Software Requirements Specification

for

Registration Monitoring Platform

Prepared by Kachen Ruamek, Phubet Siriyasathain,

Suppapich Umchansa, Kamonchanok Phanthusin

Assumption University of Thailand

24 December 2025

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) is to define the functional and non-functional requirements for the Registration Monitoring Platform (Release 1.0).

This document covers the scope of the entire monitoring ecosystem, which consists of three distinct but integrated subsystems:

1. **Web Monitoring Application:** A web-based interface for staff and administrators to visualize, track, and observe overall course schedules.
2. **TQF Master Application:** A desktop software utility designed to extract course data from TQF2 files, generate visual study flowcharts, and export them as PNG or PDF files.
3. **Hardware Verification Unit:** A physical station equipped with a QR scanner and thermal printer to verify registration data and display course details on an OLED screen.

This SRS explicitly *excludes* the functionality of the student course registration transaction itself. The system interacts with existing registration data but does not perform the registration process.

This is the first release of the system, developed as part of the Bachelor of Engineering program at Assumption University.

1.2 Document Conventions

The following conventions are used throughout this document to ensure clarity and consistency:

- **Boldface** is used for significant terms, subsystem names (e.g., **TQF Master**), and hardware components.
- *Italics* are used for file formats (e.g., *TQF2*, *.png*) and document titles.
- **Requirement Priority:**
    - **[High]:** Critical functionality that must be implemented for the system to operate (e.g., TQF2 extraction logic, QR code reading).

  - **[Medium]:** Important functionality that supports core business goals (e.g., PDF export formatting, Web dashboard filters).
  - **[Low]:** Desirable features or cosmetic enhancements (e.g., UI transitions).
- Priorities assigned to high-level requirements are assumed to be inherited by their subordinate detailed requirements unless explicitly stated otherwise.

1.3 Intended Audience and Reading Suggestions

This document is intended for the following stakeholders:

- **Developers:** To understand the extraction logic for **TQF Master**, the data visualization requirements for the Web Application, and the hardware logic, specifically the data flow between the QR scanner and OLED display.
- **Testers/QA:** To design test cases for data accuracy across the three subsystems.
- **Academic Staff/Administrators:** To review the proposed workflow and ensure the monitoring tools meet their operational needs.

Reading Suggestions:

1. **Overview:** All readers should start with Section 1 (Introduction) and Section 2 (Overall Description) to grasp the system architecture.
2. **Developers:** Focus on Section 3 (External Interface Requirement) for specific functional requirements of the platform.
3. **Testers:** Review the Functional Requirements alongside the Non-Functional Requirements (performance and reliability).

1.4 Product Scope

The **Registration Monitoring Platform** is a comprehensive support system designed to enhance the visualization and physical verification of academic course data.

The Web Application automates data collection by extracting real-time course schedules from the **AU Spark** website using a Python Selenium scraper. This data is centralized in a **Supabase** database to ensure the monitoring dashboard reflects the most current university schedule without manual input.

Objectives and Goals:

- **Visualize Complexity:** To transform raw, text-based TQF2 course data into easy-to-understand visual study flowcharts using the **TQF Master** software.
- **Real-time Monitoring:** To provide administrators with a centralized Web Application for observing overall course schedules, ensuring staff have a clear view of academic logistics.
- **Physical Verification:** To bridge the digital registration gap with a Hardware Unit that validates registration data via QR codes, displaying the results on an OLED screen and providing a physical receipt.

Benefits:

- Reduces manual effort and error in creating study plans from TQF2 files.
- Provides a secondary "check-and-balance" for the existing registration system via hardware scanning.
- Allows for easy sharing of study paths via standardized PNG/PDF exports.

1.5 References

1. R. Santos, “ESP32-CAM QR code reader user management system (web server),” Random Nerd Tutorials, May 2025. [Online]. Available: https://randomnerdtutorials.com/esp32-cam-qr-code-reader-web-server/

2. M. Penica, M. Bhattacharya, W. O’Brien, S. McGrath, M. Hayes, and E. O’Connell, “Advancing interoperable IoT-based access control systems: A unified security approach in diverse environments,” IEEE Access, vol. 11, pp. 79450–79464, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10870056

3. R. Onggo, M. S. Nugroho, D. T. Nugroho, and R. R. Hakim, “Revolutionizing classroom attendance: A wireless smart system using ESP-NOW protocol,” International Journal of Interactive Mobile Technologies, vol. 18, no. 10, pp. 23–38, 2024. [Online]. Available: https://www.researchgate.net/publication/387107654

