# Software Requirements Specification

**for**

# <Advisify>

**Version 1.0 approved**

**Prepared by <Ali Raza Bugti, Arfat Ayub, Muhammad Ahmed>**

**<Mohammad Ali Jinnah University, Karachi >**

**<11/12/2025>**

# Table of Contents



# Revision History

| Name | Date | Reason For Changes | Version |
|---|---|---|---|
| | | | |
| | | | |

# 1. Introduction

## 1.1 Purpose

The purpose of **Advisify** is to provide an **AI-based course registration system** that automates the semester registration process for BSCS students. The system is specifically designed to handle **advisory students**, where failed or pending courses, pre-requisites, and semester-wise course allocation require complex manual verification.

## 1.2 Document Conventions

- **Fonts & Styles:** Headings are bold; requirement statements are italic; code/data examples use monospaced font.
- **Requirement Numbering:** Each requirement has a unique ID (e.g., FR-1, NFR-2). Sub-requirements inherit priority unless stated otherwise.
- **Prioritization:** Requirements marked as High, Medium, or Low based on importance.
- **Terminology:** Domain-specific terms and abbreviations (e.g., FYP, AI) are consistently defined in the Glossary.

**Notes & Examples:** Important notes are bolded; examples/data in indented blocks for clarity.

## 1.3 Intended Audience and Reading Suggestions

This document is organized to serve multiple stakeholders:

- **Developers**: Focus on Sections 3 (External Interface Requirements) and 4 (System Features) for implementation details
- **Project Supervisors**: Review Sections 1 (Introduction) and 2 (Overall Description) for project scope and objectives
- **Testers**: Refer to Section 4 (System Features) for functional requirements and Section 5 (Non-functional Requirements) for testing criteria
- **University Administration**: Review Section 2.2 (Product Functions) and Section 5.5 (Business Rules) to understand system capabilities and academic logic

It is recommended to read sections sequentially, starting with the Introduction and Overall Description before proceeding to detailed requirements.

## 1.4 Product Scope

**Advisify** is a full-stack web application that automates the complex process of course registration for Bachelor of Science in Computer Science (BSCS) students. The system addresses critical challenges faced during semester registration periods, particularly for advisory students who require manual intervention from faculty advisors.

**Key objectives:**

- Eliminate physical queuing and manual transcript review processes
- Automatically detect advisory vs. regular student cases based on academic history
- Generate clash-free timetables with optimized section assignments
- Reduce registration time from hours to minutes
- Minimize human error in prerequisite validation and course allocation

**Business Benefits:**

- Reduced administrative workload for faculty advisors and registration department
- Improved student experience during registration periods
- Accurate enforcement of academic rules and prerequisite requirements
- Data-driven insights into student academic progress
- Scalable solution that handles bulk student data efficiently

The system scope is limited to the BSCS program course registration process and does not include student portals, teacher interfaces, or grade management functionalities.

## 1.5 References

- University BSCS Curriculum and Course Outline (2022-2026)
- Academic Regulations and Policies Document
- Student Transcript Format Specifications
- IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications
- FastAPI Documentation: https://fastapi.tiangolo.com/
- Next.js Documentation: https://nextjs.org/docs
- PostgreSQL Documentation: https://www.postgresql.org/docs/

# 2. Overall Description

## 2.1 Product Perspective

**Advisify** is a new, self-contained system designed to replace the existing manual course registration process for advisory students at the university. The system does not replace any existing software but introduces automation where none previously existed.

**System Context:** The system operates within the university's academic ecosystem and interacts with existing data sources (student transcripts in PDF format) and academic policies (BSCS curriculum outline). However, it functions independently without direct integration into existing university management systems.

**Major System Components:**

1. **Frontend Layer** (Next.js): Admin interface for PDF upload and result viewing.
2. **Gateway Backend** (Node.js/Express): File handling, database operations, and API orchestration
3. **AI Agent Pipeline** (FastAPI/Python): Multi-agent system implementing academic logic
4. **Database Layer** (PostgreSQL): Persistent storage for students, courses, sections, and registrations
5. **External Data Sources**: Student transcript PDFs and university course outline.

