UNIFY

Student Assistant Platform

Project Documentation

CSAI 203 - Team 27

Version 1.0

December 2024

Prepared for Academic Presentation
December 2024

Contents

Chapter 1

Testing Documentation

1.1 Manual Testing

Manual testing was performed throughout the development process to verify user interface functionality and user experience:

1.1.1 User Interface Testing

- All pages tested for proper rendering
- Navigation between pages verified
- Form submissions tested
- Error messages displayed correctly
- Responsive design verified on different screen sizes

1.1.2 Feature Testing

- User registration and login
- Task creation and management
- Course registration workflow
- AI Assistant question answering
- Schedule viewing and management
- Calendar event creation
- Message sending and receiving

1.1.3 Integration Testing

- End-to-end workflows tested manually
- Database operations verified
- Session management tested
- File uploads (AI Note) tested

1.2 Automated Testing

Comprehensive automated testing was implemented using pytest and unittest frameworks:

1.2.1 Unit Testing (59+ tests)

Model Tests (6 tests)

- User model initialization and serialization
- Calendar model initialization and serialization
- Data integrity verification

Repository Tests (5 tests)

- UserRepository with mocked database
- Data retrieval methods (get_all, get_by_id, get_by_email)
- Empty result handling
- All tests use unittest.mock to isolate database calls

Factory Pattern Tests (7 tests)

- RepositoryFactory returns correct repository types
- Case-insensitive entity type handling
- Invalid type error handling
- Alternative name support (kb, chat)

Singleton Pattern Tests (4 tests)

- DatabaseConnection singleton verification
- Instance preservation across calls
- State preservation

Controller Tests (6 tests)

- AI Assistant endpoints (ask, history, categories)
- Authentication requirements
- Input validation
- Model selection (Unify vs Ollama)

Service Tests (10 tests)

- RAG Engine document retrieval
- Keyword extraction
- Answer generation
- User context retrieval
- Duplicate removal

Integration Tests (9 tests)

- Full MVC flow (Controller + Factory + Repository)
- End-to-end user operations
- Login integration
- Model creation from repository data

Error Handling Tests (8 tests)

- 404 errors (not found)
- 400 errors (bad request)
- 401 errors (unauthorized)
- 500 errors (server errors)
- Exception handling

1.2.2 Non-Functional Requirements Testing (24 tests)

NFR1: Performance (3 tests)

Response time $\leq$ 2 seconds verification, Login endpoint <500ms, AI Assistant <500ms (Unify Model), User list <500ms

NFR2: Reliability (3 tests)

Error handling without crashes, Database connection reliability, Exception handling verification

NFR3: Security (4 tests)

SHA-256 password hashing verification, Hash format validation (64-char hex), No plaintext storage, Sensitive data encryption

NFR4: Scalability (3 tests)

Concurrent user requests (50+ tested, scalable to 500), No deadlocks on concurrent access, Thread-safe singleton pattern

NFR5: Integrity (3 tests)

Data consistency on create (100%), Data consistency on read (100%), Calendar sync accuracy (100%)

NFR6: Usability (4 tests)

Standard JSON format verification, Standard HTML structure, Proper Content-Type headers, CORS support

NFR7: Availability (4 tests)

Data backup capability, Data recovery capability, Chat history backup, Knowledge base backup

1.2.3 Test Automation

- **Test Runner:** run_tests.py - Automatic test discovery, comprehensive report generation, pass/fail summary
- **Pytest Configuration:** pytest.ini - Test path configuration, verbose output, test pattern matching
- **Test Fixtures:** conftest.py - Flask app fixture, test client fixture, authenticated client fixture

1.3 Test Coverage Summary

1.3.1 Overall Statistics

Metric	Count
Total Test Files	12
Total Tests	83+

1.3.2 Functional Tests: 59+ tests

Category	Test Count
Models	6 tests
Repositories	5 tests (with mocks)
Factory	7 tests
Singleton	4 tests
Controllers	6 tests
Services	10 tests
Integration	9 tests
Error Handling	8 tests
Authentication	4 tests