4. H. Elbehiery, “Enhancement of QR-code student’s attendance management system using GPS,” International Journal of Computer Applications, vol. 178, no. 25, pp. 1–7, May 2019. [Online]. Available: https://www.academia.edu/64553334

5. N. Kamaruddin and A. Alduais, “Development of a web-based IoT class attendance monitoring system for a primary school,” Journal of Educational Technology and Engineering,

vol. 6, no. 2, pp. 45–55, 2024. [Online]. Available: https://www.researchgate.net/publication/387031188

6. H. Mustafa and N. Mustafa, "Student records management system (SRMS) using IoT," International Journal of Computational and Experimental Science and Engineering, vol. 11, no. 2, pp. 123–130, 2025. [Online]. Available: https://www.ijcesen.com/index.php/ijcesen/ article/view/1309

7. "Online QR code attendance system," International Research Journal of Modernization in Engineering Technology and Science (IRJMETS), vol. 5, no. 5, pp. 2324–2331, May 2023. [Online].

8. J. H. Chang, "An introduction to using QR codes in scholarly journals," Science Editing, vol. 1, no. 2, pp. 113–117, 2014.

9. A. Bhargava, R. Kumar, and A. P. Bhondekar, "Generation and recognition of covert quick response codes," M.Tech Dissertation, Giani Zail Singh PTU, Bathinda, 2015.

10. A. Mishra and M. Mathuria, "A review on QR code," International Journal of Computer Applications, vol. 164, no. 9, pp. 17–19, 2017.

11. Denso Wave Incorporated, QR Code Essentials. [Online]. Available: https://www.qrcode.com/en/ (accessed: Sept. 10, 2025).

12. DSPMU Ranchi, "IEEE SRS template," [Online]. Available: https://dspmuranchi.ac.in/pdf/Blog/srs template-ieee.pdf. [Accessed: Sept. 17, 2025].

13. Krazytech, "Sample software requirements specification (SRS) report - Airline database," [Online]. Available: https://krazytech.com/projects/ sample-software-requirements-specificationsrs-report-airline-database.

14. TestRigor, "Software requirement specification (SRS) document - Best practices and guide," [Online]. Available: https://testrigor.com/blog/ software-requirement-specification-document/.

15. I. Sommerville, Software Engineering, 10th Global Edition. Pearson, 2016.

16. ConceptDraw, "Flowchart examples – Six circles diagram," [Online]. Available: https://www.conceptdraw.com/examples/6-circles-flow-chartpng. [Accessed: Sept. 17, 2025].

17. S. R. S. Al-Sayed, "Good practice of data modeling and database design for UMIS: Course registration system implementation," Int. J. Adv. Comput. Sci. Appl., vol. 8, no. 1, pp. 34–39, 2017.

18. A. Ullah, A. Khan, and S. Ahmad, "The student enrollment and course tracking system: Meta-project," Int. J. Comput. Sci. Netw. Secur., vol. 18, no. 7, pp. 101–106, 2018.

19 Y. Wang, “Design and implementation of the online course registration system at Tsinghua University,” in 2011 Int. Conf. Comput. Sci. Service Syst. (CSSS), Nanjing, China, 2011, pp. 3760–3763.

2. Overall Description

2.1 Product Perspective

The Registration Monitoring Platform is a specialized, self-contained project designed to augment the institution's academic ecosystem. This product operates as a monitoring layer. It interfaces with the external AU Spark platform via a custom Python Selenium Web Scraper. This scraper serves as the bridge, retrieving course data and populating a Supabase cloud database, which then feeds the Web Application dashboard

This product is not a replacement for the current registration database. Instead, it offers new interfaces for visualizing data that already exists. It interfaces with the current environment in two ways: digital file consumption (processing TQF2 files) and physical verification (scanning existing registration receipts).

The system consists of three main components:

1. **Web Application:** A dashboard for staff oversight.
2. **TQF Master:** A standalone desktop tool for curriculum visualization.
3. **Hardware Unit:** A physical station for student data verification.

2.2 Product Functions

The major functions of the system are divided by subsystem as follows:

2.2.1 Software: Web Application (Monitoring)

- **Automated Data Scraping:** Periodically runs a Python Selenium script to crawl the AU Spark website, extracting course codes, time slots, and room assignments, and updating the Supabase database.
- **Schedule Visualization:** Provides a graphical interface for staff to view the overall course schedule.
- **Real-time Observation:** Allows administrators to monitor the status of courses.
- **Access Control:** Manages login authentication for authorized staff and administrators.