## 2.2 Product Functions

The system provides the following high-level functions:

**F1. Bulk Student Data Processing**

- Accept single or multiple PDF files containing student transcript data
- Backend (Node.js) automatically converts PDFs to CSV using Node.js PDF parsing library (pdf-parse)
- Conversion happens immediately after upload, stored in uploads/converted/ folder
- Parse and extract structured data from converted CSV - Validate and normalize student transcript data

**F2. Data Validation**

- Automatically Make sure the grades are valid eg [A,A-,B+,B,B-, etc].
- Checks for missing Columns.
- Student ID's should be in format eg [FA22-BSCS-0136, SP21-BSCS-0090]

**F3. Advisory Case Detection (Planned For Fyp2)**

- Automatically identify students requiring faculty advisor intervention
- Detect unresolved failing grades (F/W without subsequent passing attempts)
- Identify students lagging behind ideal curriculum progression

**F4. Dashboard and Reporting**

- Display processing summary with validation.
- Show total students processed.

**Note:** Advisory detection statistics will be added in FYP 2

## 2.3 User Classes and Characteristics

**Primary User: System Administrator**

- **Frequency of Use**: High during registration periods (bi-annually), low otherwise
- **Technical Expertise**: Moderate (familiar with CSV formats, basic database concepts)
- **Responsibilities**:

    - Upload bulk student transcript data via PDF files
    - Trigger PDF-to-CSV conversion
    - Export processed CSV/registration data
    - Run validation agent (Agent 1) for data quality checks
    - View processing summary (placeholder statistics - FYP 1)
    - Export validation results as CSV

  **Priority**: High (most critical user class)

**Note**: Full advisory detection and course allocation will be implemented in FYP 2.

## 2.4 Operating Environment

**Hardware Platform:**

- Server: Linux-based cloud server or on-premise university server
- Minimum: 4GB RAM, 2 CPU cores, 20GB storage
- Client: Modern computer with web browser (admin workstation)

**Software Environment:**

- **Frontend:** Next.js 14+ (React 18+) on Node.js 18+
- **Backend Gateway**: Node.js 18+ with Express.js
- **AI Agent Pipeline:** Python 3.10+ with FastAPI (Agent 1 active in FYP 1 Agents 2-5 planned for FYP 2)
- **Database:** PostgreSQL 14+
- **Web Browser:** Modern browsers (Chrome, Firefox, Safari, Edge)
- **OS:** Cross-platform for clients; Linux for servers

**External Dependencies:**

- Internet for cloud server access
- CSV parsing libraries (Papa Parse for JS, Pandas for Python)

## 2.5 Design and Implementation Constraints

### DI-1. Technology Stack Constraints

- Frontend: Next.js (JavaScript)
- Backend Gateway: Node.js with Express
- AI Agents: Python with FastAPI (Agent 1)
- Database: PostgreSQL (relational structure)

### DI-2. Data Format Constraints

- Input PDFs must contain extractable text (not scanned images).
- Extracted CSV must follow specified format with mandatory columns.
- Student IDs must follow university format (e.g., FA22-BSCS-0136).
- Grades must follow university grading scale (A, A-, B+, …, F, W).

### DI-3. Security Constraints

- Admin authentication required (basic username/password)
- No student personal data exposed via unsecured endpoints
- Database credentials stored in environment variables

### DI-4. Deployment Constraints

- Single-server deployment (web-only interface)

## 2.6 User Documentation

### UD-1. Administrator User Manual

- Step-by-step guide for PDF upload and CSV validation
- Instructions for interpreting advisory case counts and summary reports
- Troubleshooting common CSV errors
- Screenshots of admin UI workflows

### UD-2. System Installation Guide

- Environment setup (Node.js, Python, PostgreSQL)
- Configuration of environment variables
- Database initialization scripts

### UD-3. Technical Documentation

- API endpoint documentation (Node.js and FastAPI routes)
- Database schema diagrams for students and courses

