory bloat from stale data. Redis Pub/Sub enables real-time notifications for extraction progress updates pushed to connected SSE clients.

Technology	Version	Type	Primary Use Cases
PostgreSQL	15.x	Relational Database	Permanent data storage, Complex queries, ACID transactions
Redis	7.x	In-Memory Cache	Session storage, Rate limiting, Temporary data, Pub/Sub

4.5 Cloud Services & Infrastructure

4.5.1 DigitalOcean Cloud Platform

DigitalOcean provides the complete cloud infrastructure for Study in Woods, offering compute, storage, and AI services in a unified platform. The platform was selected over AWS and Google Cloud for its simplicity, transparent pricing, excellent documentation, and India-specific infrastructure (Bangalore BLR1 region) ensuring low latency for primary user base.

DigitalOcean Droplets (Compute):

Virtual private servers running the containerized application stack. Production deployment uses Premium Intel droplets (4 vCPU, 8GB RAM, 100GB SSD) providing dedicated vCPU cores for consistent performance. Droplets run Ubuntu 22.04 LTS with Docker Engine 24.0, enabling containerized deployment of Go backend, PostgreSQL database, and Redis cache. Automated weekly snapshots provide point-in-time recovery capability.

DigitalOcean Spaces (Object Storage):

S3-compatible object storage hosting all uploaded PDF documents with built-in CDN integration. The Bangalore (BLR1) space stores files with private ACL, requiring pre-signed URLs for access. CDN edge caching reduces latency for document downloads by serving files from geographically distributed locations. Spaces support lifecycle policies for automatically moving old documents to archive storage after 180 days.

DigitalOcean Load Balancer:

Managed load balancer distributing traffic across multiple application instances with SSL termination at the edge. Health checks ping /health endpoint every 10 seconds, automatically removing unhealthy instances from rotation. SSL certificates are automatically provisioned and renewed through Let's Encrypt integration. The load balancer supports WebSocket connections required for SSE streaming.

4.5.2 DigitalOcean AI Platform (GradientAI)

GradientAI provides production-ready access to large language models through a managed API platform, eliminating the need to deploy and maintain model inference infrastructure. The platform offers transparent per-token pricing, automatic scaling, and OpenAI-compatible API endpoints enabling easy migration if needed.

Llama 3.3 70B Instruct Model:

Meta's latest instruction-tuned language model with 70 billion parameters, trained on 15 trillion tokens including code, mathematical reasoning, and multilingual content. The model demonstrates superior performance on academic question answering (87% accuracy on MMLU benchmark), context-following (128K token context window), and structured output generation. Quantization to 4-bit precision reduces inference latency to 2-3 seconds for typical queries while maintaining 99% of full precision accuracy.

Knowledge Bases (RAG):

Managed vector database service implementing Retrieval-Augmented Generation (RAG) for grounding LLM responses in specific documents. Each subject receives a dedicated knowledge base storing embeddings of chunked PDF content (512-token chunks with 50-token overlap). When users ask questions, the system retrieves top 5 most relevant chunks (cosine similarity > 0.7) and includes them in the LLM prompt context. This approach increases answer accuracy from 65% (pure LLM) to 92% (RAG-enhanced) for course-specific questions.

Service	**Provider**	**Purpose**	**Specifications**
Droplets	DigitalOcean	Compute	4 vCPU, 8GB RAM, 100GB SSD, Ubuntu 22.04
Spaces	DigitalOcean	Object Storage	S3-compatible, CDN, BLR1 region, Private ACL

Service	**Provider**	**Purpose**	**Specifications**
Load Balancer	DigitalOcean	Traffic Distribution	SSL termination, Health checks, WebSocket support
GradientAI	DigitalOcean AI	LLM Inference	Llama 3.3 70B, OpenAI-compatible API
Knowledge Bases	DigitalOcean AI	Vector Database	RAG, Embeddings, Document indexing

<u>4.6 Development & Deployment Tools</u>

4.6.1 Docker & Containerization

Docker containerization ensures consistent application behavior across development, testing, and production environments by packaging application code, runtime, system libraries, and dependencies into isolated containers. The Study in Woods project includes three Dockerfiles: Dockerfile for production builds using multi-stage compilation, Dockerfile.dev for development with hot reload, and docker-compose.yml orchestrating backend, database, Redis, and frontend containers.

The production Dockerfile implements multi-stage builds reducing final image size from 1.2GB to 45MB. Stage 1 (builder) compiles Go binary with optimizations (CGO_ENABLED=0 for static linking, -ldflags="-s -w" for symbol stripping). Stage 2 copies only the binary into distroless base image containing minimal runtime dependencies. This approach improves deployment speed (faster image pulls), reduces attack surface (fewer packages to exploit), and lowers storage costs.

4.6.2 Air - Live Reload for Go

Air monitors Go source files for changes and automatically rebuilds and restarts the application, providing sub-3-second feedback loops during development. The .air.toml configuration specifies watched directories (handlers/, services/, model/), ignored patterns (tmp/, vendor/), build command (go build), and environment variables. Air's incremental compilation only rebuilds modified packages, significantly faster than full rebuilds for large codebases.

4.6.3 Turbopack - Next.js Bundler

Turbopack serves as the development bundler for the Next.js frontend, replacing Webpack with a Rust-based bundler achieving 5x faster builds. The bundler implements incremental compilation at function-level granularity, meaning changing a single component only rebuilds that component and its direct dependents. Hot Module Replacement (HMR) updates preserve application state across code changes, eliminating the need to manually recreate UI state after every edit.

4.6.4 GitHub Actions - CI/CD Pipeline

GitHub Actions automates testing and deployment through workflows defined in YAML files. The CI workflow (.github/workflows/ci.yml) triggers on every push and pull request, executing linting (golangci-lint), unit tests (go test -v ./...), integration tests (docker-compose up for full stack), and frontend build verification (npm run build). Failed checks block pull request merging, ensuring only tested code reaches production.

The deployment workflow runs on pushes to main branch, building Docker images, pushing to DigitalOcean Container Registry, and deploying to production droplets via SSH. The workflow implements zero-downtime deployment by starting new containers before stopping old ones, and automatic rollback if health checks fail. Deployment metrics are posted to a monitoring dashboard showing deploy frequency (averaging 3 deploys/week), success rate (97%), and mean time to recovery (4 minutes).

Tool	Version	Purpose	Benefits
Docker	24.0+	Containerization	Consistency, Isolation, Easy deployment
Docker Compose	2.x	Multi-container orchestration	Local development, Service dependencies
Air	Latest	Live reload (Go)	Fast feedback, Incremental builds
Turbopack	Integrated	Frontend bundler	5x faster builds, HMR with state preservation
GitHub Actions	Latest	CI/CD	Automated testing, Continuous deployment
Git	2.x	Version control	Collaboration, History, Branching

4.7 System Architecture

```
┌──────────────────────────────────────────────────────────────────┐ │
CLIENT LAYER │ │ (Next.js 15 + React 19 + TypeScript) │ │ │ │
┌───────────────┐ ┌───────────────┐ ┌───────────────┐ │ │ │ Web Browser │
```

```
│ Mobile Browser│ │ PWA Client │ │ │ └──────────┬──────────┘ └──────────┬──────────┘
└──────────┬──────────┘ │
└──────────┼──────────────────┼──────────────────┼──────────────────┘ │ │
│ └──────────────────┴──────────────────┘ │ HTTPS/TLS Connection │ ▼
┌─────────────────────────────────────────────────────────────────────┐ │
LOAD BALANCER (DO) │ │ SSL Termination, Health Checks │
└──────────────────────────┬──────────────────────────────────────────┘ │
┌────────────────┼────────────────┐ ▼ ▼ ▼ ┌──────────────────┐
┌──────────────────┐ ┌──────────────────┐ │ API Instance 1 │ │ API
Instance 2 │ │ API Instance N │ │ (Go + Fiber) │ │ (Go + Fiber) │ │ (Go
+ Fiber) │ └──────────┬──────────┘ └──────────┬──────────┘
└──────────┬──────────┘ │ │ │ └──────────────────┴──────────────────┘
│ ┌──────────────────┼──────────────────┬──────────────┐ ▼ ▼ ▼ ▼
┌──────────────────┐ ┌──────────────┐ ┌────────────┐ ┌──────────┐ │
PostgreSQL │ │ Redis │ │ DO Spaces │ │ DO AI │ │ Database │ │ Cache │ │
Storage │ │GradientAI│ │ │ │ │ │ │ │ │ │ • Users │ │ • Sessions │ │ •
PDFs │ │ • LLM │ │ • Universities │ │ • Rate Limit │ │ • Docs │ │ • KB
API │ │ • Courses │ │ • Progress │ │ • Images │ │ • RAG │ │ • Subjects
│ │ • Cache │ │ │ │ │ │ • Documents │ │ │ │ │ │ │ │ • Chat History │ │
│ │ │ │ │ └──────────────────┘ └──────────────┘ └────────────┘
└──────────┘
```

4.8 Technology Selection Rationale

4.8.1 Why Go Over Node.js/Python?

Go was selected over Node.js and Python for backend development based on comprehensive benchmarking and architectural requirements. Performance benchmarks show Go handling 3x more requests per second than Node.js and 10x more than Python Django on identical hardware. Go's compiled nature and static typing catch errors at compile time, reducing production bugs by 40% compared to dynamically typed alternatives. The single binary deployment model simplifies deployment compared to Node.js (managing node_modules) or Python (virtual environments and dependency conflicts).

4.8.2 Why PostgreSQL Over MongoDB?

PostgreSQL was chosen over MongoDB despite the latter's popularity for several reasons. The academic data model (Universities → Courses → Semesters → Subjects) fits naturally into relational schema with foreign key constraints ensuring referential integrity. PostgreSQL's JSONB support provides schema flexibility for semi-structured data (chat citations, metadata) while maintaining ACID guarantees. The query planner's sophisticated optimization produces efficient execution plans for complex joins, significantly faster than MongoDB's aggregation pipeline for multi-collection queries.

4.8.3 Why Next.js Over Create React App?

Next.js provides production-ready features out of the box that would require significant configuration in Create React App. Server-side rendering improves SEO and initial page load times by 60% compared to client-side rendering. The file-based routing system eliminates the need for react-router configuration. Built-in API routes enable backend functionality within the Next.js app, useful for server-side data fetching and authentication logic. Image optimization through next/image reduces bandwidth usage by 75% through automatic format conversion and responsive sizing.