2.2.2 Software: TQF Master (Data Extraction)

- **Data Extraction:** parses standard TQF2 files to identify course names, codes, and structures.
- **Flowchart Generation:** Automatically creates a visual "Study Flowchart" connecting courses based on the extracted data.
- **Export Capability:** Saves the generated flowcharts as high-quality .PNG images or .PDF document.

2.2.3 Hardware: Verification Unit (Scanner & Printer)

- **QR Code Scanning:** Reads registration data from student QR codes.
- **OLED Display:** Visually displays the specific registration data on an OLED screen immediately after scanning.
- **Receipt Printing:** Generates a physical paper receipt of the registration data using a thermal printer.

2.3 User Classes and Characteristics

- **Administrator:**
  - *Role:* Supervises the overall course schedule via the Web Application.
  - *Characteristics:* High familiarity with academic scheduling; requires a comprehensive view of all data.
- **Academic Staff:**
  - *Role:* Uses **TQF Master** to generate course documents and the Web Application to check specific schedules.
  - *Characteristics:* Needs easy-to-use tools to convert complex files (TQF2) into visual aids.
- **General User (Student):**
  - *Role:* Interacts exclusively with the Hardware Unit to verify their registration.
  - *Characteristics:* No technical expertise required; interaction is limited to scanning a QR code and collecting a receipt.

2.4 Operating Environment

- **Web Application:** Accessible via standard web browsers (Chrome, Edge, Firefox).
- **TQF Master:** A desktop application designed for Windows/macOS environments, requiring local file system access to read TQF2 files and save PDFs.
- **Hardware Unit:**
    - **Controller:** Microcontroller-based system (ESP32).
    - **Peripherals:** Optical QR Scanner, OLED Display (I2C/SPI interface), Thermal Printer (TTL/Serial interface).

2.5 Design and Implementation Constraints

- **Source Dependency:** The data extraction relies entirely on the HTML DOM structure of the AU Spark website. Changes to the AU Spark interface may require immediate updates to the Selenium script.
- **Database Integration:** The system is constrained to use Supabase as the primary backend-as-a-service for data persistence and real-time updates.
- **No Registration Logic:** The system is strictly constrained to *monitoring* and *displaying*. It must not perform add/drop functions or modify the central database.
- **TQF2 File Structure:** The **TQF Master** software relies on a specific internal structure of the TQF2 files. If the university changes this file format, the extraction logic may fail.
- **Hardware Memory:** The microcontroller has limited memory, restricting the amount of text that can be buffered for printing or displaying on the OLED at one time.

2.6 Assumptions and Dependencies

- **AU Spark Availability:** It is assumed that the AU Spark website is operational and accessible during the monitoring periods and that it does not implement anti-scraping measures that would block the Selenium scrip
- **Existing Data Availability:** It is assumed the external Registration System will generate valid QR codes and TQF2 files are available.

- **Hardware Availability:** The project assumes specific hardware components (OLED, Thermal Printer) remain available for purchase and compatible with the chosen microcontroller libraries.
- **QR Code Standard:** It is assumed the QR codes contain text data in a standard encoding (UTF-8/ASCII) that the scanner can interpret.

3. External Interface Requirements

3.1 User Interfaces

3.1.1 Web Application (Monitoring)

The web interface is built using **Next.js** and **React**. It is designed for clear visualization of complex schedules involving 100+ courses.

- **Schedule View:**
  - **Aggregated Blocks:** The main view displays time blocks (e.g., *Mon 09:00 - 12:00*).
  - **Interaction:** Clicking a Time Block triggers a modal/pop-up or dropdown listing all courses active during that specific period.
  - **Detail View:** Clicking a specific course name within the list opens a detailed view showing the Room Number, Lecturer, and Section.
- **Standard Elements:**
  - A navigation bar must be present on all pages (Home, Schedule, Admin Login).
  - The interface must be responsive (usable on Desktop and Tablet screens).

3.1.2 TQF Master (Desktop Application)

- **Interface Type:** Web-based User Interface (integrated or separate from the Monitoring Dashboard).
- **File Input:** A "Drag and Drop" zone accepting .DOCX files (Word Documents).
- **Extraction Preview:** A text-area panel that displays the raw data extracted (Course Code, Pre-requisites) immediately after file loading for user verification.
- **Flowchart Canvas:** A visual pane showing the generated study path (nodes and arrows).
- **Export:** A control panel with a "Download PDF" button that captures the current viewport state.