1.3.3 Non-Functional Tests: 24 tests

Category	Test Count
Performance	3 tests
Reliability	3 tests
Security	4 tests
Scalability	3 tests
Integrity	3 tests
Usability	4 tests
Availability	4 tests

1.3.4 Test Execution

```
Command: pytest tests/ -v
Or: python tests/run_tests.py
```

Chapter 2

User Documentation

2.1 Installation Guide

2.1.1 Prerequisites

- Python 3.8 or higher
- SQL Server (or MySQL)
- pip (Python package manager)
- Git (optional, for cloning)

2.1.2 Step 1: Clone or Download the Project

```
Navigate to project directory:
c:\Users\Acer\Desktop\Unify\UNIFY
```

2.1.3 Step 2: Create Virtual Environment (Recommended)

```
python -m venv venv
venv\Scripts\activate  (Windows)
source venv/bin/activate  (Linux/Mac)
```

2.1.4 Step 3: Install Dependencies

```
pip install -r requirements.txt
```

Required packages include: Flask, pyodbc, requests, python-docx, PyPDF2, transformers, pytest, pytest-flask

2.1.5 Step 4: Database Setup

- Ensure SQL Server is running
- Create database named 'unify'

- Run database schema: src/database/schema.sql
- Or use setup script: python src/utils/setup_backend.py

2.1.6 Step 5: Configuration

- Copy ENV_TEMPLATE.txt to .env
- Configure database connection in .env

```
DB_HOST=DESKTOP-V6DPJFP\SQLEXPRESS
DB_NAME=unify
SECRET_KEY=your-secret-key-here
```

2.2 Running the Application

2.2.1 Option 1: Using app.py (Root)

```
cd c:\Users\Acer\Desktop\Unify\UNIFY
python app.py
```

2.2.2 Option 2: Using src/app.py

```
cd c:\Users\Acer\Desktop\Unify\UNIFY\src
python app.py
```

2.2.3 Application Configuration

The application will start on: `http://localhost:5000`

Default Configuration:

- Host: 0.0.0.0
- Port: 5000
- Debug Mode: Enabled (for development)

2.3 Using the Application

2.3.1 Initial Setup

Registration

1. Navigate to login page
2. Click "Register" or go to /auth/register
3. Fill in username, email, and password
4. Submit registration

Login

1. Go to /auth/login
2. Enter email and password
3. Click "Login"
4. Session will be created

2.3.2 Main Features

1. Overview Dashboard (/overview)

View statistics (courses, tasks, events), See today's schedule, Quick access to recent activities

2. Tasks Management (/tasks)

Create new tasks with title, due date, priority. View all tasks (pending/completed). Update task status. Delete tasks. Pomodoro Focus Timer integrated

3. Course Registration (/course-registration)

Browse available courses, Register for courses, View registered courses, AI-powered schedule optimization

4. Schedule (/schedule)

View class schedule, See weekly calendar, Manage time slots

5. AI Assistant (/ai-assistant)

Ask questions about courses, policies, schedules. View chat history. Browse knowledge base. Select model (Unify Model or Ollama). Get personalized answers based on user data

6. Notes & Summaries (/notes)

Upload PDF, DOCX, or TXT files. AI-powered summarization. View and manage summaries

7. Messages (/messages)

Send messages to other users. View received messages. Mark messages as read

8. Transcript (/transcript)

View academic transcript. See GPA calculation. View course grades

2.3.3 Navigation

- Sidebar navigation for all main features
- Top bar with user profile and logout
- Breadcrumb navigation on some pages

2.4 Configuration

2.4.1 Database Configuration

```
DB_HOST=DESKTOP-V6DPJFP\SQLEXPRESS
DB_NAME=unify
DB_USER=your_username (if needed)
DB_PASSWORD=your_password (if needed)
```

