Software Requirements Specification (SRS)

For: Student Registration System

Prepared by: Anmol Shukla

Date: 31/07/2025

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this SRS is to define the requirements and functional specifications of the **Student Registration System** (SRS) Web Application. This document provides the foundation for design, development, testing, and maintenance by defining what the system must do and the constraints under which it must operate.

1.2 Document Conventions

- **Shall** denotes a mandatory requirement.
- **Should** denotes a recommended requirement.
- Numbered formatting is used for traceability.
- UML diagrams are used to represent models.

1.3 Intended Audience and Reading Suggestions

- **Developers:** For system implementation.
- **Testers:** For verifying compliance with requirements.
- **Project Managers:** For tracking progress and scope.
- **Stakeholders/Clients:** For validation and acceptance.
- **Future Maintainers:** For upgrades and debugging.

1.4 Product Scope

The Student Registration System aims to streamline academic data management processes by enabling easy, efficient, and secure recording, viewing, modification, and deletion of student data. The focus is on simplicity and usability for educational institutions seeking to digitize their registration workflow.

1.5 References

- IEEE 830-1998 SRS Standard
- HTML5, CSS3, JavaScript documentation
- [Insert links to UI wireframe tools]

2. Overall Description

2.1 Product Perspective

- Standalone web application (can be extended with backend).
- Replaces manual/Excel-based registration workflow.
- Modular, extendable architecture for future compatibility.

2.1.1 System Context Diagram (Sample Description)

```
+---------+         +---------------------------+         +----------+
|  Admin  |<------->| Student Registration System|<------->|  Browser |
+---------+         +---------------------------+         +----------+
                               |
                          +------------------+
                          | Persistent Store |
                          +------------------+
```

- **Admin** interacts via browser UI.
- Data stored locally (with possible extensions to server database).

2.2 System Interfaces

User Interface: Web UI (form, tables, controls)
Data Interface: Local Storage (JSON objects); extensible to API/database.

2.3 Product Functions

- Student registration (add new)
- Validation of all inputs
- Persistent list display (table of students)
- Edit/update and delete functionality
- Confirmations and warnings
- Support for search/filter (future)

2.4 User Characteristics

User Class	Skills Needed	Access Scope
Admin	Basic computer, English	Full (CRUD)
Student	N/A (future scope)	View/update own record

2.5 Operating Environment

Component	Specification
Browser	Chrome, Firefox, Safari, Edge (>2020)
Devices	PC/laptop, smartphone, tablet
OS	Windows, Linux, macOS, Android, iOS
RAM/CPU	2GB+/1GHz+

2.6 Constraints

- Use strictly HTML5, CSS3, and JavaScript.
- Responsive and accessible by WAI guidelines.
- Must work on modern browsers.
- User privacy: No sensitive data stored.
- Performance: <2 second latency for CRUD.

2.7 Assumptions & Dependencies

- Internet connection for initial resource load.
- User permissions set via browser/device.

2.8 Future Scope

- Role-based access (e.g., Teachers, Students)
- Export functionality (CSV/PDF)
- Integration with institutional database/server
- Email notifications

3. System Features and Requirements

3.1 Functional Requirements (Detailed)

No.	Description	Priority
FR-1	System shall render a registration form with fields: Name, Student ID, Email, Contact Number.	High
FR-2	Name must accept only alphabets (60 char max), mandatory field.	High
FR-3	Student ID shall be unique (alphanumeric, 15 char max).	High
FR-4	Email field must validate a proper email format.	High
FR-5	Contact Number must be 10-digit numeric only (with optional '+91' prefix for India).	High
FR-6	On successful validation, new record shall append to the students table and persist.	High
FR-7	System shall display validation errors inline next to fields.	Medium
FR-8	System shall list all registered students in a responsive, sortable table with action buttons per record.	High
FR-9	System shall enable editing of student record via a modal or inline form.	High
FR-10	System shall prompt for confirmation before delete and remove the record if confirmed.	High
FR-11	System shall persist data in browser localStorage; fallback to sessionStorage if not available.	Medium
FR-12	System should allow searching/filtering by name/ID (Optional, for future).	Low
FR-13	Form fields shall be clearable/reset with a button.	Medium
FR-14	The system shall provide user feedback on save/delete (success/failure notification).	Medium

FR-15	All UI actions shall be accessible via keyboard navigation (tab/focus).	Medium
FR-16	System shall log all CRUD actions (future, minimal/audit purpose).	Low

3.2 Non-Functional Requirements

3.2.1 Performance

- Handling at least 200 student entries with <1.5 second max UI response.
- Page load with all assets under 3 seconds (broadband, desktop).

3.2.2 Reliability & Availability

- Data loss prevented via confirmation dialogs.
- Data persists after reload/refresh through localStorage.