3.1.3 Hardware Display (OLED)

- **Idle Screen:** Displays a prompt such as "Ready to Scan".

- **Result Screen:** Upon a successful QR scan, the 0.96" (or similar) OLED screen displays text in a specific format:
    - Line 1: "Registration Verified"
    - Line 2: [Course Code 1]

3.2 Hardware Interfaces

3.2.1 ESP32 Microcontroller (The Controller)

The ESP32 acts as the central processing unit. It interfaces with two main peripherals:

- **QR Code Scanner:**
    - **Interface Type:** UART (Serial Communication). *Assumption: Standard UART TTL*.
    - **Connection:** The scanner's TX pin connects to the ESP32's RX pin.
    - **Function:** Sends the decoded string (ASCII text) from the QR code to the ESP32 buffer.
- **OLED Display:**
    - **Interface Type:** I2C (Inter-Integrated Circuit).
    - **Connection:** Uses SDA (Data) and SCL (Clock) pins on the ESP32.
    - **Function:** Receives display commands and character bitmaps from the ESP32.

3.2.2 QR Scanner Device

- **Input:** Optical scanning of physical paper receipts.
- **Data Format:** The scanner must be configured to output data ending with a carriage return \n or similar delimiter so the ESP32 knows when the message ends.

3.3 Software Interfaces

3.3.1 AU Spark Website (Data Source)

- **Interface:** HTML DOM (Document Object Model).
- **Mechanism:** A **Python Selenium** script runs an automated browser instance.

- **Interaction:** The script navigates to the AU Spark URL, locates specific HTML tags (e.g., or <table> containing schedule data), scrapes the inner text, and cleans the data.
- **Constraint:** This interface is unidirectional (Read-Only).

3.3.2 Supabase (Database & Backend)

- **Interface:** RESTful API or Supabase Client Library (supabase-js / supabase-py).
- **Inbound Data:** The Python Selenium script sends POST/UPSERT requests to populate the database with scraped course data.
- **Outbound Data:** The Next.js Web Application sends GET requests to retrieve schedule data for the "Block" view.

3.3.3 TQF Master Libraries

- **Backend Framework:** Python FastAPI serves as the API endpoint for file processing.
- **Parsing Engine:** Uses python-docx combined with custom Regex algorithms to scrape text from uploaded Word documents.
- **Frontend** Visualization: Uses React Flow library for rendering the node-based study plan.
- **Export Engine:** Uses jsPDF and html-to-image (or React Flow viewport capture) to generate the final PDF file.

3.4 Communications Interfaces

3.4.1 Web Protocols

- **HTTP/HTTPS:** Used for all communication between the Web Client (Next.js) and the Supabase Cloud. All data in transit must be encrypted via SSL (HTTPS).
- **WebScraping Protocol:** The Selenium script uses the **WebDriver** protocol to control the browser.

3.4.2 Hardware Protocols (Serial/UART)

- **Baud Rate:** The communication speed between the QR Scanner and the ESP32 (e.g., 9600 bps or 115200 bps). Both devices must be set to the same rate.
- **Data Bits:** Standard 8N1 (8 data bits, No parity, 1 stop bit).

4. System Features

4.1 Automated Course Data Scraping (Backend Logic)

4.1.1 Description and Priority

This feature automates the extraction of course schedule data from the external AU Spark website using a Python Selenium script and updates the Supabase cloud database.

- **Priority: [High]** (Critical: Without this data, the web application is empty).

4.1.2 Stimulus/Response Sequences

- **Stimulus:** The Administrator triggers the "Update Data" function via the Admin Dashboard, or the system reaches a scheduled cron-job time (e.g., daily at 00:00).
- **System Response:**
    1. The Python script launches a headless browser instance.
    2. The script logs into AU Spark.
    3. The script iterates through the course list pages.
    4. The system parses the HTML to extract Course Code, Name, Section, Room, and Time.
    5. The system cleans the data (removes HTML tags).
    6. The system performs an UPSERT (Update or Insert) operation to the Supabase database.
    7. The system returns a "Success" or "Error" log to the administrator.