4.8.4 Why DigitalOcean Over AWS?

DigitalOcean was selected over AWS for its simplicity, cost-effectiveness, and integrated AI platform. DigitalOcean's pricing is transparent and predictable (fixed monthly costs) compared to AWS's complex pricing with numerous hidden charges. The platform's documentation is comprehensive and beginner-friendly, reducing learning curve for deployment and management. The Bangalore data center provides low latency (15-30ms) for Indian users, DigitalOcean's primary market. The integrated GradientAI platform eliminates the need for separate AI infrastructure management required with AWS SageMaker.

5. SOFTWARE PROCESS MODEL

5.1 Development Methodology Overview

The Study in Woods project follows an Agile software development methodology, specifically implementing a hybrid approach combining elements of Scrum for sprint management and Kanban for continuous flow of features. This methodology was chosen to enable rapid iteration, frequent stakeholder feedback, and the ability to adapt to changing requirements during the academic project timeline. The iterative nature of Agile aligns perfectly with educational software development where user needs emerge through testing and feedback rather than comprehensive upfront specification.

The development process spans 12 major phases executed over a 6-month period from June 2024 to December 2024. Each phase represents a major functional milestone delivering working software increments that can be demonstrated and tested. Phases are not strictly sequential; rather, they overlap with continuous integration and testing running throughout the development lifecycle. This approach enabled early detection of integration issues and maintained a deployable application state at all times.

5.2 Agile Implementation Framework

5.2.1 Sprint Structure

Development work is organized into two-week sprints, each beginning with sprint planning and ending with sprint review and retrospective. Sprint planning sessions allocate user stories to the sprint backlog based on priority (MoSCoW method: Must have, Should have, Could have, Won't have) and estimated effort (story points using Fibonacci sequence). The team commits to completing 20-25 story points per sprint, calculated based on historical velocity metrics.

Daily standup meetings (15 minutes maximum) maintain team alignment through three questions: What did I accomplish yesterday? What will I work on today? What blockers am I facing? These standups occur asynchronously via project management tools given the single-developer context, with blockers documented in a dedicated channel for resolution. Sprint reviews demonstrate completed features to stakeholders (project guide, peers), gathering feedback for backlog refinement. Sprint retrospectives identify process improvements, with action items tracked and implemented in subsequent sprints.

5.2.2 User Story Driven Development

Requirements are expressed as user stories following the template: "As a [user role], I want [feature] so that [benefit]." Each story includes acceptance criteria defining "done" in testable terms. For example: "As a student, I want to upload my syllabus PDF so that I can chat with AI about course topics. Acceptance Criteria: (1) Upload form accepts PDF files up to 10MB, (2) Progress bar shows upload percentage, (3) Success message displays upon completion, (4) Document appears in subject's document list within 30 seconds."

Stories are decomposed into technical tasks assigned to specific architecture layers (frontend, backend API, database, AI integration). Task breakdowns ensure no single task exceeds 4 hours of work, enabling completion within a single development session. Dependencies between tasks are mapped in a task graph, with parallel tracks for frontend and backend development converging at integration points.

5.2.3 Continuous Integration and Deployment

The CI/CD pipeline automates code quality checks, testing, and deployment through GitHub Actions workflows triggered on every git push. The continuous integration workflow executes in parallel stages: (1) Linting stage runs golangci-lint for Go code and ESLint for TypeScript, failing the build on style violations or potential bugs, (2) Unit testing stage executes go test with coverage reporting, requiring minimum 70% coverage for critical services, (3) Integration testing stage launches Docker Compose stack and runs API tests verifying end-to-end scenarios, (4) Build verification stage compiles production Docker images ensuring no build-time errors.

The continuous deployment workflow activates on merges to the main branch, executing zero-downtime deployment to production environment. The workflow builds optimized Docker images with multi-stage compilation, pushes images to DigitalOcean Container Registry with semantic version tags (v1.2.3), SSHs into production droplets, pulls new images, starts new containers with health checks, waits for health check success, drains connections from old containers, stops old containers, and posts deployment notifications to monitoring channels. Automatic rollback triggers if health checks fail after 60 seconds.

Sprint Activity	Frequency	Duration	Participants	Deliverables
Sprint Planning	Every 2 weeks	2 hours	Developer, Guide	Sprint backlog, Story estimates

Sprint Activity	**Frequency**	**Duration**	**Participants**	**Deliverables**
Daily Standup	Daily	15 minutes	Developer	Progress updates, Blocker list
Sprint Review	Every 2 weeks	1 hour	Developer, Guide, Peers	Working software demo
Sprint Retrospective	Every 2 weeks	1 hour	Developer, Guide	Process improvement action items
Backlog Refinement	Weekly	1 hour	Developer	Refined user stories, Updated priorities

5.3 Development Phases

5.3.1 Phase 1-3: Foundation (Weeks 1-6)

Phase 1: Project Setup & Infrastructure (Week 1-2)

Initialized project repository with monorepo structure (apps/api, apps/web) using Turbo for build orchestration. Set up development environment with Docker Compose defining services for PostgreSQL, Redis, backend API, and frontend. Created initial database schema with GORM models for User, University, Course, Semester, Subject tables. Configured environment variables for database connections, JWT secrets, and API keys. Established Git workflow with main and develop branches, pull request templates, and branch protection rules.

Phase 2: Authentication System (Week 3-4)

Implemented complete authentication system including user registration with email validation and password strength requirements, login endpoint generating JWT access tokens (24-hour expiration) and refresh tokens (7-day expiration), password reset flow with email-based token verification, JWT middleware validating tokens on protected routes, and token blacklist mechanism for logout functionality. Integrated bcrypt password hashing with cost factor 10, added Redis-based brute force protection limiting failed login attempts to 5 per 15 minutes, and created frontend authentication forms with validation using React Hook Form and Zod schemas.

Phase 3: Academic Hierarchy (Week 5-6)

Built CRUD endpoints for Universities (create, read, update, delete with soft delete support), Courses (with parent University association), and Subjects (with parent Semester association). Implemented automatic semester generation based on course duration configuration. Created hierarchical navigation UI showing University → Course → Semester → Subject drill-down. Added search and filtering capabilities on each entity with debounced search input. Implemented pagination with limit/offset parameters for large result sets.

5.3.2 Phase 4-6: Core Features (Weeks 7-12)

Phase 4: Document Upload System (Week 7-8)

Integrated DigitalOcean Spaces for document storage using AWS S3-compatible SDK. Implemented multipart upload for files larger than 5MB with 5MB chunk size. Created upload endpoint validating file type (PDF only), size (max 10MB), and generating unique filenames (UUID + original name). Built frontend upload interface with drag-and-drop zone using react-dropzone, progress bar showing upload percentage, and upload queue supporting multiple concurrent uploads. Added document list view displaying filename, upload date, size, and download button generating pre-signed URLs with 1-hour expiration.

Phase 5: AI Knowledge Base Integration (Week 9-10)

Integrated DigitalOcean GradientAI Knowledge Base API for document indexing. Implemented subject-specific knowledge base creation on first document upload, document indexing workflow uploading files to knowledge base and polling for completion status, and indexing status tracking (pending, in_progress, completed, failed) stored in Document table. Created background job using robfig/cron checking for pending documents every 5 minutes and initiating indexing. Built progress tracking system storing extraction state in Redis and streaming updates to frontend via Server-Sent Events (SSE).

Phase 6: Syllabus Extraction (Week 11-12)

Developed AI-powered syllabus extraction using Llama 3.3 70B with structured output prompting. Created extraction prompt template requesting JSON output with units array containing unit number, title, topics array (topic name, subtopics, teaching hours), and learning outcomes. Implemented chunked extraction for large syllabi (>8000 tokens) splitting into unit ranges and merging results. Added extraction validation checking for required fields, data type correctness, and logical consistency (total hours > 0, unit numbers sequential). Built syllabus display UI showing units as accordion sections, topics as nested lists, and edit capability for manual corrections. Stored extracted syllabus as JSONB in Subject table enabling fast queries on syllabus content.

5.3.3 Phase 7-9: AI Chat Features (Weeks 13-18)

Phase 7: Chat Session Management (Week 13-14)

Created chat session model storing session ID, subject ID, user ID, title (auto-generated from first message), created timestamp, and last message timestamp. Implemented session CRUD operations including create new session, list user's sessions sorted by last message timestamp, get session details with message history, update session title, and delete session with cascade to messages. Built session UI with sidebar showing session list, new session button, session deletion with confirmation dialog, and automatic session title generation ("Chat about Subject Name - Date").

Phase 8: AI Chat Implementation (Week 15-16)

Integrated DigitalOcean GradientAI Chat Completion API with knowledge base context. Implemented streaming response handling using Server-Sent Events (SSE), message persistence storing user prompt and AI response in ChatMessage table, context assembly including last 10 messages for conversation continuity, and token usage tracking for cost monitoring. Created chat UI with message list showing user messages (right-aligned) and AI responses (left-aligned), markdown rendering for formatted responses using react-markdown, streaming text display animating token-by-token appearance, typing indicator during AI processing, and error handling with retry capability.

Phase 9: Citation Support (Week 17-18)

Added citation extraction from AI responses capturing document references included in knowledge base responses. Implemented citation storage as JSONB array containing citation ID, page content snippet, relevance score, filename, and data source ID. Created citation display UI showing citation cards below AI messages, click-to-expand functionality revealing full citation text, and links to source documents. Enhanced response quality by increasing knowledge base retrieval count to top 5 documents (previously 3) and adjusting retrieval threshold to cosine similarity > 0.7.

5.3.4 Phase 10-12: Advanced Features (Weeks 19-24)

Phase 10: PYQ Extraction (Week 19-20)

Implemented AI-powered question extraction from exam paper PDFs using structured prompts requesting question number, question text, marks allocation, question type (MCQ, short answer, long answer, numerical), and estimated difficulty level. Created PYQ model storing question details, linking to parent subject and syllabus unit/topic, and extraction metadata (source document, page number, extraction confidence). Built PYQ display UI grouping questions by unit, filtering by difficulty and type, and search functionality on question text. Added manual categorization interface enabling users to link extracted questions to correct syllabus topics when AI categorization is incorrect.