**UD-4. Sample Data Files**

- Example CSV files with correct formatting
- Sample course outline data

## 2.7 Assumptions and Dependencies

**Assumptions**

1. Student transcript PDFs follow consistent structure AND contain readable text
2. University BSCS course outline is stable
3. Course codes and names in transcripts match the course outline
4. Admin users have basic PDF/CSV handling skills
5. System will be used during registration periods by limited users

**Dependencies**

1. Node.js packages (Express, Multer, csv-parser) and Python packages (FastAPI, Pandas, Pydantic) installed correctly.
2. PostgreSQL database is running and accessible.
3. BSCS curriculum data pre-loaded into the database.
4. Input CSV files are correctly formatted.
5. Stable network connection between frontend, backend, and FastAPI.
6. Python virtual environment set up to avoid package conflicts.

# 3. External Interface Requirements

## 3.1 User Interfaces

**UI-1. Admin Dashboard**

- Landing page after login
- Shows quick statistics: total students
- Navigation: "Upload PDF", "Run Automation", "Reports"
- Clean, responsive design (desktop focus), blue-white color scheme

**UI-2. PDF Upload Page**

- Browse and select single or multiple PDF files
- Displays file names and upload progress bar
- Only .pdf files accepted, max 50 MB total size
- Shows PDF-to-CSV conversion status
- Inline validation error messages for parsing failures

**UI-3. Run Automation Page**

- Button to send converted CSV to FastAPI (Agent 1 - Validation only).
- Progress indicator showing validation status.
- Displays success/error messages from validation agent.
- Shows total students processed.

**Note:** Advisory detection (Agent 2) will be added in FYP 2

**UI-4. Results Summary Page**

- Display validation results: total students, valid/invalid records
- Show validation error details (invalid grades, missing fields, format errors)
- Export option: "Download Validation Report (CSV) FYP2"

**Note:** Advisory detection results, course assignments, and timetables will be added in FYP 2

**UI Standards**

- Font: Sans-serif (Inter, Roboto, system default)
- Buttons: Rounded corners, hover states
- Keyboard navigation: Tab/Enter
- Error messages: Inline, clear guidance

## 3.2 Hardware Interfaces

**HI-1. Server Hardware**

- Standard server hardware or cloud VM
- No specialized hardware required
- Standard TCP/IP network communication

## Software Interfaces

**SI-1. Frontend ↔ Node.js Backend**

• REST API over HTTP/HTTPS
• JSON responses, multipart/form-data for PDF uploads
• Endpoints:
- POST /api/upload/pdf upload's PDF files
• JWT authentication (FYP2)

**SI-2. Node.js Backend ↔ FastAPI**

• REST API over HTTP
• JSON request/response

- Endpoints:
  - POST /agents/run-pipeline process Agent 1 (Validation Agent only - FYP 1)
- Response: Validation results + summary statistics
- Standard HTTP status codes for error handling

**Note:** Advisory flags and detection will be added in Agent 2 (FYP 2)

**SI-3. Backend ↔ PostgreSQL Database**

- Node.js using pg driver or Prisma ORM
- Operations: INSERT student/course data, SELECT for summaries
- ACID transactions, connection pooling with PostgreSQL URI

**SI-4. External Libraries**

- Frontend: React 18+, Tailwind CSS.
- Node.js: Express, Multer, bcrypt, csv-parser (CSV processing), (pdf-parse).
- FastAPI: Pandas (data manipulation), Pydantic (data validation).

## 3.3 Communications Interfaces

**CI-1. HTTP/HTTPS Communication**

- Protocol: HTTP/1.1 (development), HTTPS/TLS 1.2+ (production)
- Ports:
  - Frontend: 3000
  - Node.js Backend: 5000
  - FastAPI: 8000
  - PostgreSQL: 5432 (default)
- Message Format: JSON for API data, multipart/form-data for PDF uploads
- Security: HTTPS for production, CORS enabled for frontend-backend

**CI-2. Database Communication**

- Protocol: PostgreSQL Wire Protocol
- Authentication: Username/password via environment variables
- Encryption: SSL/TLS in production

**CI-3. Email Notifications**

- Not implemented in Version 1.0, reserved for future enhancement

**CI-4. Data Transfer**

- Expected PDF size: 1-50 MB (single or multiple files)
- PDF-to-CSV conversion time: ~1-3 minutes for 500 students

- Agent pipeline processing time: ~2-5 minutes for 500 students
- Total end-to-end time: ~5-8 minutes

# 4. System Features

This section details the functional requirements organized by major system features. Each feature includes a description, priority, stimulus/response sequences, and detailed functional requirements.