4.1.3 Functional Requirements

- **REQ-SCRAPE-01:** The system shall use the Selenium WebDriver library to navigate the AU Spark website.
- **REQ-SCRAPE-02:** The system shall extract the following data fields for each course: Course Code, Course Name, Section Number, Lecturer Name, Room Number, Day, and Time Slots.
- **REQ-SCRAPE-03:** The system shall store the extracted data in the Supabase database, overwriting existing records if the Course Code and Section match (ensuring data freshness).

- **REQ-SCRAPE-04:** The system shall log any extraction failures (e.g., network timeout, HTML structure change) and alert the administrator.

[Figure 4.1: Data Flow Diagram illustrating the flow from AU Spark -> Selenium Script -> Supabase DB]

4.2 Interactive Schedule Visualization (Web Application)

4.2.1 Description and Priority

This feature defines the "Block" view user interface on the Next.js website, allowing staff to visualize the schedule of over 100 courses without a cluttered table.

- **Priority: [High]**

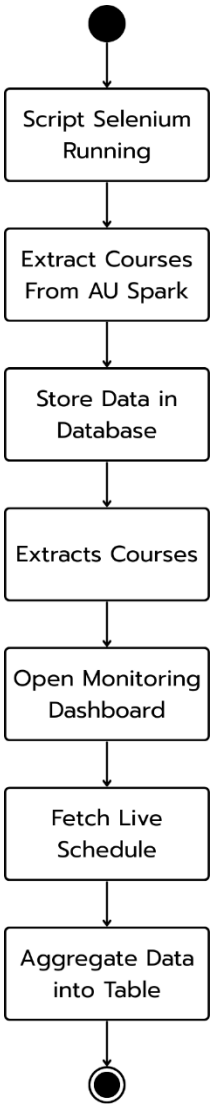
4.2.2 Stimulus/Response Sequences

- **Stimulus:** A user (Staff) clicks on a "Time Block" (e.g., "Mon 09:00 - 12:00") on the main schedule grid.
- **System Response:**
    1. The system queries Supabase for all active courses during that specific day and time range.
    2. The system displays a Modal (Pop-up) window listing the courses.
    3. User clicks on a specific course in the list.
    4. The system expands the view to show details (Room, Lecturer).

4.2.3 Functional Requirements

- **REQ-WEB-01:** The Web Application shall display a high-level weekly grid aggregated by time blocks (Morning, Afternoon) rather than individual rows for every course.
- **REQ-WEB-02:** The system shall provide a "Click-to-Expand" interaction model; clicking a time block shall trigger a modal window.
- **REQ-WEB-03:** The modal window shall display a scrollable list of all courses occurring in that selected time block.

- **REQ-WEB-04:** The system shall allow filtering of the schedule blocks by "Faculty" or "Room" (if applicable).
- **REQ-WEB-05:** The User Interface shall be responsive, adjusting the grid layout for Desktop and Tablet resolutions.



[Figure 4.2: Activity Diagram illustrating the flow from extraction data process to display data on the dashboard]

4.3 TQF2 Data Extraction & Flowcharting (TQF Master)

4.3.1 Description and Priority

A web-based feature that parses TQF2 **Word (DOCX)** documents to generate interactive study flowcharts using the React Flow library

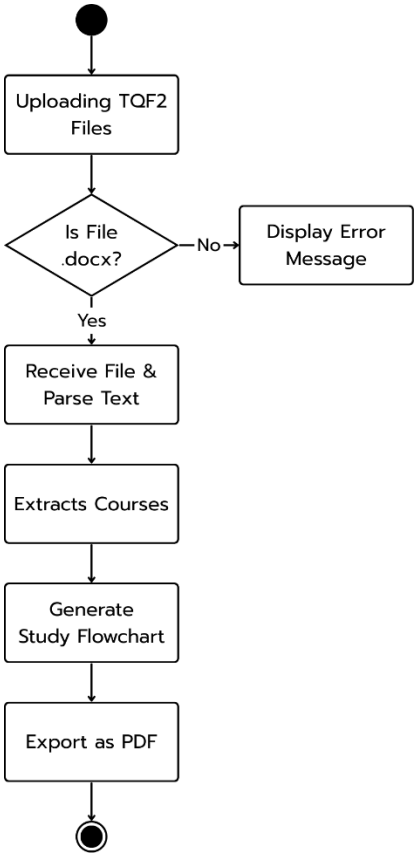
Priority: [Medium] (Core feature for the desktop app).