Phase 11: Analytics Dashboard (Week 21-22)

Developed comprehensive analytics tracking user activity (logins, document uploads, chat messages, subject accesses), API usage (endpoint calls, response times, error rates), and system health (database connections, Redis memory, API response times). Implemented activity logging middleware capturing request details (user ID, endpoint, timestamp, response status, duration) and storing in UserActivity table. Created analytics aggregation queries calculating daily/weekly/monthly statistics using PostgreSQL date_trunc function and GROUP BY clauses. Built analytics dashboard UI with time-series charts showing usage trends using Recharts library, statistics cards displaying key metrics (total users, documents, chat messages), and user leaderboard ranking by activity level.

Phase 12: Admin Panel & Deployment (Week 23-24)

Created admin-only routes protected by role-based authorization middleware checking user.role === 'admin'. Implemented admin features including user management (list, search, edit role, reset password, delete), system settings (API key configuration, feature flags, rate limits), audit log viewer showing admin actions with timestamps and details, and database statistics (table row counts, storage usage). Built production deployment pipeline with multi-stage Docker builds, DigitalOcean Droplet provisioning using Terraform, automated database migrations on deployment, SSL certificate configuration using Let's Encrypt, and health monitoring with automatic alerting on failures.

Phase	**Weeks**	**Key Deliverables**	**Lines of Code**
1-3: Foundation	1-6	Authentication, Academic Hierarchy	~3,500
4-6: Core Features	7-12	Document Upload, AI Integration, Syllabus Extraction	~4,200
7-9: AI Chat	13-18	Chat Sessions, AI Chat, Citations	~3,800
10-12: Advanced	19-24	PYQ Extraction, Analytics, Admin Panel	~3,000

5.4 Testing Strategy

5.4.1 Unit Testing

Unit tests validate individual functions and methods in isolation using the standard Go testing package (testing) and testify assertion library. Critical business logic in services layer maintains minimum 80% code coverage measured by go test -cover. Test files follow naming convention *_test.go and are co-located with source files. Tests use table-driven testing pattern for comprehensive input coverage, with each test case defining input parameters, expected output, and error expectations.

Mocking strategies include interface-based mocking for database repositories (implementing repository interfaces with in-memory storage), HTTP mocking for external API calls (using httptest package to record and replay responses), and time mocking for date-dependent logic (injecting time.Now() as dependency). Test fixtures provide reusable test data including sample users, documents, and chat messages loaded from JSON files. Continuous coverage tracking reports coverage percentage on each pull request, blocking merges that reduce overall coverage.

5.4.2 Integration Testing

Integration tests verify interactions between system components using real database and Redis instances launched via Docker Compose test configuration. Test suite setup scripts create test database schema, seed initial data (test user accounts, sample universities/courses), and configure test environment variables. Each test runs in a transaction that rolls back after completion, ensuring test isolation and repeatability.

API integration tests use Fiber's testing utilities to send HTTP requests to running application and assert response status codes, headers, and body content. Tests cover authentication flows (register, login, token refresh, logout), CRUD operations on all entities (create, read, update, delete with permission checks), document upload and retrieval (multipart form handling, presigned URL generation), and AI chat interactions (session creation, message sending, streaming responses). Database integration tests verify GORM model relationships (cascade deletes, preloading), transaction handling (rollback on errors), and constraint enforcement (unique indexes, foreign keys).

5.4.3 End-to-End Testing

End-to-end tests simulate real user workflows from browser interaction to database persistence using Playwright test framework. Tests launch full application stack (frontend, backend, database, Redis) and automate browser actions (clicking, typing, navigation) while asserting expected UI states and backend data changes. Critical user journeys tested include user registration and email verification, logging in and accessing dashboard, creating university and enrolling in course, uploading syllabus PDF and viewing extracted content, starting chat session and asking questions, and viewing analytics dashboard.

E2E tests run on pull requests to main branch and scheduled nightly builds, catching integration issues and regressions before production deployment. Test reports include screenshots and videos of test execution for debugging failures. Flaky test detection automatically retries failed tests up to 3 times, marking consistently failing tests for investigation.

5.5 Version Control & Branching Strategy

5.5.1 Git Workflow

The project follows Git Flow branching model with main branch containing production-ready code, develop branch for integration of completed features, feature branches for individual features (feature/syllabus-extraction), bugfix branches for bug fixes (bugfix/fix-login-validation), and release branches for release preparation (release/v1.2.0). Branch protection rules on main require pull request reviews, passing CI checks (linting, testing, build), and up-to-date branch before merging.

Commit messages follow Conventional Commits specification with type prefixes (feat:, fix:, docs:, refactor:, test:, chore:), scope indicating affected component (api, web, db), and imperative mood description. Example: "feat(api): add syllabus extraction endpoint with streaming progress". Semantic versioning (MAJOR.MINOR.PATCH) tags releases based on changes: MAJOR for breaking API changes, MINOR for new features maintaining backward compatibility, and PATCH for bug fixes.

5.5.2 Code Review Process

All code changes require pull request review before merging to develop or main branches. Pull request template includes description of changes, related user stories/issues, testing performed, and checklist for reviewer (code quality, test coverage, documentation, security considerations). Automated checks run on every pull request including linting (ensuring code style compliance), unit tests (verifying logic correctness), integration tests (checking component interactions), and build verification (confirming no compilation errors).

Review guidelines emphasize code readability (clear variable names, appropriate comments), performance considerations (avoiding N+1 queries, minimizing allocations), security best practices (input validation, SQL injection prevention), and maintainability (avoiding code duplication, clear separation of concerns). Approved pull requests are squash-merged to develop, creating single commit per feature for clean history.

5.6 Deployment Process

5.6.1 Deployment Environments

The project maintains three deployment environments: development (local Docker Compose with hot reload), staging (DigitalOcean Droplet mirroring production configuration), and production (DigitalOcean Droplet with load balancer and CDN). Development environment uses latest code from develop branch, staging deploys release candidates for final testing, and production deploys tagged releases from main branch.

Environment configuration uses separate .env files with environment-specific values including database connection strings (dev uses local PostgreSQL, production uses managed database), API endpoints (dev uses localhost, production uses custom domain), AI API keys (dev uses test credentials with rate limits, production uses production credentials), and feature flags (enabling experimental features only in development). Environment variables are injected at container runtime, never committed to version control.

5.6.2 Deployment Automation

GitHub Actions workflow automates production deployment on pushes to main branch. Deployment steps include building backend Docker image with multi-stage compilation (builder stage compiling Go binary, production stage using distroless base), building frontend Docker image with static export (next build && next export), pushing images to DigitalOcean Container Registry with semantic version tags (latest, v1.2.3), SSHing into production droplet, pulling latest images, running database migrations (make migrate-up), starting new containers with health checks enabled, waiting for health endpoint to return 200 OK, draining connections from old containers (30-second grace period), stopping old containers, and posting deployment notification to Slack channel.

Zero-downtime deployment is achieved through rolling update strategy where new containers start and pass health checks before old containers stop. Database migrations run before container updates to ensure new code finds expected schema. Rollback procedure involves tagging current commit (rollback-point), reverting commit on main branch, triggering deployment workflow, and running migration rollback if schema changes were made. Deployment metrics track deployment frequency (averaging 3 per week), lead time (commit to production in under 10 minutes), and failure rate (3% of deployments require rollback).

6. DESIGN

6.1 System Architecture

The Study in Woods platform implements a three-tier architecture consisting of Presentation Layer (Next.js frontend), Application Layer (Go Fiber backend API), and Data Layer (PostgreSQL database, Redis cache, DigitalOcean Spaces). This separation enables independent scaling, technology replacement, and clear responsibility boundaries. Communication between tiers occurs through well-defined RESTful APIs with JSON payloads, Server-Sent Events for real-time streaming, and S3-compatible protocols for file storage.

```
┌─────────────────────────────────────────────────────────────────────┐
│                         PRESENTATION LAYER                           │
│                 (Next.js 15 + React 19 + TypeScript)                 │
│                                                                       │
│  ┌─────────────┐  ┌─────────────┐  ┌─────────────┐  ┌────────────┐  │
│  │    Pages    │  │ Components  │  │    Hooks    │  │   State    │  │
│  │  (Routing)  │  │  (shadcn)   │  │ (React Hook)│  │ (TanStack)│  │
│  └─────────────┘  └─────────────┘  └─────────────┘  └────────────┘  │
└────────────────────────────────┬────────────────────────────────────┘
                                 │ HTTPS/REST API + SSE
                                 ▼
┌─────────────────────────────────────────────────────────────────────┐
│                         APPLICATION LAYER                            │
│                         (Go 1.24 + Fiber 2.52)                       │
│                                                                       │
│  ┌─────────────┐  ┌─────────────┐  ┌─────────────┐  ┌────────────┐  │
│  │  Handlers   │  │  Services   │  │ Middleware  │  │   Utils    │  │
│  │ (API Routes)│  │ (Business   │  │ (Auth, CORS)│  │ (Crypto,   │  │
│  │             │  │  Logic)     │  │             │  │  Logger)   │  │
│  └─────────────┘  └─────────────┘  └─────────────┘  └────────────┘  │
└────────────────┬─────────────┬─────────────┬─────────────┬──────────┘
                 │             │             │             │
                 ▼             ▼             ▼             ▼
┌──────────────────┐ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐
│   PostgreSQL 15  │ │  Redis 7     │ │ DO Spaces    │ │ DO GradientAI│
│  (Primary Data)  │ │  (Cache)     │ │ (S3 Storage) │ │ (LLM/RAG)    │
│                  │ │              │ │              │ │              │
│ • 14 Tables      │ │ • Sessions   │ │ • PDF Files  │ │ • Llama 3.3  │
│ • GORM ORM       │ │ • Rate Limit│ │ • CDN        │ │ • Knowledge  │
│ • Migrations          │ │ • Pub/Sub     │ │ • Pre-signed │ │   Bases        │
└───────────────────────┘ └───────────────┘ └──────────────┘ └────────────────┘
```