3.2.3 Security

- Input sanitized against XSS/HTML injection.
- No passwords/sensitive info.

3.2.4 Usability

- Clean layout, clear labeling.
- Inline error messages.
- Mobile and desktop usability.

3.2.5 Maintainability

- Source code uses modular JS/ES6 Classes.
- Styling with separate CSS.

3.2.6 Portability

- OS/browser independent, platform-agnostic.

3.3 External Interface Requirements

3.3.1 User Interfaces

- **User Story:** "As an admin, I want to easily add and review students so I don't have to manage paper records."
- **Form:**
  - Text fields (Name, ID, Email, Contact).
  - Buttons: Submit, Reset.
  - Table with data, action icons (edit, delete).
- **Responsive:** Auto-fit grid, stack for mobiles.

3.3.2 Hardware Interfaces

- Standard device input: mouse, keyboard, touchscreen.

3.3.3 Software Interfaces

- localStorage API for data (key: 'students')
- JavaScript (ES6+), minimal 3rd party libraries

3.3.4 Communications Interfaces

- Not required in initial version; can use RESTful API in future.

3.4 System Attributes

- **Flexibility**
- **Accessibility:** ARIA labels, color contrast
- **Stability:** Graceful degradation if storage unavailable

4. System Models and Design

4.1 Use Case Diagram (Textual)

```
+----------+           +-----------------+
|  Admin   |---------> | Register Student|
|          |---------> | View Students   |
|          |---------> | Edit Student    |
|          |---------> | Delete Student  |
+----------+           +-----------------+
```

Actors: Admin (future: Student)

4.2 Use Case Descriptions

Use Case	Steps
Register Student	1. Fill form → 2. Validate → 3. Add to table → 4. Persist
Edit Student	1. Select edit → 2. Show form → 3. Validate & update → 4. Persist
Delete Student	1. Click delete → 2. Confirm → 3. Remove from table → 4. Persist
View Students	1. Load page → 2. Fetch & render student table

4.3 Class Diagram (Text Representation)

```
Class: Student
- Attributes: name, id, email, contact
- Methods: edit(), delete()

Class: StudentManager
- Attributes: students[]
- Methods: addStudent(), editStudent(), deleteStudent(), getStudents(), save(), load()
```

4.4 Sequence Diagram (Register Student)

```
Admin → Form: Input Data
Form → Validator: Check validity
Validator → Form: Valid/Errors
Form (if valid) → StudentManager: addStudent()
StudentManager → localStorage: Save
Form → Table: Render new row
```

4.5 Activity Diagram (Edit Student)

```
[Start] --> [Select Student (Edit)]
    --> [Display Form Pre-filled]
    --> [User Edits Data]
    --> [Validate]
        -[Invalid]--> [Show Error] --> [User Edits]
        -[Valid]--> [Update Student]
    --> [Persist in localStorage]
    --> [Update Table]
    --> [End]
```

4.6 Entity-Relationship Diagram (ER)

- **STUDENT** (student_id PK, name, email, contact)

5. Data Dictionary

Field	Type	Size	Description	Validation
name	String	60	Full name	Alphabets only, required
student_id	String	15	Unique identifier (alphanum)	Unique, required
email	String	100	Email address	Email format, required
contact	String	10/13	Contact number (+country code)	Digits, required

6. User Interface Design

6.1 Wireframes

(Draw via diagrams or describe here, e.g.):

- **Registration Form:**
    - Four labeled fields (vertical alignment), Submit & Reset buttons at bottom.
- **Student Table:**
    - Columns: Name, Student ID, Email, Contact, Actions (Edit/Delete icons on right).
- **Edit Modal:**
    - Same as Registration Form, prefilled.

6.2 Navigation Map

Home (Students Table) → Register Student Form → Table View → Edit/Delete Modal/Actions

6.3 Error Handling and Notifications

Error	Display Location	Description
Invalid input	At field	Red text/outline under invalid field
Duplicate ID	Modal/Inline	Warns if Student ID is not unique
Success	Top of form	Green notification message
Confirmation	Popup/modal	Delete/Edit confirmation

6.4 Accessibility Considerations

- ARIA labels on all form elements.
- Sufficient color contrast for visually impaired.

7. Appendices

7.1 Glossary

- **CRUD:** Create, Read, Update, Delete.
- **localStorage:** HTML5 key-value API for local persistence.

7.2 Analysis Models

- Include screenshots or sketched wireframes.

7.3 References

- W3C HTML5/CSS3
- IEEE Standards
- [Your institutional coding guidelines]

7.4 Revision History

Version	Date	Author	Changes Made
1.0	[Date]	Anmol Shukla	Initial release
1.1	[Date]	Anmol Shukla	Added detailed models