4.3.2 Stimulus/Response Sequences

- **Stimulus:** User uploads a TQF2.docx file via the interface.
- **System Response:**
    1. The Frontend sends the file to the **FastAPI** backend.
    2. The Backend parses the DOCX file using python-docx and Regex to identify course codes and prerequisites.
    3. The Backend returns a JSON object defining nodes (courses) and edges (connections).
    4. The Frontend renders the graph using **React Flow**, applying "Manhattan edge routing" for clean lines.
    5. User clicks "Export PDF".
    6. The system captures the canvas and downloads a PDF.

4.3.3 Functional Requirements

- **REQ-TQF-01:** The system shall accept .DOCX files as input.
- **REQ-TQF-02:** The system shall use Regex algorithms to extract course codes and prerequisite relationships from unstructured Word text.
- **REQ-TQF-03:** The User Interface shall render the study plan using React Flow, utilizing a grid-based positioning algorithm to arrange courses by semester/year.
- **REQ-TQF-04:** The system shall use Manhattan edge routing to draw orthogonal lines between prerequisite courses to minimize visual clutter.
- **REQ-TQF-05:** The system shall export the visual representation to PDF using client-side libraries (jsPDF).



[Figure 4.3: Activity diagram illustrating the flow from input file process to export output as PDF file]

4.4 Hardware Verification & Display

4.4.1 Description and Priority

This feature covers the physical verification process using the ESP32-based hardware unit to scan receipts and display data.

- **Priority: [Medium]** (Verification layer).

4.4.2 Stimulus/Response Sequences

- **Stimulus:** A staff member scans a student's registration receipt QR code using the attached optical scanner.
- **System Response:**
    1. The Scanner sends the decoded string to the ESP32 microcontroller.
    2. The ESP32 parses the string.
    3. The ESP32 clears the OLED screen.

4. The ESP32 formats the data and draws the text on the OLED display.
5. (Optional) The system triggers a beep sound to indicate a successful scan.

4.4.3 Functional Requirements

- **REQ-HW-01:** The hardware unit shall be capable of reading standard QR codes containing ASCII text data via the UART interface.
- **REQ-HW-02:** The system shall parse the input data to separate distinct course codes (e.g., splitting by commas or newlines).
- **REQ-HW-03:** The OLED screen shall display "Verification Successful" followed by the list of registered courses.
- **REQ-HW-04:** The system shall handle "Invalid Data" or "Read Error" by displaying an error message on the OLED screen.
- **REQ-HW-05:** The response time from scanning to displaying text on the OLED shall be less than 2 seconds.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- **Hardware Response Time:** The interval between scanning a QR code and displaying the result on the OLED screen must not exceed 2.0 seconds. This ensures a smooth flow of students during high-traffic registration periods.
- **Web Dashboard Latency:** The Web Application (Next.js) shall render the schedule "Block" view in under 3.0 seconds over a standard 4G/WiFi connection.
- **Scraping Efficiency:** The Python Selenium extraction script must complete the full crawl of the AU Spark course list (approx. 100+ courses) within 15 minutes.
- **Scraper Politeness:** To prevent overloading the AU Spark server or triggering anti-DDoS protections, the scraper must implement a minimum delay (e.g., 2 seconds) between page requests.

5.2 Safety Requirements

- **Hardware Enclosure:** The ESP32 controller and wiring must be enclosed in a non-conductive case (e.g., 3D printed PLA box) to prevent short circuits and protect the user from exposed electrical pins.
- **Thermal Safety:** If the thermal printer is utilized heavily, the system must monitor for overheating. The software should pause printing if the print head temperature exceeds manufacturer safety limits.
- **Data Integrity Safety:** The system must not accidentally delete existing records in the Supabase database during a "sync" unless the new data is verified to be valid.

5.3 Security Requirements

- **Database Access Control:** Access to the Supabase database shall be restricted using Row Level Security (RLS) policies. Only authenticated Administrators can write/modify data; the public web view is Read-Only.
- **Credential Management:** The login credentials for the AU Spark website (used by the Selenium script) must not be hardcoded in the source code. They must be stored in secure Environment Variables on the server.