## 4.1 System Feature: PDF Upload and Data Validation

Description: Accepts student transcript PDFs, automatically converts to CSV, and validates data format and content.

Priority: **High**

### *4.1.1 Stimulus/Response Sequences*

**Stimulus:** Admin uploads one or multiple PDF transcript files via web interface.

**Response:** System stores PDFs, converts them to CSV, validates CSV format, normalizes data, and returns success/error feedback.

**Sequence:**

1. Admin selects PDF file(s) and clicks "Upload PDFs"
2. System validates file extensions (.pdf only) and stores PDFs in uploads/ folder using Multer
3. Node.js backend immediately triggers PDF-to-CSV conversion:
    - Extracts text using pdf-parse library
    - Parses student data with regex patterns
4. System returns upload success message with conversion status to admin
5. Admin navigates to "Run Automation" page and clicks "Run Automation" button
6. Node.js backend:
    - Reads converted CSV file
    - Parses CSV to JSON format
    - Sends JSON data to FastAPI endpoint: POST /agents/run-pipeline
7. FastAPI processes data through Agent 1 (Validation Agent)
8. FastAPI returns validation results to Node.js backend
9. Node.js displays results on dashboard (FYP 1: placeholder statistics shown)

    **Note**: Steps 10+ (Advisory detection with Agent 2, course allocation with Agent 3,timetable generation with Agent 4) will be implemented in FYP 2.

**REQ-PDF-001:** Accept PDF files with .pdf extension only.

**REQ-PDF-002:** Support multiple PDF uploads (batch processing).

**REQ-PDF-003**: Maximum total upload size: 50 MB.

**REQ-PDF-004:** Extract text from PDFs using Node.js PDF parsing library (pdf-parse or pdf-lib)

**REQ-PDF-005:** Parse extracted text using regex patterns and generate structured CSV format with columns: student_id, name, current_semester, total_credits, cgpa, course_code, course_name, credits, grade, semester_taken, original_semester, improved

**REQ-PDF-006:** If PDF parsing fails, log error with filename and reason.

**REQ-PDF-007:** Store converted CSV temporarily for agent processing

**REQ-VAL-001:** Accept CSV files with mandatory columns: student_id, name, current_semester, total_credits, cgpa, course_code, course_name, credits, grade, semester_taken, original_semester, improved

**REQ-VAL-002:** Accept CSV generated from PDF conversion with mandatory columns

**REQ-VAL-003:** Validate student_id format: [A-Z]{2}[0-9]{2}-[A-Z]+-[0-9]{4} (e.g., FA22-BSCS-0136)

**REQ-VAL-004:** Validate grades: A, A-, B+, B, B-, C+, C, C-, D, F, W

**REQ-VAL-005:** Validate credits: positive integers (1–4)

**REQ-VAL-006:** Validate CGPA: 0.00–4.00

**REQ-VAL-007:** current_semester: integer 1–8

**REQ-VAL-008:** Trim whitespace from all string fields

**REQ-VAL-009:** Normalize grade values to uppercase

**REQ-VAL-010:** For multiple attempts of same course, mark previous failing attempts as improved = true. Preserve all rows

**REQ-VAL-011:** Generate validation report: total rows, invalid rows with reasons, number of students, unique courses

**REQ-VAL-012:** If >10% rows invalid, reject entire upload

**REQ-VAL-013:** Output structured JSON grouped by student_id

**REQ-VAL-014:** Log validation errors for admin review

**REQ-VAL-015:** Reject converted CSV if total rows >5000 (data sanity check)

## 4.2 System Feature: Advisory Case Detection (FYP 2)

**Description:** Determines if a student requires faculty advisor intervention.

**Priority:** High

### *4.2.1 Stimulus/Response Sequences*

**Stimulus:** Clean student data passed from validation agent.

**Response:** System classifies students as advisory or regular.

### *4.2.2 Functional Requirements*

**REQ-ADV-001:** Classify student as advisory if any:

- At least one F/W course with improved=false
- Missing core course from previous semesters