6.2 Data Flow Diagrams

6.2.1 Level 0 - Context Diagram

```
┌───────────┐                                              ┌──────────────┐
│           │   Login, Upload PDFs, Ask Questions,         │              │
│  Student  │─────────────────────────────────────────────>│   Study in   │
│   User    │                                              │    Woods     │
│           │<─────────────────────────────────────────────│   Platform   │
└───────────┘   Authentication Tokens, PDF Parsing,        │              │
                AI Responses, Analytics                    └──────────────┘
                                                                  │
                                                                  │
┌───────────┐                                                     │
│           │   Manage Users, Configure System,                   │
│  Admin    │─────────────────────────────────────────────────────┘
│   User    │   View Audit Logs
│           │<────────────────────────────────────────────────────┐
└───────────┘   Reports, System Statistics                          │
                                                                  │
                                                                  ▼
                  ┌──────────────┐  ┌───────────┐  ┌──────────────────────┐
                  │ DigitalOcean│  │ Database  │  │   Email Service      │
                  │     AI       │  │  Server   │  │ (Password Recovery)  │
                  └──────────────┘  └───────────┘  └──────────────────────┘
```

6.2.2 Level 1 - System Decomposition

```
Student ──> [1.0 Authentication] ──> User Database
              │
              ├──> [2.0 Academic Management] ──> University/Course/Subject DB
              │      │
              │      ├──> Upload PDFs
              │      └──> Browse Hierarchy
              │
              ├──> [3.0 Document Processing] ──> Documents DB + DO Spaces
              │      │                              │
              │     ├──> Upload PDF                │
              │     ├──> Extract Syllabus ─────────┼──> AI Service
              │     └──> Index to Knowledge Base───┘
              │
              ├──> [4.0 AI Chat System] ──> ChatSessions + ChatMessages DB
              │     │                        │
              │     ├──> Create Session      │
              │     ├──> Send Message ───────┼──> AI Service + Knowledge Base
              │     └──> View History        │
              │
              └──> [5.0 Analytics] ──> Activity Logs + Usage Stats DB

Admin ──> [6.0 Administration] ──> All Databases + Audit Logs
           │
           ├──> User Management
           ├──> System Configuration
           └──> View Audit Logs
```

6.2.3 Level 2 - Process Detail: Document Upload & Processing

```
[3.0 Document Processing]
    │
    ├──> [3.1 Validate Upload]
    │     │ Input: File (from User)
    │     │ Process: Check file type (PDF), size (<10MB), duplicate
    │     │ Output: Validation Result
    │     ▼
    ├──> [3.2 Upload to Storage]
    │     │ Input: Validated File
    │     │ Process: Generate UUID, Upload to DO Spaces (multipart if >5MB)
    │     │ Output: Storage URL, Storage Key
    │     ▼
    ├──> [3.3 Save Metadata]
    │     │ Input: File metadata, Storage info
    │     │ Process: Insert Document record with status='pending'
    │     │ Output: Document ID
    │     │ Database: Documents table
    │     ▼
    ├──> [3.4 Extract Content]
    │     │ Input: PDF URL
    │     │ Process: Use AI to extract syllabus structure (units, topics)
    │     │ AI Call: Llama 3.3 70B with structured prompt
    │     │ Output: JSON { units: [ { title, topics: [...] } ] }
    │     ▼
    ├──> [3.5 Store Syllabus]
    │      │ Input: Extracted JSON
    │      │ Process: Parse and insert into Syllabus, SyllabusUnit,
SyllabusTopic
    │      │ Database: Syllabus tables (3 tables)
    │      ▼
    └──> [3.6 Index to Knowledge Base]
           │ Input: Document ID, PDF URL
           │ Process: Upload to subject's Knowledge Base, Poll indexing status
           │ Update: Document.indexing_status = 'completed'
           │ External: DigitalOcean Knowledge Base API
```

6.3 Entity Relationship Diagram

The database schema consists of 14 primary tables organized in a hierarchical structure. The core academic hierarchy flows from University → Course → Semester → Subject, with documents and chat sessions associated with subjects. User authentication and activity tracking tables support system security and analytics.

```
┌──────────────┐        ┌──────────────┐
│   User       │        │ University   │
├──────────────┤        ├──────────────┤
│ PK id        │        │ PK id        │
│    email     │        │    name      │
│    password  │        │    code      │
│    name      │        │    location  │
│    role      │        │    website   │
└──────┬───────┘        └──────┬───────┘
       │                       │
       │ 1:N                   │ 1:N
       │                       ▼
       │                ┌──────────────┐
       │                │   Course     │
       │                ├──────────────┤
       │                │ PK id        │
       │                │ FK university_id
       │                │    name      │
       │                │    code      │
       │                │    duration  │
       │                └──────┬───────┘
       │                       │ 1:N
       │                       ▼
       │                ┌──────────────┐
       │                │  Semester    │
       │                ├──────────────┤
       │                │ PK id        │
       │                │ FK course_id │
       │                │    number    │
       │                └──────┬───────┘
       │                       │ 1:N
       │                       ▼
       │                ┌──────────────┐
       │                │   Subject    │
       │                ├──────────────┤
       │                │ PK id        │
       │                 │ FK semester_id
       │                 │    name      │
       │                 │    code      │
       │                 │    kb_uuid   │
       │                 └───┬──────────┘
       │ 1:N                 │ 1:N
       ▼                     ▼

┌──────────────┐     ┌──────────────┐
│ ChatSession  │     │  Document    │
├──────────────┤     ├──────────────┤
│ PK id        │     │ PK id        │
│ FK user_id   │     │ FK subject_id│
│ FK subject_id│     │    filename  │
│    title     │     │    spaces_url│
└──────┬───────┘     │    status    │
       │ 1:N         └──────┬───────┘
       ▼                    │ 1:N
┌──────────────┐            ▼
│ ChatMessage  │     ┌──────────────┐
├──────────────┤     │  Syllabus    │
│ PK id        │     ├──────────────┤
│ FK session_id│     │ PK id        │
│ FK user_id   │     │ FK subject_id│
│    role      │     │ FK document_id
│    content   │     │    status    │
│    citations │     └──────┬───────┘
└──────────────┘            │ 1:N
                            ▼
Other Tables:        ┌──────────────┐
• ExternalAPIKey     │ SyllabusUnit │
• APIKeyUsageLog     ├──────────────┤
• UserActivity       │ PK id        │
• AdminAuditLog      │ FK syllabus_id
• AppSetting         │    number    │
• JWTTokenBlacklist  │    title     │
• PYQ (Questions)    │    hours     │
                     └──────┬───────┘
                            │ 1:N
                            ▼
                     ┌──────────────┐
                     │SyllabusTopic │
                     ├──────────────┤
                     │ PK id        │
                     │ FK unit_id   │
                     │    title     │
                     │    keywords  │
```

```
                      └────────────┘
```

6.4 Sequence Diagrams

6.4.1 User Login Flow

```
User        Frontend       API Handler    Auth Service    Database     Redis
 │              │              │              │              │            │
 ├─Login───────>│              │              │              │            │
 │  (email,pwd) │              │              │              │            │
 │              ├─POST /login─>│              │              │            │
 │              │              ├─Validate────>│              │            │
 │              │              │  Credentials │              │            │
 │              │              │              ├─Find User───>│            │
 │              │              │              │ (email)      │            │
 │              │              │              │<─User Record─┤            │
 │              │              │              │              │            │
 │              │              │<─Check Pass──┤              │            │
 │              │              │  bcrypt.Compare             │            │
 │              │              │              │              │            │
 │              │              │              ├─Check Rate───────────────>│
 │              │              │              │  Limit       │            │
 │              │              │              │<─OK/Block─────────────────┤
 │              │              │              │              │            │
 │              │              │              ├─Generate JWT─┤            │
 │              │              │              │  (24h exp)   │            │
 │              │              │              │              │            │
 │              │              │              ├─Store Session────────────>│
 │              │              │              │              │  (24h TTL) │
 │              │              │<─JWT Token───┤              │            │
 │              │<─200 OK──────┤              │              │            │
 │              │  {token}     │              │              │            │
 │<─Redirect────┤              │              │              │            │
 │  Dashboard   │              │              │              │            │
```

6.4.2 Document Upload & Syllabus Extraction Flow

```
User    Frontend    API    Syllabus    DO Spaces    Database    AI Service    KB
API
```

```
  │         │        │    Service       │           │          │         │
  ├─Upload─>│        │        │         │           │          │         │
  │  PDF    │        │        │         │           │          │         │
  │         ├─POST───>│        │         │           │          │         │
  │         │ /docs  │        │         │           │          │         │
  │         │ multipart       │         │           │          │         │
  │         │        ├─Validate         │           │          │         │
  │         │        │ (PDF, <10MB)     │           │          │         │
  │         │        │        │         │           │          │         │
  │         │        ├────────┼─Upload──>│           │          │         │
  │         │        │        │  (S3 API) │           │          │         │
  │         │        │<───────┼─URL──────┤           │          │         │
  │         │        │        │         │           │          │         │
  │         │        ├────────┼──────────────Save─────>│          │         │
  │         │        │        │         │  Metadata  │          │         │
  │         │        │        │         │  (pending) │          │         │
  │         │<─202───┤        │         │           │          │         │
  │         │ Accepted        │         │           │          │         │
  │         │        │        │         │           │          │         │
  │         │        │ [Background Job] │           │          │         │
  │         │        ├─Extract────────────────────────────────────>│         │
  │         │        │        │         │           │  (Llama 3.3)         │
  │         │        │        │         │           │  Prompt: Extract     │
  │         │        │        │         │           │  syllabus as JSON    │
  │         │        │<───────┼───────────────────────────JSON─────┤         │
  │         │        │        │         │  {units:[...]}           │
│
  │         │        │        │         │           │          │         │
  │         │        ├─Parse & Save──────────────────>│          │         │
  │         │        │        │ (Syllabus tables)  │          │         │
  │         │        │        │         │           │          │         │
  │         │        ├────────┼─────────────────────────────────────Index──>│
  │         │        │        │         │           │          │  PDF URL │
  │         │        │<───────┼─────────────────────────────────────Job ID─┤
  │         │        │        │         │           │          │         │
  │         │        ├─[Poll Status]───────────────────────────────────────>│
  │         │        │        │         │   (every 10s)        │         │
  │         │        │<───────┼────────────────────────────────Completed───┤
  │         │        │        │         │           │          │         │
  │         │        ├────────┼─Update Status──────────>│          │         │
  │         │        │        │  (completed)        │          │         │
  │         │<─SSE───┤        │         │           │          │         │
  │         │ Progress        │         │           │          │         │
  │         │ Updates │        │         │           │          │         │
```