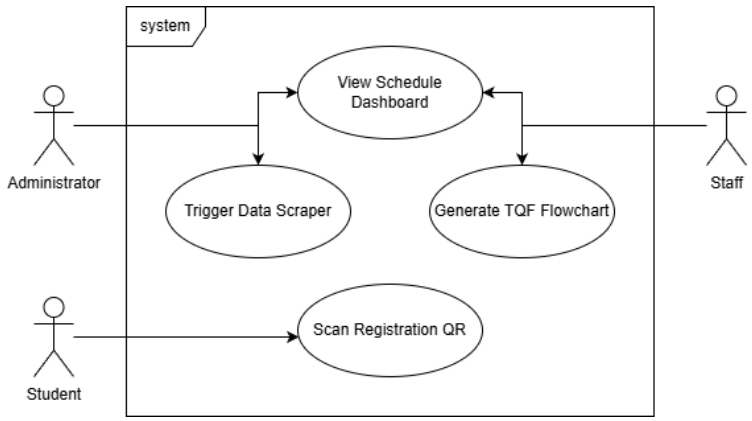
- **Admin Authentication:** The Web Application dashboard for system management must require a secure login (Email/Password) to prevent unauthorized access to system logs or manual overrides.

5.4 Software Quality Attributes

- **Availability:** The Web Application should be available 99.9% of the time, barring outages of the underlying Supabase infrastructure.
- **Robustness (Scraper):** The TQF Master and Scraper must be able to handle "bad data" (e.g., a TQF2 file with missing pages or an AU Spark page that fails to load) without crashing. It should skip the error and log it.
- **Portability (TQF Master):** The desktop application should be packaged as an executable (.exe) so that staff members do not need to install Python or libraries manually.

5.5 Business Rules

- **BR-01:** Only the "Administrator" role is authorized to manually trigger the Scraper or clear the database.
- **BR-02:** Staff members may view the schedule and generate flowcharts but cannot modify the core schedule data scraped from AU Spark.
- **BR-03:** The hardware verification unit is strictly for "Read" operations; it cannot modify the student's registration status in the database.



[Figure 4.4: Use Case Diagram showing the separation of Admin, Staff, and Student privileges]

6. Other Requirements

6.1 Database Requirements

- **Platform:** The system must use Supabase (PostgreSQL-based) as the backend-as-a-service.
- **Schema:** The database must maintain a relational structure linking Courses, Sections, Instructors, and TimeSlots.
- **Backup:** The database should be backed up automatically (daily) to prevent data loss in case of corruption during a scraping update.

6.2 Legal and Ethical Requirements (Scraping)

- **Educational Use Only:** The data extracted from AU Spark is for internal monitoring and educational project purposes only. It must not be resold or used for commercial gain.
- **Attribution:** The Web Application must clearly state that the schedule data is sourced from AU Spark and is not the official registration platform.

6.3 TQF2 File Format Requirements

- **Language:** The **TQF Master** application is required to support **English** language TQF2 files only. Thai language files are out of scope for Release 1.0.
- **File Type:** Input files must be strictly in **.PDF** format. Word documents or images are not supported.

6.4 Implementation Constraints

- **Hardware Cost:** The total cost of the Hardware Verification Unit (ESP32 + Scanner + OLED + Printer) should be minimized to demonstrate the feasibility of a low-cost monitoring solution.

Appendix A: Glossary

A.1 Acronyms and Abbreviations

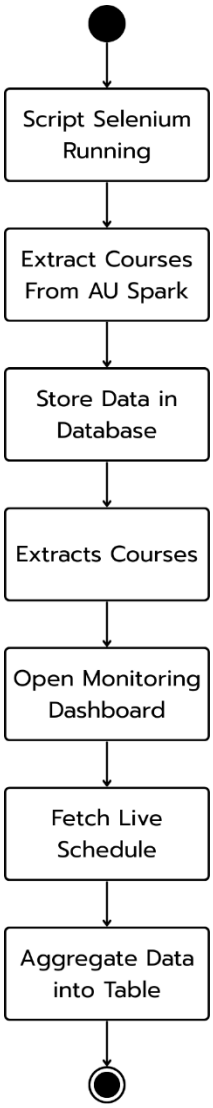
Acronym	Term	Definition
API	Application Programming Interface	A set of rules that allows different software entities to communicate with each other.
DOM	Document Object Model	The data representation of the objects that comprise the structure and content of a document on the web (used here for scraping AU Spark).
HTTP/S	Hypertext Transfer Protocol / Secure	The protocol used for transmitting web data between the client and the server.
I2C	Inter-Integrated Circuit	A synchronous, multi-master, multi-slave, packet-switched, single-ended, serial communication bus used to connect the ESP32 to the OLED display.
OLED	Organic Light-Emitting Diode	The display technology used in the hardware unit to show verification results.
PDF	Portable Document Format	The file format used for the TQF2 curriculum documents.
PNG	Portable Network Graphics	A raster-graphics file format that supports lossless data compression, used for exporting flowcharts.
QR Code	Quick Response Code	A type of matrix barcode (or two-dimensional barcode) used to store the student's registration data on the receipt.
SRS	Software Requirements Specification	The document (this document) that describes what the software will do and how it will be expected to perform.
TQF	Thai Qualifications Framework	A standardized system of higher education qualifications in Thailand.
TTL	Transistor-Transistor Logic	A class of digital circuits; in this context, refers to the serial communication level for the Thermal Printer.
UART	Universal Asynchronous Receiver-Transmitter	The hardware communication protocol used between the QR Scanner and the ESP32 Controller.