**REQ-ADV-002:** Otherwise, classify as regular

**REQ-ADV-003:** For advisory students, list pending issues: course code, course name, reason

**REQ-ADV-004:** Calculate total completed credits, completed core courses, pending core courses

**REQ-ADV-005:** Ignore courses marked improved=true in pending failures

**REQ-ADV-006:** Reference course outline DB for core courses

**REQ-ADV-007:** Output JSON: student_id, is_advisory, pending_courses, completed_credits, completed_core_courses

**REQ-ADV-008:** Log advisory detection statistics for admin dashboard

## 4.3 System Feature: Registration Data Management

**Description:** Persist student records to database.

**Priority:** High

### *4.3.1 Functional Requirements*

**REQ-DB-001:** Insert/update student records (student_id, name, current_semester, total_credits, cgpa)

**REQ-DB-002:** Use transactions to ensure atomic writes

**REQ-DB-003:** Rollback on DB write failure and show error

**REQ-DB-004:** Provide CSV export functionality for results

**Note:** Basic Database is available to store student data.

## 4.4 System Feature: Admin Authentication

**Description:** Ensure only authorized admins can access the system.

**Priority:** Medium

### *4.4.1 Functional Requirements*

**REQ-AUTH-001:** Require username/password authentication

**REQ-AUTH-002:** Hash passwords with bcrypt

**REQ-AUTH-003:** JWT token issued on successful login

**REQ-AUTH-004:** Logout invalidates session/token

## 4.5 Implementation Status Summary

**FYP1 Completed Features**

- PDF upload and storage (REQ-PDF-001 to REQ-PDF-007)
- Automatic PDF-to-CSV conversion
- Data validation agent (REQ-VAL-001 to REQ-VAL-015)
- Basic admin dashboard with placeholder statistics
- Database integration with PostgreSQL (REQ-DB-001, REQ-DB-002)
- CSV export functionality for validation results

**FYP2 Planned Features**

- Advisory case detection
- Course allocation
- Timetable generation
- Admin authentication system
- Real-time dashboard with live statistics

# 5. Other Nonfunctional Requirements

## 5.1 Performance Requirements

**PERF-001:** Dashboard shall load within 2 seconds on standard broadband.

**PERF-002:** Database queries for individual student details shall complete within 500 ms.

**PERF-003:** System shall support up to 5 concurrent admin users without performance degradation.

**PERF-001**-A: PDF-to-CSV conversion shall process up to 10 PDF files (total 500 students) within 3 minutes.

**PERF-001**-B: System shall validate converted CSV within 30 seconds.

**PERF-002**-A: Complete pipeline (PDF upload **then** converstion) shall complete within 6 minutes for 500 students.

## 5.2 Safety Requirements

**SAFE-001:** Database backups shall be created before each major registration cycle.

**SAFE-002:** System shall provide a "Preview Mode" for admins to review allocations before committing. **(FYP2)**

**SAFE-003:** Destructive actions (deletions/overwrites) shall require admin confirmation.

## 5.3 Security Requirements

**SEC-001:** System shall enforce HTTPS (TLS 1.2+) in production.

**SEC-002:** Database credentials and API keys shall be stored in environment variables, not hardcoded.

**SEC-003:** SQL injection prevention through parameterized queries or ORM usage.

**SEC-004:** PDF and form inputs shall be sanitized to prevent XSS attacks.

**SEC-005:** CORS policies shall restrict API access to authorized frontend domains.

**SEC-006:** Sensitive student data shall not be exposed in error messages or logs.

**SEC-007:** Uploaded PDF files shall be deleted from server after successful conversion to CSV (data retention policy).

**SEC-008:** PDF parsing shall sanitize extracted text to prevent code injection attacks

## 5.4 Software Quality Attributes

**QUAL-001** (Maintainability): Code shall follow modular architecture separating frontend, backend, agents, and database.