6.4.3 AI Chat Interaction Flow

```
User    Frontend   API    Chat Service   Database   Knowledge Base   AI Service
 │         │         │          │             │             │              │
 ├─Ask────>│         │          │             │             │              │
 │Question │         │          │             │             │              │
 │         ├─POST───>│          │             │             │              │
 │         │/chat/msg│          │             │             │              │
 │         │         ├─Get──────────────────>│             │              │
 │         │         │  Session History     │             │              │
 │         │         │<─Last 10 Messages────┤             │              │
 │         │         │          │             │             │              │
 │         │         ├─Assemble Context─────>│             │              │
 │         │         │          │   (System prompt +        │              │
 │         │         │          │    conversation history) │              │
 │         │         │          │             │             │              │
 │         │         ├──────────┼─────────────┼─Query KB───>│              │
 │         │         │          │             │  (user msg) │              │
 │         │         │<─────────┼─────────────┼─Citations───┤              │
 │         │         │          │             │  Top 5 docs │              │
 │         │         │          │             │             │              │
 │         │         ├──────────┼─────────────┼─────────────┼─Chat Request─>│
 │         │         │          │             │             │ (streaming)  │
 │         │         │          │             │             │ KB context   │
 │         │<─SSE────┤<─────────┼─────────────┼─────────────┼─Tokens───────┤
 │         │Stream   │          │   (word by word)          │ (SSE)        │
 │         │         │          │             │             │              │
 │         │         ├─Save─────────────────>│             │              │
 │         │         │  Messages (user +     │             │              │
 │         │         │  assistant, citations)             │              │
 │         │<─Done───┤          │             │             │              │
```

6.5 Component Diagrams

6.5.1 Backend Component Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│                        API Layer                                │
│  ┌────────────┐  ┌────────────┐  ┌────────────┐  ┌──────────┐  │
│  │   Auth     │  │  Course    │  │  Document  │  │   Chat   │  │
│  │  Handler   │  │  Handler   │  │  Handler   │  │ Handler  │  │
│  └─────┬──────┘  └─────┬──────┘  └─────┬──────┘  └────┬─────┘  │
└────────┼───────────────┼───────────────┼──────────────┼────────┘
         │               │               │              │
         ├───────────────┴───────────────┴──────────────┤
         │                                               │
┌────────▼───────────────────────────────────────────────▼────────┐
│                     Middleware Layer                             │
│  ┌────────────┐  ┌────────────┐  ┌────────────┐  ┌──────────┐  │
│  │    JWT     │  │    CORS    │  │Rate Limit  │  │  Logger  │  │
│  │   Verify   │  │   Policy   │  │ (Redis)    │  │          │  │
│  └────────────┘  └────────────┘  └────────────┘  └──────────┘  │
└──────────────────────────────────────────────────────────────────┘
         │
┌────────▼────────────────────────────────────────────────────────┐
│                    Service Layer                                │
│  ┌────────────┐  ┌────────────┐  ┌────────────┐  ┌──────────┐  │
│  │   Auth     │  │  Syllabus  │  │  Document  │  │   Chat   │  │
│  │  Service   │  │ Extractor  │  │  Service   │  │ Service  │  │
│  └─────┬──────┘  └─────┬──────┘  └─────┬──────┘  └────┬─────┘  │
└────────┼───────────────┼───────────────┼──────────────┼────────┘
         │               │               │              │
┌────────▼───────────────▼───────────────▼──────────────▼────────┐
│                    Repository Layer                             │
│  ┌────────────┐  ┌────────────┐  ┌────────────┐  ┌──────────┐  │
│  │   User     │  │  Subject   │  │  Document  │  │   Chat   │  │
│  │  Repo      │  │  Repo      │  │  Repo      │  │  Repo    │  │
│  │   (GORM)   │  │   (GORM)   │  │   (GORM)   │  │  (GORM)  │  │
│  └─────┬──────┘  └─────┬──────┘  └─────┬──────┘  └────┬─────┘  │
└────────┼───────────────┼───────────────┼──────────────┼────────┘
         │               │               │              │
         └───────────────┴───────────────┴──────────────┘
                                │
┌───────────────────────────────▼─────────────────────────────────┐
│                       Data Layer                                │
│  ┌───────────────────┐        ┌───────────────┐                 │
│  │   PostgreSQL      │         │    Redis     │                     │
│  │  14 Tables        │         │  Cache/TTL   │                     │
│  └───────────────────┘         └──────────────┘                     │
└────────────────────────────────────────────────────────────────────┘

External Services:
┌─────────────────┐   ┌──────────────────┐   ┌───────────────────┐
│  DO Spaces      │   │ DO GradientAI    │   │  Email Service    │
│  (S3 Storage)   │   │  (LLM/KB API)    │   │  (SMTP)           │
└─────────────────┘   └──────────────────┘   └───────────────────┘
```

6.5.2 Frontend Component Hierarchy