A.2 Definitions

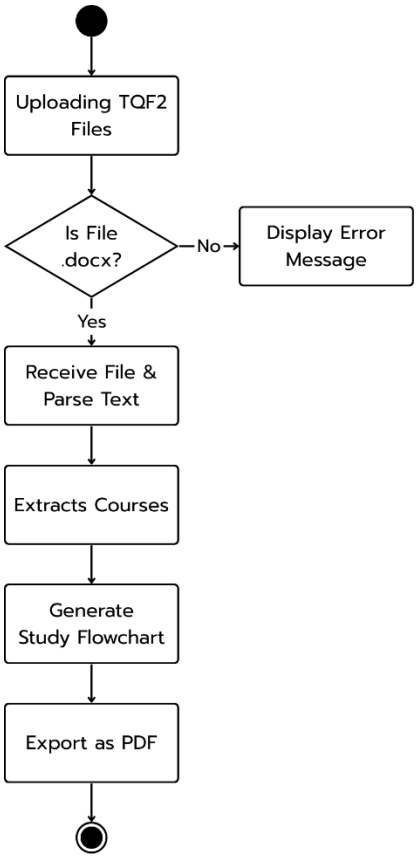
- **AU Spark**
    - The external university website that serves as the source of truth for course schedules. The Monitoring Platform scrapes data from this site.
- **ESP32**
    - A low-cost, low-power system on a chip (SoC) microcontrollers with integrated Wi-Fi and dual-mode Bluetooth. It serves as the "brain" of the Hardware Verification Unit.
- **Next.js**
    - The React framework used to build the frontend of the Web Application.
- **Scraper (or Web Scraper)**
    - The automated Python script responsible for navigating the AU Spark website, extracting course data, and saving it to the database.
- **Supabase**
    - An open-source Backend-as-a-Service (BaaS) platform used in this project to host the PostgreSQL database and manage real-time subscriptions.
- **TQF Master**
    - The specific name of the desktop software component developed in this project that processes TQF2 files and generates study flowcharts.
- **TQF2 File**
    - A document detailing the curriculum mapping and course descriptions. For this project, it specifically refers to the *Program Specification* document in PDF format.
- **UPSERT**
    - A database operation that acts as a combination of UPDATE and INSERT. If a record exists, it is updated; if not, a new record is created. This ensures the schedule data is always fresh without creating duplicates.

Appendix B: Analysis Model

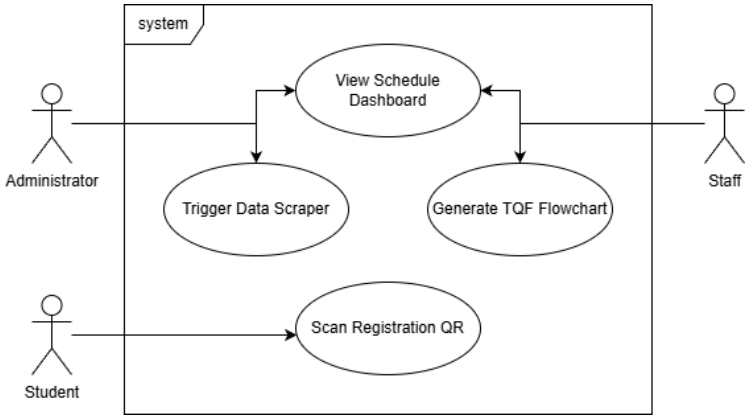
[Figure 4.1: Data Flow Diagram illustrating the flow from AU Spark -> Selenium Script -> Supabase DB]



[Figure 4.2: Activity Diagram illustrating the flow from extraction data process to display data on the dashboard]



[Figure 4.3: Activity diagram illustrating the flow from input file process to export output as PDF file]



[Figure 4.4: Use Case Diagram showing the separation of Admin, Staff, and Student privileges]