
National University of Computer and Emerging Sciences
FAST-NUCES

# Req2Design: Automated Requirement-to-Design Analysis System

*Mid Project Report*

## Submitted By:

| | |
|---|---|
| **Ali Rizwan** | 22i-2447 |
| **Muhammad Shahmeer** | 22i-1522 |
| **Abdul Haq Zulfiqar** | 22i-2585 |

## Supervisor:

Ma'am Isma Ul Hassan

## Co-Supervisor:

Sir Owais Idrees

October 17, 2025

# Contents


**1 Content for Your Mid Report** **3**
- 1.1 Introduction . . . 3
- 1.2 Problem Statement . . . 3
- 1.3 Scope . . . 4
- 1.4 Modules . . . 4
  - 1.4.1 Module 1: Requirement Intake & Preprocessing + SRS (Part 1) . . . 5
  - 1.4.2 Module 2: SRS (Part 2) Completion & Export . . . 5
  - 1.4.3 Module 3: Use Case Documentation . . . 5
  - 1.4.4 Module 4: Use Case Diagram Generator . . . 6
- 1.5 User Classes and Characteristics . . . 6

**2 Project Requirements** **9**
- 2.1 Use-case Diagram and Detailed Use Cases . . . 9
  - 2.1.1 Use Case Diagram . . . 9
  - 2.1.2 Use Case Diagram . . . 10
  - 2.1.3 Detailed Use Case Descriptions . . . 11
- 2.2 Functional Requirements . . . 19
  - 2.2.1 Module 1: Requirement Intake & Preprocessing + SRS (Part 1) . . . 19
  - 2.2.2 Module 2: SRS (Part 2) Completion & Export . . . 20
  - 2.2.3 Module 3: Use Case Documentation . . . 21
  - 2.2.4 Module 4: Use Case Diagram Generator . . . 22
- 2.3 Non-Functional Requirements . . . 23
  - 2.3.1 Reliability . . . 23
  - 2.3.2 Usability . . . 24
  - 2.3.3 Performance . . . 24
  - 2.3.4 Portability . . . 25
  - 2.3.5 Security . . . 26
- 2.4 Domain Model . . . 26

**3 System Design** **27**
- 3.1 System Overview . . . 27
- 3.2 Architectural Design . . . 28
  - 3.2.1 High-Level Architecture Diagram . . . 28
  - 3.2.2 Architecture Description . . . 28
- 3.3 Design Models . . . 29
  - 3.3.1 Object-Oriented Design Models . . . 29
  - 3.3.2 Activity Diagram . . . 30
  - 3.3.3 Class Diagram . . . 31
  - 3.3.4 Sequence Diagram . . . 32

3.3.5 State Transition Diagram . . . . . 33
3.4 Data Design . . . . . 34
3.4.1 Database Overview . . . . . 34
3.4.2 Logical Schema Overview . . . . . 34
3.5 System Interface Design . . . . . 34
3.5.1 External Interfaces . . . . . 34
3.5.2 User Interfaces . . . . . 35
3.6 Design Constraints . . . . . 35
3.7 Summary . . . . . 35

**4 Conclusion** **36**
4.1 Summary of the Project . . . . . 36
4.2 Key Achievements . . . . . 36
4.3 System Evaluation . . . . . 37
4.4 Limitations . . . . . 37
4.5 Future Work . . . . . 37
4.6 Conclusion . . . . . 38

# Chapter 1

# Content for Your Mid Report

## 1.1 Introduction

The product specified in this document is Req2Design, an automated requirement-to-design analysis system that transforms natural language requirements into standardized software engineering documentation. This document is intended for multiple reader types: software developers who will implement the system modules, project supervisors and evaluators (Ma'am Isma Ul Hassan and Sir Owais Idrees) who will assess project progress, software engineering students who represent the primary user base, startup founders and SMEs who will utilize the tool for rapid requirement documentation, and academic researchers interested in NLP-based automation for software engineering processes. The system addresses a critical gap in requirement engineering by providing IEEE 830-compliant SRS documents, Cockburn-template use cases, and UML 2.x diagrams through an intelligent, semi-automated workflow.