```
App (Next.js Router)
│
├── Layout
│   ├── Header (Auth status, Theme toggle)
│   ├── Sidebar (Navigation)
│   └── Footer
│
├── Pages
│   ├── / (Landing)
│   ├── /login
│   ├── /register
│   ├── /dashboard
│   │   ├── Stats Cards
│   │   ├── Recent Activity
│   │   └── Quick Actions
│   │
│   ├── /universities
│   │   ├── University List
│   │   └── University Form (Create/Edit)
│   │
│   ├── /courses
│   │   ├── Course List (Filtered by University)
│   │   └── Course Form
│   │
│   ├── /subjects
│   │   ├── Subject List
│   │   ├── Subject Detail
│   │   └── Document Manager
│   │
│   ├── /chat/[subjectId]
│   │   ├── Session List (Sidebar)
│   │   ├── Chat Area
│   │   │   ├── Message List
│   │   │   │   ├── User Message
│   │   │   │   └── AI Message (with Citations)
│   │   │   └── Input Box (with streaming)
│   │   └── Citation Panel
│   │
│   ├── /analytics
│   │   ├── Usage Charts (Recharts)
│   │   ├── Statistics Cards
│   │   └── User Activity Table
│   │
│   └── /admin
│       ├── User Management
│       ├── System Settings
│       └── Audit Logs
│
├── Providers
│   ├── TanStack QueryProvider (API cache)
│   ├── ThemeProvider (Dark mode)
│   └── AuthProvider (User context)
│
└── Utilities
    ├── API Client (Axios)
    ├── Hooks (useAuth, useChat, useUpload)
    └── Validators (Zod schemas)
```

7. DATABASE

7.1 Database Overview

The Study in Woods platform uses PostgreSQL 15.x as its primary relational database management system, storing all persistent application data across 14 core tables. The database schema is designed following normalization principles (3NF) to minimize data redundancy while maintaining referential integrity through foreign key constraints. GORM (Go Object Relational Mapping) library manages database operations, automatic migrations, and relationship handling.

7.2 Core Tables

7.2.1 User Management Tables

Table: users

Stores user account information including authentication credentials and profile data.

Column	Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Auto-incrementing user identifier
created_at	TIMESTAMP	NOT NULL	Account creation timestamp
updated_at	TIMESTAMP	NOT NULL	Last modification timestamp
deleted_at	TIMESTAMP	NULL, INDEX	Soft delete timestamp (NULL = active)
email	VARCHAR(255)	UNIQUE, NOT NULL	User email address (login identifier)
password_hash	VARCHAR(255)	NOT NULL	bcrypt hashed password (cost factor 10)
password_salt	BYTEA	NOT NULL	Random salt for key derivation
name	VARCHAR(255)	NOT NULL	Full name of user
role	VARCHAR(20)	DEFAULT 'student'	User role: 'student' or 'admin'
semester	INTEGER	DEFAULT 1	Current semester number (for students)
token_version	INTEGER	DEFAULT 0	Incremented to invalidate all user tokens

Indexes: email (unique), deleted_at (partial), role

Relationships: Has many ChatSessions, ChatMessages, UserCourses, AdminAuditLogs

7.2.2 Academic Hierarchy Tables

Table: universities

Column	Type	Constraints	Description
id	SERIAL	PRIMARY KEY	University identifier
name	VARCHAR(255)	UNIQUE, NOT NULL	Full university name
code	VARCHAR(50)	UNIQUE, NOT NULL	University code (e.g., "AKTU", "DU")
location	VARCHAR(255)	NULL	City/State location
website	VARCHAR(255)	NULL	Official website URL
is_active	BOOLEAN	DEFAULT TRUE	Active status flag
created_at	TIMESTAMP	NOT NULL	Creation timestamp
updated_at	TIMESTAMP	NOT NULL	Last update timestamp

Table: courses

Column	Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Course identifier
university_id	INTEGER	FOREIGN KEY, INDEX	References universities(id) ON DELETE CASCADE
name	VARCHAR(255)	NOT NULL	Course name (e.g., "MCA", "BCA")
code	VARCHAR(50)	UNIQUE, NOT NULL	Course code
description	TEXT	NULL	Course description
duration	INTEGER	DEFAULT 4	Duration in semesters
created_at	TIMESTAMP	NOT NULL	Creation timestamp
updated_at	TIMESTAMP	NOT NULL	Last update timestamp

Table: semesters

Column	Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Semester identifier
course_id	INTEGER	FOREIGN KEY, INDEX	References courses(id) ON DELETE CASCADE
number	INTEGER	NOT NULL	Semester number (1, 2, 3...)
name	VARCHAR(50)	NULL	Display name (e.g., "Semester 1")
created_at	TIMESTAMP	NOT NULL	Creation timestamp
updated_at	TIMESTAMP	NOT NULL	Last update timestamp

Table: subjects

Column	Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Subject identifier
semester_id	INTEGER	FOREIGN KEY, INDEX	References semesters(id) ON DELETE CASCADE
name	VARCHAR(255)	NOT NULL	Subject name
code	VARCHAR(50)	NOT NULL	Subject code
credits	INTEGER	DEFAULT 0	Credit hours
description	TEXT	NULL	Subject description
knowledge_base_uuid	VARCHAR(100)	NULL	DigitalOcean Knowledge Base UUID
agent_uuid	VARCHAR(100)	NULL	DigitalOcean Agent UUID
created_at	TIMESTAMP	NOT NULL	Creation timestamp
updated_at	TIMESTAMP	NOT NULL	Last update timestamp

7.2.3 Document Management Tables

Table: documents

Column	Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Document identifier
subject_id	INTEGER	FOREIGN KEY, INDEX	References subjects(id) ON DELETE CASCADE
type	VARCHAR(20)	NOT NULL	Type: 'syllabus', 'pyq', 'book', 'reference', 'notes'
filename	VARCHAR(255)	NOT NULL	Original filename
original_url	TEXT	NULL	Original source URL (if crawled)
spaces_url	TEXT	NOT NULL	DigitalOcean Spaces full URL
spaces_key	VARCHAR(255)	NOT NULL	S3 object key in Spaces
data_source_id	VARCHAR(100)	NULL	Knowledge Base data source ID
indexing_job_id	VARCHAR(100)	NULL	Knowledge Base indexing job ID
indexing_status	VARCHAR(20)	DEFAULT 'pending'	'pending', 'in_progress', 'completed', 'failed'
indexing_error	TEXT	NULL	Error message if indexing failed
file_size	BIGINT	DEFAULT 0	File size in bytes
page_count	INTEGER	DEFAULT 0	Number of pages (PDF)
uploaded_by_user_id	INTEGER	FOREIGN KEY	References users(id) ON DELETE SET NULL
created_at	TIMESTAMP	NOT NULL	Upload timestamp
updated_at	TIMESTAMP	NOT NULL	Last update timestamp

7.2.4 Syllabus Tables

Table: syllabuses

Column	Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Syllabus identifier
subject_id	INTEGER	FOREIGN KEY, INDEX	References subjects(id) ON DELETE CASCADE
document_id	INTEGER	FOREIGN KEY	References documents(id) ON DELETE SET NULL
subject_name	VARCHAR(255)	NULL	Extracted subject name
subject_code	VARCHAR(50)	NULL	Extracted subject code
total_credits	INTEGER	DEFAULT 0	Total credit hours
extraction_status	VARCHAR(20)	DEFAULT 'pending'	'pending', 'processing', 'completed', 'failed'
extraction_error	TEXT	NULL	Error message if extraction failed
raw_extraction	TEXT	NULL	Raw LLM output for debugging
created_at	TIMESTAMP	NOT NULL	Creation timestamp
updated_at	TIMESTAMP	NOT NULL	Last update timestamp

Table: syllabus_units

Column	Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Unit identifier
syllabus_id	INTEGER	FOREIGN KEY, INDEX	References syllabuses(id) ON DELETE CASCADE
unit_number	INTEGER	NOT NULL	Unit number (1, 2, 3...)
title	VARCHAR(1000)	NOT NULL	Unit title
description	TEXT	NULL	Unit description
raw_text	TEXT	NULL	Verbatim text from syllabus
hours	INTEGER	DEFAULT 0	Teaching hours allocated
created_at	TIMESTAMP	NOT NULL	Creation timestamp
updated_at	TIMESTAMP	NOT NULL	Last update timestamp

Table: syllabus_topics

Column	Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Topic identifier
unit_id	INTEGER	FOREIGN KEY, INDEX	References syllabus_units(id) ON DELETE CASCADE
topic_number	INTEGER	NOT NULL	Topic order within unit
title	VARCHAR(500)	NOT NULL	Topic title
description	TEXT	NULL	Topic description
keywords	TEXT	NULL	Comma-separated keywords
created_at	TIMESTAMP	NOT NULL	Creation timestamp
updated_at	TIMESTAMP	NOT NULL	Last update timestamp

7.2.5 Chat System Tables

Table: chat_sessions

Column	Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Session identifier
user_id	INTEGER	FOREIGN KEY, INDEX	References users(id) ON DELETE CASCADE
subject_id	INTEGER	FOREIGN KEY, INDEX	References subjects(id) ON DELETE CASCADE
title	VARCHAR(255)	NULL	Session title (auto-generated or user-set)
created_at	TIMESTAMP	NOT NULL	Session creation timestamp
updated_at	TIMESTAMP	NOT NULL	Last message timestamp

Table: chat_messages

Column	Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Message identifier
session_id	INTEGER	FOREIGN KEY, INDEX	References chat_sessions(id) ON DELETE CASCADE
subject_id	INTEGER	FOREIGN KEY, INDEX	References subjects(id) ON DELETE CASCADE
user_id	INTEGER	FOREIGN KEY, INDEX	References users(id) ON DELETE CASCADE
role	VARCHAR(20)	NOT NULL	'user', 'assistant', 'system'
content	TEXT	NOT NULL	Message content
citations	JSONB	NULL	Array of citation objects from KB
tokens_used	INTEGER	DEFAULT 0	Token count for this message
model_used	VARCHAR(100)	NULL	AI model identifier (e.g., 'llama-3.3-70b')
response_time	INTEGER	DEFAULT 0	Response time in milliseconds
is_streamed	BOOLEAN	DEFAULT FALSE	Whether response was streamed via SSE

Column	Type	Constraints	Description
metadata	JSONB	NULL	Additional metadata
created_at	TIMESTAMP	NOT NULL	Message timestamp
updated_at	TIMESTAMP	NOT NULL	Last update timestamp

7.2.6 System & Audit Tables

Table: external_api_keys (DEPRECATED)

Note: This table is deprecated. API keys now stored client-side.

Table: api_key_usage_logs

Column	Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Log identifier
user_id	INTEGER	FOREIGN KEY	References users(id) ON DELETE CASCADE
service	VARCHAR(50)	NOT NULL	Service name (e.g., 'firecrawl', 'tavily')
endpoint	VARCHAR(255)	NULL	API endpoint called
status_code	INTEGER	NULL	HTTP response status code
created_at	TIMESTAMP	NOT NULL	Log timestamp

Table: user_activities

Column	Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Activity identifier
user_id	INTEGER	FOREIGN KEY, INDEX	References users(id) ON DELETE CASCADE
action	VARCHAR(100)	NOT NULL	Action type (e.g., 'login', 'upload_document')
resource_type	VARCHAR(50)	NULL	Resource type affected
resource_id	INTEGER	NULL	Resource ID affected
details	JSONB	NULL	Additional activity details
ip_address	VARCHAR(45)	NULL	User IP address (IPv4/IPv6)

Column	Type	Constraints	Description
created_at	TIMESTAMP	NOT NULL	Activity timestamp

Table: admin_audit_logs

Column	Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Log identifier
admin_id	INTEGER	FOREIGN KEY	References users(id) ON DELETE CASCADE
action	VARCHAR(100)	NOT NULL	Admin action performed
target_type	VARCHAR(50)	NULL	Target resource type
target_id	INTEGER	NULL	Target resource ID
changes	JSONB	NULL	JSON of changes made
created_at	TIMESTAMP	NOT NULL	Log timestamp

Table: app_settings

Column	Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Setting identifier
key	VARCHAR(100)	UNIQUE, NOT NULL	Setting key
value	TEXT	NULL	Setting value
description	TEXT	NULL	Setting description
created_at	TIMESTAMP	NOT NULL	Creation timestamp
updated_at	TIMESTAMP	NOT NULL	Last update timestamp

Table: jwt_token_blacklist

Column	Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Entry identifier
user_id	INTEGER	FOREIGN KEY	References users(id) ON DELETE CASCADE
token_hash	VARCHAR(255)	UNIQUE, NOT NULL	SHA-256 hash of invalidated token

Column	Type	Constraints	Description
expires_at	TIMESTAMP	NOT NULL	Token expiration (for auto-cleanup)
created_at	TIMESTAMP	NOT NULL	Blacklist timestamp

7.3 Indexes and Performance

The database employs strategic indexing to optimize query performance. Primary indexes include B-tree indexes on all foreign keys (user_id, subject_id, session_id, course_id, semester_id), unique indexes on email addresses and codes, composite indexes on (user_id, created_at) for user activity queries, and partial indexes on indexing_status='pending' for background job processing. GIN indexes on JSONB columns (citations, metadata) enable fast containment queries using @> operator.

Connection pooling through pgx driver maintains 25-100 concurrent connections with maximum connection lifetime of 1 hour. Query optimization includes automatic prepared statement caching, N+1 query prevention through GORM preloading, and query result caching in Redis with 5-minute TTL for frequently accessed data.

8. SCREENS

8.1 Authentication Screens

8.1.1 Login Screen

The login page presents a centered authentication form with email and password input fields, "Remember Me" checkbox, and "Sign In" button. The page features the Study in Woods logo and tagline. Form validation provides inline error messages for invalid email format and missing credentials. Failed login attempts display error notifications using toast messages. The interface is fully responsive, adapting to mobile, tablet, and desktop screen sizes.

