

📝 Software Requirements Specification (SRS)

USP Enrollment System

Version: 1.3

**Platform**:

* Web App (Phases 1 & 2)
* Mobile Web App (Phase 2)
* Web App with Microservices using Python Flask + SQLite (Phase 3)

**1. Introduction**

**1.1 Purpose**

The USP Enrollment System is designed to automate and manage the enrollment process at the University of the South Pacific. This system enables students to view and register for courses, track prerequisites, view fees and holds, and apply for academic services like grade rechecks and graduation.

**1.2 Scope**

The system will be delivered incrementally:

* **Phase 1**: Executable prototype with agile documentation.
* **Phase 2**: Mobile-compatible app with internal fees and holds, MVC architecture refactor.
* **Phase 3**: Microservices integration, new features (grade recheck, graduation, transcript), and implementation using Python, Flask, and SQLite.

**1.3 Audience**

* SAS Managers
* Students
* Academic Staff
* Developers & Testers

**2. System Overview**

**2.1 Actors**

* **Student**: Views courses, enrolls, views fees/holds, requests services.
* **SAS Manager**: Manages course registration window and student restrictions.
* **Admin**: Defines rules for holds and student access.

**3. Functional Requirements (FRs)**

**🔹 Phase 1 – Initial Functional Requirements**

**FR1: Student Profile Management**

* FR1.1: Add/edit/view personal details (ID, name, DOB, email, phone).

**FR2: Course Management**

* FR2.1: Display course catalogue and details.
* FR2.2: Graphically show prerequisites.
* FR2.3: Highlight unmet prerequisites.

**FR3: Enrollment Management**

* FR3.1: Allow students to enroll in up to 4 courses.
* FR3.2: Validate prerequisites before enrollment.
* FR3.3: View current enrollment.

**FR4: Fees and Holds (External)**

* FR4.1: Fetch fee invoice from external system.
* FR4.2: Display current academic holds.

**FR5: Registration Control**

* FR5.1: SAS Manager can open/close registration.

**🔹 Phase 2 – Incremental Functional Requirements**

**FR6: Internal Fees & Holds**

* FR6.1: Maintain fee and hold data within the internal database.
* FR6.2: Update student hold status and restrict functionality accordingly.

**FR7: Improved Prerequisite Management**

* FR7.1: Validate and store accurate prerequisite relationships.

**FR8: Mobile Interface (Mobile Web App)**

* FR8.1: Present a responsive UI for mobile browsers.
* FR8.2: Keep business logic and database layer intact (MVC separation).

**FR9: Version Control & Agile Documentation**

* FR9.1: Maintain a Git-based version control.
* FR9.2: Document user stories, standups, and tasks in an agile project management tool (e.g., GitHub Projects or Azure DevOps).

**🔹 Phase 3 – Final Increment (Flask + SQLite Implementation)**

**FR10: Grade Recheck**

* FR10.1: Allow students to apply for grade recheck.
* FR10.2: Notify student on grade changes.

**FR11: Graduation & Aegrotat/Compassionate Pass**

* FR11.1: Apply for graduation or special passes.
* FR11.2: Auto-notify student via email if eligible.

**FR12: Hold Rules Configuration**

* FR12.1: Admin defines access rules based on hold status.
* FR12.2: Restrict or allow features like course registration, grade view, transcript generation, etc.

**FR13: Transcript Generation**

* FR13.1: Generate and download academic transcript as a PDF.

**FR14: Microservices**

* FR14.1: Develop a standalone microservice for grade recheck or transcript.
* FR14.2: Use GET and POST endpoints.
* FR14.3: Secure access via token or basic auth.

**FR15: Aspect-Oriented Programming (AOP)**

* FR15.1: Log specific events such as Login/Registration.
* FR15.2: Use decorators or middleware in Flask for cross-cutting concerns.

**FR16: Testing**

* FR16.1: Unit test with unittest or pytest.
* FR16.2: Perform integration testing of endpoints.
* FR16.3: Ensure 90%+ code coverage.

**4. Non-Functional Requirements (NFRs)**

**🔹 Performance**

* NFR1: Web responses must load within 2 seconds under normal load.
* NFR2: Microservices must respond to API requests within 1 second.

**🔹 Usability**

* NFR3: System should be mobile responsive and easy to navigate.
* NFR4: Must provide user feedback on errors or invalid actions.

**🔹 Security**

* NFR5: Only authorized users (admin/student) can access role-specific actions.
* NFR6: Use token-based authentication for microservices.
* NFR7: Input validations must be in place to prevent SQL injection and XSS.

**🔹 Maintainability**

* NFR8: Follow three-tier architecture (MVC) across all components.
* NFR9: Code must be modular and version-controlled using Git.

**🔹 Portability**

* NFR10: Flask + SQLite system must run on localhost and Linux hosting environments.

**5. System Architecture**

**Technologies Used:**

|  |  |
| --- | --- |
| Layer | Tool / Language |
| UI | HTML, CSS, JS (Bootstrap), Jinja2 |
| Backend | Python (Flask) |
| Database | SQLite |
| Microservices | Flask-based services (localhost) |
| Version Control | Git + GitHub |
| Testing | unittest, pytest |
| Deployment | Localhost (Flask) or Heroku for demo |

**6. Assumptions and Constraints**

* The existing external systems (HR/forms, grades, initial fees) are simulated or mocked.
* Only SAS Manager can manage the open/close registration window.
* Students must use USP email addresses.

**7. Appendices**

**A. Diagrams to include:**

* ER Diagram
* Microservices Communication Flow

**B. API Documentation**

To be generated via:

* Flask docstrings
* Swagger/OpenAPI YAML