2.4.2 Flask Configuration

```
SECRET_KEY=your-secret-key-here
DEBUG=True (for development)
```

2.4.3 LLM Configuration (Optional)

```
LLM_PROVIDER=ollama
OLLAMA_URL=http://localhost:11434
OLLAMA_MODEL=llama3
```

2.4.4 Multi-Tenant Mode (Optional)

```
MULTI_TENANT_MODE=false
```

Chapter 3

Technical Documentation

3.1 Architecture Overview

Unify follows a layered architecture pattern with clear separation of concerns:

```
    PRESENTATION LAYER
Templates (HTML) + Static Files

            ↓

    CONTROLLER LAYER
Flask Blueprints - HTTP Handling

            ↓

    SERVICE LAYER
Business Logic - RAG Engine, etc.

            ↓

    REPOSITORY LAYER
Data Access - Database Queries

            ↓

    DATA LAYER
SQL Server Database + Models
```

3.1.1 Key Components

- **Application Factory Pattern:** create_app() function
- **Blueprint Pattern:** Modular route organization
- **Repository Pattern:** Data access abstraction

- **Service Layer:** Business logic separation
- **Singleton Pattern:** Database connection management

3.2 MVC Structure

3.2.1 MODELS (src/models/)

Purpose: Data structures and business entities

Examples: User, Student, Task, Course, Enrollment, Calendar, KnowledgeBase, ChatHistory

3.2.2 VIEWS (src/templates/)

Purpose: HTML templates for user interface

Structure: Base template with sidebar navigation, Feature-specific templates (overview.html, tasks.html), AI Assistant interface

3.2.3 CONTROLLERS (src/controllers/)

Purpose: HTTP request handlers, route definitions

Blueprints: auth_controller, user_controller, task_controller, course_controller, ai_assistant_controlle and more

3.2.4 Data Flow

Request → Controller → Service → Repository → Database
Database → Repository → Service → Controller → Response (JSON/HTML)

3.3 API Endpoints

3.3.1 Authentication Endpoints (/auth)

Method	Endpoint	Description
POST	/auth/login	Login user
POST	/auth/register	Register new user
POST	/auth/logout	Logout user
GET	/auth/me	Get current user info

3.3.2 User Endpoints (/users)

Method	Endpoint	Description
GET	/users/	Get all users
GET	/users/¡user_id¿	Get user by ID
POST	/users/create	Create new user

3.3.3 Task Endpoints (/tasks)

Method	Endpoint	Description
GET	/tasks/api	Get all tasks
GET	/tasks/api/¡task_id¿	Get task by ID
GET	/tasks/api/user	Get tasks for current user
POST	/tasks/api	Create new task
PUT	/tasks/api/¡task_id¿	Update task
DELETE	/tasks/api/¡task_id¿	Delete task

3.3.4 AI Assistant Endpoints (/ai-assistant)

Method	Endpoint	Description
GET	/ai-assistant/	AI Assistant main page
POST	/ai-assistant/ask	Ask question to AI
GET	/ai-assistant/history	Get chat history
POST	/ai-assistant/clear-history	Clear chat history
GET	/ai-assistant/categories	Get KB categories
GET	/ai-assistant/knowledge-base	Get KB documents

3.3.5 Other Endpoints

- **Course Endpoints:** /courses/ (GET, POST)
- **Enrollment Endpoints:** /enrollments/ (GET, POST)
- **Schedule Endpoints:** /schedule/ (GET)
- **Calendar Endpoints:** /calendar/api (GET, POST)
- **Message Endpoints:** /messages/api (GET, POST)

- **Transcript Endpoints:** /transcript/api (GET)
- **Overview Endpoints:** /overview/api/stats (GET)

3.4 Database Schema

Database: unify — **System:** SQL Server (also supports MySQL)