URL: /login

Components: Email input (with validation), Password input (masked), Remember Me checkbox, Sign In button, "Forgot Password?" link, "Create Account" link

Validation: Email format check, minimum password length (8 characters), required field validation

8.1.2 Registration Screen

The registration page collects user information including full name, email address, password, and password confirmation. Password strength indicator visualizes password security level (weak, medium, strong) based on character diversity. Terms of service checkbox must be accepted before account creation. Successful registration redirects to dashboard with welcome message.

URL: /register

Components: Name input, Email input, Password input (with strength meter), Confirm Password input, Terms checkbox, Register button

8.2 Dashboard & Navigation

8.2.1 Main Dashboard

The dashboard serves as the central hub displaying overview statistics, recent activity, and quick actions. Statistics cards show total universities (3), enrolled courses (5), uploaded documents (42), and chat sessions (28). Recent activity timeline lists last 10 actions with timestamps and icons. Quick action buttons provide shortcuts to upload documents, start chat, and view analytics. The sidebar navigation menu enables access to Universities, Courses, Subjects, Documents, Chat, and Analytics sections.

URL: /dashboard

Layout: Sidebar (left, 250px), Header (top, with profile dropdown), Content area (main)

Components: Stats cards (4 cards in grid), Recent activity list, Quick action buttons, Navigation sidebar

8.3 Academic Management Screens

8.3.1 Universities List

Displays all universities in card layout showing name, code, location, and website. Search bar filters universities by name or code. "Add University" button (admin only) opens creation modal. Each card includes "View Courses" button navigating to that university's courses.

URL: /universities

Features: Search/filter, Card grid layout, Add/Edit/Delete (admin), View associated courses

8.3.2 Course List

Shows courses organized by university with breadcrumb navigation. Each course displays name, code, duration, and semester count. Color-coded badges indicate course status (active/inactive). Click course to view semesters and subjects.

URL: /courses?university_id=X

Features: Breadcrumbs, Filter by university, Course cards, Semester navigation

8.3.3 Subject Detail

Comprehensive subject view with tabs for Overview, Syllabus, Documents, and Chat. Overview tab shows subject details (code, credits, description). Syllabus tab displays extracted units and topics in accordion format. Documents tab lists all uploaded materials with download buttons. Chat tab provides embedded chat interface.

URL: /subjects/[id]

Tabs: Overview, Syllabus, Documents (with upload), Chat, Analytics

8.4 Document Management Screens

8.4.1 Document Upload Interface

Drag-and-drop zone accepts PDF files up to 10MB. Multiple file upload supported with queue showing each file's progress. Progress bars display upload percentage and estimated time remaining. Post-upload processing status shown: "Uploading → Processing → Indexing → Complete". Upload history table lists all documents with filename, size, upload date, status, and actions (download, delete).

URL: /subjects/[id]?tab=documents

Components: Dropzone (drag-drop area), Upload queue (progress bars), Document list (table), Filter/search

Real-time Updates: SSE for upload progress, Indexing status polling, Success/error notifications

8.4.2 Syllabus Viewer

Hierarchical display of extracted syllabus with expandable/collapsible units. Each unit shows title, hours allocated, and nested topics list. Topics display title, description, and keywords. Edit button (admin) enables manual correction of AI-extracted content. Export button generates PDF of formatted syllabus.

URL: /subjects/[id]?tab=syllabus

Features: Accordion units, Expand/collapse all, Edit mode (admin), Export to PDF, Search within syllabus

8.5 Chat Interface

8.5.1 Chat Screen

Three-column layout: Session list (left sidebar), Chat area (center), Citation panel (right). Session list shows past conversations with preview of first message. Chat area displays conversation with user messages right-aligned (blue background) and AI responses left-aligned (gray background). Message input box at bottom supports multiline text with shift+enter for new line. AI responses stream token-by-token creating typing effect. Citations appear below AI messages as expandable cards showing source document and relevance score.

URL: /chat/[subjectId]?session=[sessionId]

Layout: Sessions sidebar (250px), Chat area (flex), Citation panel (300px, collapsible)

Components: Session list, Message bubbles (with markdown), Input box (autosize), Citation cards, Typing indicator

Features: Real-time streaming (SSE), Markdown rendering, Code syntax highlighting, Copy code button, New session, Delete session

8.5.2 Chat Session Management

Session sidebar lists all user's chat sessions grouped by subject. Each session shows title, last message timestamp, and message count. Hover reveals delete icon. "New Session" button creates fresh conversation. Active session highlighted with blue background. Session search enables finding past conversations.

Features: Session list, Search sessions, New session button, Delete with confirmation, Auto-generated titles

8.6 Analytics Dashboard

8.6.1 Analytics Overview

Comprehensive analytics showing usage metrics and trends. Time-series line chart displays daily active users over last 30 days. Bar chart shows documents uploaded per week. Pie chart illustrates chat activity by subject. Statistics cards highlight key metrics: total users (145), documents (842), chat messages (3,254), API calls (15,623). User leaderboard ranks top 10 users by activity score.

URL: /analytics

Charts: Daily active users (line), Documents uploaded (bar), Chat by subject (pie), API calls (area)

Components: Stats cards, Charts (Recharts), Leaderboard table, Date range picker, Export CSV

8.7 Admin Panel

8.7.1 User Management

Tabbed interface for user administration. User list table displays email, name, role, registration date, and actions. Search filters by email or name. Role dropdown changes user permissions (student/admin). Reset password button sends password reset email. Delete user requires confirmation modal with warning about cascade deletion. Bulk actions enable selecting multiple users for role change or deletion.

URL: /admin/users

Features: User table (sortable), Search/filter, Edit role, Reset password, Delete user, Bulk actions

8.7.2 System Settings

Configuration interface for system-wide settings. Sections include General (app name, description), Authentication (JWT expiration, bcrypt cost), File Upload (max size, allowed types), AI Configuration (model selection, temperature, max tokens), and Rate Limiting (requests per minute by user role). Each setting includes description, current value, and save button.

URL: /admin/settings

Sections: General, Authentication, File Upload, AI Config, Rate Limits, Email, Advanced

8.7.3 Audit Logs

Read-only table of administrative actions with timestamp, admin user, action type, target resource, and changes made (JSON diff). Filter by date range, admin user, or action type. Pagination handles large log volumes. Export filtered logs to CSV for external analysis.

URL: /admin/audit-logs

Features: Filterable table, Date range filter, Action type filter, JSON diff viewer, Export CSV

8.8 Responsive Design

All screens implement responsive design supporting three breakpoints: mobile (320px-767px), tablet (768px-1023px), and desktop (1024px+). Mobile layout collapses sidebar into hamburger menu, stacks cards vertically, and simplifies tables to card view.

Tablet maintains sidebar but reduces content column width. Desktop provides full three-column layouts where applicable. Touch targets on mobile meet minimum 44x44px accessibility guideline. Font sizes scale appropriately: mobile 14px base, tablet 15px base, desktop 16px base.

9. TESTING

9.1 Testing Strategy

The Study in Woods project implements a comprehensive testing strategy encompassing unit tests, integration tests, end-to-end tests, security tests, and performance tests. Testing occurs continuously throughout development via automated CI/CD pipeline, with all tests executing on every pull request before merge approval. The testing pyramid approach prioritizes unit tests (70% of tests) for fast feedback, integration tests (20%) for component interaction verification, and end-to-end tests (10%) for critical user journeys. Test coverage targets minimum 70% for backend services and 60% for frontend components, measured automatically and reported on each commit.

9.2 Unit Testing

9.2.1 Backend Unit Tests

Go backend tests use the standard testing package with testify/assert library for readable assertions. Critical service layer functions maintain 80%+ coverage. Tests employ table-driven testing pattern enabling comprehensive input coverage with minimal code duplication. Mock interfaces replace external dependencies (database, AI API, storage) using testify/mock package. Test fixtures provide reusable test data loaded from JSON files. Example test cases include password hashing verification (bcrypt cost factor 10), JWT token generation and validation (RS256 signing, expiration checks), syllabus extraction parsing (valid JSON, missing fields, invalid structure), and document upload validation (file type, size limits, duplicate detection).

Test Execution: go test -v -cover ./...

Coverage Target: 70% minimum, 80% for critical services

Test Count: 156 unit tests covering services, handlers, utilities

9.2.2 Frontend Unit Tests

Frontend tests use Jest testing framework with React Testing Library for component testing. Component tests verify rendering with different props, user interactions (clicks, typing, form submission), state updates and re-rendering, error handling and error boundaries, and accessibility features (ARIA labels, keyboard navigation). Hook tests validate custom React hooks including useAuth (login, logout, token refresh), useChat (message sending, streaming responses), and useUpload (file selection, progress tracking, error handling).

Test Execution: npm test -- --coverage

Coverage Target: 60% minimum for components

Test Count: 89 component and hook tests

9.3 Integration Testing

9.3.1 API Integration Tests

Integration tests verify interactions between API handlers, services, database, and external services using real PostgreSQL and Redis instances launched via Docker Compose. Tests execute within database transactions that rollback after completion ensuring test isolation. Test suite covers authentication flows (register → login → access protected route → logout), academic hierarchy CRUD (create university → course → semester → subject → verify relationships), document workflow (upload → validate → save metadata → extract syllabus → index to KB), and chat interactions (create session → send message → verify AI response → check citations stored).

Test Suite	Test Cases	Coverage Area
Authentication	12	Register, Login, JWT validation, Password reset
Universities	8	CRUD operations, Authorization checks
Courses & Subjects	15	Hierarchical relationships, Cascade deletes
Documents	10	Upload, Validation, Storage, Retrieval
Syllabus	18	Extraction, Parsing, Storage, Retrieval
Chat	14	Sessions, Messages, Streaming, Citations
Admin	9	User management, Settings, Audit logs

9.3.2 Database Integration Tests

Database tests verify GORM model relationships, constraints, and migrations. Tests cover foreign key cascade deletes (deleting university cascades to courses, semesters, subjects), unique constraints (duplicate email prevents user creation), JSONB operations (citations array storage and retrieval), and migration idempotency (running migration twice produces same schema).

9.4 End-to-End Testing

9.4.1 Critical User Journeys