**QUAL-002** (Testability): Each agent module shall be independently testable with mock inputs.

**QUAL-003** (Usability): Admin interface shall allow CSV upload and retrieval of results within 5 minutes for a new user.

**QUAL-004** (Reliability): System shall achieve 95% uptime during registration periods.

**QUAL-005** (Correctness): System shall enforce prerequisite and credit rules accurately.

**QUAL-006** (Robustness): System shall handle malformed CSVs gracefully without crashing.

## 5.5 Business Rules (to be implemented in FYP2)

**BR-001:** A student is classified as advisory if any failing grades (F/W without subsequent passing attempt) exist or they are behind ideal curriculum.

**BR-002:** Failed courses shall be retaken and have highest priority in allocation.

**BR-003:** Courses shall not be assigned if prerequisites are unsatisfied, except for retakes.

**BR-004:** FYP I (Final Year Project I) shall be assigned only to students with >95 credits.

**BR-005:** Target credit load per student: 15–18 credits, maximum 21.

**BR-006:** If total assigned credits <15 after mandatory courses, add non-prerequisite courses from next semester. (FYP2)

**BR-007:** Courses marked improved = true are ignored in pending failures.

**BR-008:** Course outline (BSCS curriculum) is the single source of truth for prerequisites, default semesters, and core classification.

**BR-009:** Section capacity: 40 students per section by default.

**BR-010:** Timetable slots constrained to weekdays, 9:00 AM– 5:00 PM.

### 5.6 Other Requirements

**OR-001 (Deployment):** System shall be deployable using Docker containers for consistent development and production environments.

**OR-002 (Logging):** System shall maintain logs for major operations (CSV uploads, agent executions, database writes) with timestamps.

**OR-003 (Configuration):** System shall support environment-specific configurations (development, staging, production) via environment variables.

**OR-004 (Backup):** Admin shall perform database backups weekly during non-peak periods.

**OR-005 (Future Enhancement):** Architecture shall allow future addition of student/faculty portals without major refactoring.

**OR-006 (PDF Processing):** System shall support common PDF formats (PDF 1.4-1.7) with readable text content.

**OR-007 (Error Recovery):** If PDF conversion fails for some files, system shall process successfully converted files and report failed ones separately

# Appendix A: Glossary

**PDF Parser:** Node.js module (using pdf-parse library) that extracts text from PDF files and converts to structured CSV format

**Advisory Case:** Student requiring manual faculty review due to failing grades or curriculum deviation.

**Agent:** Modular software component implementing specific business logic (e.g., validation agent, allocation agent).

**BSCS:** Bachelor of Science in Computer Science.

**CGPA:** Cumulative Grade Point Average.

**Core Course:** Mandatory course required for degree completion as per official curriculum.

**Course Outline:** Official BSCS curriculum document defining course sequences and prerequisites.

**CSV:** Comma-Separated Values file format.

**FastAPI:** Python web framework used for AI agent pipeline.

**FYP:** Final Year Project.

**Improved Flag:** Boolean marking that a failed course attempt has been superseded by a later passing attempt.

**Next.js:** React-based frontend web framework.

**Node.js:** JavaScript runtime for backend server.

**PostgreSQL:** Relational database management system.

**Prerequisite:** Course required to be completed before enrolling in dependent courses.

**Regular Case:** Student following standard curriculum progression without unresolved failures.

**Section:** Specific class offering of a course with assigned day, time, and enrolled students.

**Transcript:** Official record listing all courses taken by a student with grades and credits.

# Appendix B: Analysis Models

## Appendix C: To Be Determined List

**TBD-001**: Final decision on AI/LLM integration for advisory reasoning (deterministic rules vs. LLM-assisted logic) - *Decision Pending*

**TBD-002**: Exact time slot durations and break periods for timetable generation - *To be finalized with university admin*

**TBD-003**: Email notification service integration for post-processing reports - *Future enhancement consideration*

**TBD-004**: PDF timetable export format and styling preferences - *Low priority, defer to Phase 2*

**TBD-005**: Multi-program support (extending beyond BSCS to other degree programs) - *Out of scope for FYP, potential future work*