3.4.1 Main Tables

Table	Key Fields
User	User_ID, Username, Email, Password_Hash, Created_At
Student	Student_ID, User_ID (FK), Department, Year_Level, GPA
Instructor	Instructor_ID, User_ID (FK), Department, Office, Email
Course	Course_ID, Course_Name, Credits, Instructor_ID (FK), Schedule
Enrollment	Enrollment_ID, Student_ID (FK), Course_ID (FK), Status, Grade
Task	Task_ID, Student_ID (FK), Task_Title, Due_Date, Priority, Status
Schedule	Schedule_ID, Student_ID (FK), Course_ID (FK), Day, Start_Time, End_Time
Calendar	Event_ID, Student_ID (FK), Title, Date, Time, Source
Message	Message_ID, Sender_ID (FK), Receiver_ID (FK), Message_Text, Is_Read
Knowledge_Base	KB_ID, Title, Content, Category, Keywords, Created_Date
Chat_History	Chat_ID, User_ID (FK), Question, Answer, Sources, Created_Date
AI_Note	Note_ID, Student_ID (FK), Original_Text, Summary, File_Name

3.4.2 Indexes

- User: Email, Username
- Student: User_ID
- Enrollment: Student_ID, Course_ID
- Task: Student_ID, Due_Date, Status
- Message: Sender_ID, Receiver_ID

3.5 Design Patterns

3.5.1 1. Application Factory Pattern

Implementation: create_app() function in src/app.py
Purpose: Flexible application creation for testing and deployment
Benefits: Easy configuration, testing support, multiple instances

3.5.2 2. Repository Pattern

Implementation: RepositoryFactory in src/repositories/repository_factory.py
Purpose: Abstract data access layer
Benefits: Easy testing (mocking), database independence, clean separation

3.5.3 3. Singleton Pattern

Implementation: DatabaseConnection in src/core/db_singleton.py
Purpose: Single database connection instance
Benefits: Resource efficiency, connection pooling, state management

3.5.4 4. Service Layer Pattern

Implementation: Services in src/services/
Purpose: Business logic separation from controllers
Benefits: Reusability, testability, maintainability

3.5.5 5. Blueprint Pattern

Implementation: Flask Blueprints in src/controllers/
Purpose: Modular route organization
Benefits: Code organization, scalability, team collaboration

3.5.6 6. MVC Pattern

Implementation: Models, Views (Templates), Controllers
Purpose: Separation of concerns
Benefits: Maintainability, testability, scalability

3.6 Assumptions and Constraints

3.6.1 Assumptions

Database

- SQL Server is available and running
- Database 'unify' exists or will be created
- User has appropriate database permissions
- Trusted connection (Windows Authentication) or credentials provided

Environment

- Python 3.8+ installed
- All dependencies installable via pip
- Port 5000 available for Flask application
- Windows environment (primary, but cross-platform compatible)

User Roles

- Three roles: Student, Instructor, Admin
- Students can register for courses
- Instructors can manage courses
- Admins have full access

AI Features

- Ollama is optional (system works without it)
- AI Note summarization requires transformers library
- Hugging Face models downloadable (internet connection)

File Uploads

- Supported formats: PDF, DOCX, TXT
- File size limits: Reasonable (not explicitly enforced)
- Upload directory: src/uploads/

3.6.2 Constraints

Database

- SQL Server syntax required (not pure MySQL)
- Table names may need brackets: [User] instead of User
- IDENTITY instead of AUTO_INCREMENT
- NVARCHAR instead of TEXT

Performance

- Response time target: <2 seconds (NFR1)
- Concurrent users: Up to 500 (NFR4)
- Database connection pooling via singleton

Security

- Passwords must be hashed (SHA-256)
- No plaintext password storage
- Session-based authentication
- SQL injection prevention via parameterized queries

Scalability

- Single database instance (not distributed)
- File storage on local filesystem
- No load balancing (single server)

Browser Compatibility

- Modern browsers (Chrome, Firefox, Edge, Safari)
- JavaScript enabled required
- CSS3 support required