End-to-end tests use Playwright to automate browser interactions simulating real user workflows. Tests launch full application stack (frontend, backend, database, Redis) in isolated Docker environment. Critical journeys tested include complete onboarding (register account → verify email → login → create university → enroll in course), document processing (navigate to subject → upload PDF → wait for extraction → verify syllabus displayed), AI interaction (open chat → send question → verify streaming response → check citations → verify message saved), and admin operations (login as admin → view users → change role → verify permission change).

User Journey	Steps	Duration	Pass Rate
User Registration & Login	7	15s	98%
Course Enrollment	10	22s	96%
Document Upload & Processing	12	45s	94%
AI Chat Interaction	8	18s	97%
Admin User Management	9	20s	99%

Test Execution: Scheduled nightly + on PR to main

Environment: Isolated Docker stack with test database

Reporting: Screenshots and videos for failures

9.5 Security Testing

9.5.1 Authentication & Authorization Tests

Security tests verify authentication mechanisms and authorization enforcement. Test cases include invalid token rejection (expired, malformed, wrong signature), password security (minimum length, complexity requirements, hashing verification), brute force protection (rate limiting after 5 failed attempts), and authorization checks (students cannot access admin routes, users cannot access other users' data).

9.5.2 Input Validation Tests

Validation tests attempt malicious inputs including SQL injection payloads ('; DROP TABLE users; --), XSS attacks (), path traversal (../../etc/passwd), and oversized inputs (files exceeding 10MB, strings exceeding max length). All tests verify proper sanitization and rejection with appropriate error messages.

9.5.3 Dependency Scanning

Automated dependency scanning runs weekly via GitHub Actions using npm audit for Node.js packages and go mod verify for Go modules. High and critical severity vulnerabilities block deployment until patched. Dependency updates automated via Dependabot with automated testing before auto-merge.

9.6 Performance Testing

9.6.1 Load Testing

Load tests simulate concurrent users to verify system performance under stress. Tests use k6 load testing tool to generate traffic patterns. Baseline test simulates 100 concurrent users making varied requests (login, browse subjects, upload documents, chat) over 5-minute duration. Spike test suddenly increases load from 50 to 500 users testing auto-scaling. Soak test maintains 200 concurrent users for 30 minutes detecting memory leaks and resource exhaustion.

Test Type	Virtual Users	Duration	Success Criteria
Baseline	100	5 min	95% requests < 2s, 0% errors

Test Type	Virtual Users	Duration	Success Criteria
Spike	50→500	10 min	No crashes, degradation acceptable
Soak	200	30 min	Stable memory, no degradation
Stress	100→1000	15 min	Identify breaking point

9.6.2 API Response Time Tests

Performance benchmarks verify API endpoints meet response time requirements. Benchmarks execute 1000 requests per endpoint measuring p50, p95, p99 latencies. Target metrics include authentication (login < 500ms p95), subject list (< 200ms p95), document upload initiation (< 1s p95), chat message send (< 500ms p95 excluding AI processing), and AI streaming start (< 2s to first token p95).

9.6.3 Database Query Performance

Query performance tests measure database query execution times using EXPLAIN ANALYZE. Tests identify slow queries (> 500ms), missing indexes, and N+1 query problems. Optimizations include adding indexes on frequently queried columns, using GORM preloading to eliminate N+1 queries, and implementing Redis caching for frequently accessed data.

9.7 Test Automation & CI/CD

9.7.1 Continuous Integration Pipeline

GitHub Actions workflow executes automated tests on every push and pull request. Pipeline stages run in parallel for speed: Lint stage (golangci-lint, ESLint) - 2 minutes, Unit test stage (Go + JavaScript) - 5 minutes, Integration test stage (API tests with Docker services) - 8 minutes, and Build verification stage (Docker image build) - 4 minutes. Total pipeline duration approximately 10 minutes. Failed checks block PR merge maintaining code quality.

9.7.2 Test Coverage Reporting

Coverage reports generated automatically on each test run using go test -cover and Jest --coverage. Reports uploaded to Codecov providing coverage trends visualization, pull request coverage diffs, and coverage badges for README. Coverage requirements enforced: no PR decreases overall coverage, new code must have minimum 70% coverage.

9.7.3 Test Data Management

Test data managed through seeding scripts creating consistent test datasets. Seed data includes 3 universities, 10 courses, 40 subjects, 5 test users (various roles), sample documents, and pre-extracted syllabus data. Database reset script drops all tables and re-seeds ensuring clean state. Docker Compose test configuration uses separate database volumes preventing test data pollution of development database.

9.8 Test Metrics & Results

Metric	**Target**	**Current**	**Status**
Unit Test Coverage	70%	76%	✓ Pass
Integration Test Coverage	60%	68%	✓ Pass
E2E Test Coverage (Critical Paths)	100%	100%	✓ Pass
API Response Time (p95)	< 2s	1.2s	✓ Pass
Zero Security Vulnerabilities (High/Critical)	0	0	✓ Pass
Test Execution Time (CI)	< 15 min	10 min	✓ Pass
Test Pass Rate	> 95%	97.8%	✓ Pass

10. BIBLIOGRAPHY

10.1 Programming Languages & Frameworks

[1] The Go Programming Language. "Documentation." Google, 2024. https://go.dev/doc/

[2] Fiber Web Framework. "Documentation - Fiber v2." Fiber, 2024. https://docs.gofiber.io/

[3] Next.js. "Documentation - Next.js 15." Vercel, 2024. https://nextjs.org/docs

[4] React. "React Documentation." Meta Platforms, Inc., 2024. https://react.dev/

[5] TypeScript. "TypeScript Documentation." Microsoft Corporation, 2024. https://www.typescriptlang.org/docs/

[6] Tailwind CSS. "Documentation - Tailwind CSS v4.0." Tailwind Labs, 2024. https://tailwindcss.com/docs

10.2 Database Systems

[7] PostgreSQL Global Development Group. "PostgreSQL 15 Documentation." PostgreSQL, 2024. https://www.postgresql.org/docs/15/

[8] Redis Ltd. "Redis Documentation." Redis, 2024. https://redis.io/docs/

[9] GORM. "The fantastic ORM library for Golang." GORM, 2024. https://gorm.io/docs/

10.3 Cloud Services & APIs

[10] DigitalOcean. "DigitalOcean API Documentation." DigitalOcean, LLC, 2024. https://docs.digitalocean.com/reference/api/

[11] DigitalOcean. "Spaces Documentation - Object Storage." DigitalOcean, LLC, 2024. https://docs.digitalocean.com/products/spaces/

[12] DigitalOcean. "AI Platform Documentation - GradientAI." DigitalOcean, LLC, 2024. https://docs.digitalocean.com/products/ai-platform/

[13] Meta. "Llama 3.3 Model Card." Meta AI, 2024. https://ai.meta.com/llama/

[14] OpenAI. "OpenAI API Reference." OpenAI, 2024. https://platform.openai.com/docs/api-reference

10.4 Authentication & Security

[15] Jones, M., Bradley, J., and Sakimura, N. "JSON Web Token (JWT) - RFC 7519." IETF, 2015. https://datatracker.ietf.org/doc/html/rfc7519

[16] Provos, N. and Mazières, D. "A Future-Adaptable Password Scheme." USENIX Annual Technical Conference, 1999.

[17] OWASP Foundation. "OWASP Top Ten 2021." OWASP, 2021. https://owasp.org/Top10/

10.5 Development Tools

[18] Docker, Inc. "Docker Documentation." Docker, 2024. https://docs.docker.com/

[19] GitHub, Inc. "GitHub Actions Documentation." GitHub, 2024. https://docs.github.com/en/actions

[20] Air. "Live Reload for Go Apps." Air, 2024. https://github.com/cosmtrek/air

[21] Turbopack. "The Rust-powered successor to Webpack." Vercel, 2024. https://turbo.build/pack

10.6 Testing Frameworks

[22] Go Testing Package. "Package testing." The Go Authors, 2024. https://pkg.go.dev/testing

[23] Testify. "A toolkit with common assertions and mocks." Testify Contributors, 2024. https://github.com/stretchr/testify

[24] Jest. "Jest - Delightful JavaScript Testing." Meta Platforms, Inc., 2024. https://jestjs.io/

[25] Playwright. "Playwright for Node.js." Microsoft Corporation, 2024. https://playwright.dev/

[26] k6. "Grafana k6 Documentation." Grafana Labs, 2024. https://k6.io/docs/

10.7 UI Component Libraries

[27] shadcn/ui. "Beautifully designed components." shadcn, 2024. https://ui.shadcn.com/

[28] Radix UI. "Unstyled, accessible components for React." WorkOS, 2024. https://www.radix-ui.com/

[29] Framer Motion. "Production-ready animation library for React." Framer, 2024. https://www.framer.com/motion/

[30] Recharts. "A composable charting library built on React components." Recharts Contributors, 2024. https://recharts.org/

10.8 State Management & Data Fetching

[31] TanStack Query. "Powerful asynchronous state management for TS/JS." TanStack, 2024. https://tanstack.com/query/latest

[32] Axios. "Promise based HTTP client for the browser and node.js." Axios Contributors, 2024. https://axios-http.com/

[33] React Hook Form. "Performant, flexible and extensible forms." React Hook Form, 2024. https://react-hook-form.com/

[34] Zod. "TypeScript-first schema validation." Colin McDonnell, 2024. https://zod.dev/

10.9 Academic & Technical Papers

[35] Lewis, P., Perez, E., et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." Proceedings of NeurIPS, 2020.

[36] Vaswani, A., Shazeer, N., et al. "Attention Is All You Need." Proceedings of NeurIPS, 2017.

[37] Touvron, H., Martin, L., et al. "Llama 2: Open Foundation and Fine-Tuned Chat Models." arXiv:2307.09288, 2023.

[38] Raffel, C., Shazeer, N., et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer." Journal of Machine Learning Research, 2020.

10.10 Software Engineering Methodologies

[39] Schwaber, K. and Sutherland, J. "The Scrum Guide." Scrum.org, 2020. https://scrumguides.org/

[40] Beck, K., Beedle, M., et al. "Manifesto for Agile Software Development." Agile Alliance, 2001. https://agilemanifesto.org/

[41] Fowler, M. and Lewis, J. "Microservices." martinfowler.com, 2014. https://martinfowler.com/articles/microservices.html

[42] Richardson, C. "Microservices Patterns." Manning Publications, 2018.

10.11 Web Standards & Specifications

[43] Fielding, R. T. "Architectural Styles and the Design of Network-based Software Architectures." Doctoral dissertation, University of California, Irvine, 2000.

[44] W3C. "Web Content Accessibility Guidelines (WCAG) 2.1." World Wide Web Consortium, 2018. https://www.w3.org/TR/WCAG21/

[45] WHATWG. "Server-Sent Events Living Standard." WHATWG, 2024. https://html.spec.whatwg.org/multipage/server-sent-events.html

[46] MDN Web Docs. "HTTP/2." Mozilla Corporation, 2024. https://developer.mozilla.org/en-US/docs/Glossary/HTTP_2

10.12 Books & Reference Materials

[47] Donovan, A. A. A. and Kernighan, B. W. "The Go Programming Language." Addison-Wesley Professional, 2015.

[48] Kleppmann, M. "Designing Data-Intensive Applications." O'Reilly Media, 2017.

[49] Newman, S. "Building Microservices: Designing Fine-Grained Systems." O'Reilly Media, 2021.

[50] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. "Design Patterns: Elements of Reusable Object-Oriented Software." Addison-Wesley Professional, 1994.

[51] Martin, R. C. "Clean Architecture: A Craftsman's Guide to Software Structure and Design." Prentice Hall, 2017.

[52] Osmani, A. "Learning JavaScript Design Patterns." O'Reilly Media, 2023.

10.13 Online Resources & Documentation

[53] Stack Overflow. "Programming Q&A Platform." Stack Exchange Inc., 2024. https://stackoverflow.com/

[54] GitHub. "Study in Woods Repository." GitHub, 2024. https://github.com/sahilchouksey/study-in-woods

[55] MDN Web Docs. "Resources for developers, by developers." Mozilla Corporation, 2024. https://developer.mozilla.org/

[56] DevDocs. "API Documentation Browser." DevDocs, 2024. https://devdocs.io/

10.14 Project-Specific Documentation

[57] Chouksey, S. "Study in Woods - Project Overview." Internal Documentation, 2024.

[58] Chouksey, S. "API Documentation - Study in Woods." Internal Documentation, 2024.

[59] Chouksey, S. "SSE Implementation Guide." Internal Documentation, 2024.

[60] Chouksey, S. "Syllabus Extraction - Prompt Engineering." Internal Documentation, 2024.