## 1.2 Problem Statement

Software requirement documentation is recognized as one of the most error-prone and time-consuming phases in the software development lifecycle, yet it forms the foundation for all subsequent design, implementation, and testing activities. Poorly documented or ambiguous requirements frequently lead to project delays, cost overruns, miscommunication among stakeholders, and in severe cases, complete project failure. Despite its critical importance, requirement documentation is often neglected, rushed, or performed without adherence to recognized industry standards such as IEEE 830 or established use case templates.

Current tools available in the market suffer from significant limitations: they are either excessively manual, requiring substantial user effort and expertise, or overly rigid, failing to accommodate unstructured or conversational requirement inputs. Most existing solutions lack compliance with recognized standards like IEEE 830 for Software Requirements Specification or UML 2.x guidelines for diagrammatic representations, rendering them unsuitable for academic evaluation or professional project documentation. Furthermore, automated tools, where they exist, typically generate only plain text outputs and fail to enforce critical quality attributes such as completeness, traceability, consistency, and clarity across requirement artifacts.

Use case generation tools rarely incorporate established templates like Alistair Cockburn's format, which is widely taught in software engineering curricula and expected in both academic and professional contexts. These gaps collectively result in documentation that is fragmented, incomplete, inconsistent, and ultimately not suitable for practi-

cal software engineering tasks or formal evaluation. Req2Design directly addresses these challenges by combining advanced natural language processing techniques with standards-based output generation, ensuring both intelligent automation and technical correctness while significantly reducing the time and effort required for high-quality requirement documentation.

## 1.3 Scope

Req2Design aims to automate the creation of structured, standard-compliant requirement documentation for software engineering projects through a four-module intelligent workflow system. The project scope encompasses the generation of IEEE 830-compliant Software Requirements Specification (SRS) documents covering essential sections including Introduction, Overall Description, Functional Requirements (with structured numbering such as FR-01, FR-02), core Non-Functional Requirements (usability, reliability, performance, portability), and External Interface specifications.

The system will accept requirements in multiple input formats including typed text, pasted content, and audio recordings, with automatic transcription capabilities using speech-to-text technology. Natural language processing techniques will be employed to preprocess raw inputs, identify ambiguities, resolve inconsistencies, and prompt users for clarification where necessary. Beyond SRS generation, the system will automatically produce textual use case documentation following the Alistair Cockburn template format, including actors, goals, preconditions, main flow of events, and postconditions for each identified use case.

The third major component involves automated UML 2.x Use Case Diagram generation directly derived from textual use cases, ensuring compliance with UML standards including proper system boundaries, actor placement, and relationship representations. All generated artifacts will be exportable in multiple academic-friendly formats including PDF and DOCX for immediate integration into university submissions, research papers, or professional documentation.

The system architecture will follow a modular design with clear separation of concerns: Module 1 handles requirement intake and preprocessing along with initial SRS sections; Module 2 completes the SRS with functional/non-functional requirements and external interfaces; Module 3 generates Cockburn-style use case documentation; and Module 4 produces UML diagrams with export capabilities. The scope intentionally excludes advanced NFR categories such as scalability, maintainability, and security; traceability matrices; feasibility studies; and design-level diagrams beyond use cases (such as class diagrams or sequence diagrams) to maintain project feasibility within the Final Year Project timeframe while ensuring that core deliverables are comprehensive, standards-compliant, and genuinely useful for the target stakeholders.

## 1.4 Modules

The Req2Design system is architecturally divided into four integrated modules, each responsible for a distinct phase of the requirement-to-design automation workflow.

### 1.4.1 Module 1: Requirement Intake & Preprocessing + SRS (Part 1)

This module serves as the entry point for the system, responsible for capturing raw requirements from users and converting them into structured, analyzable content. It handles both textual inputs (typed or pasted) and audio recordings, employing speech-to-text transcription engines such as Whisper for audio processing. The module applies NLP preprocessing techniques to clean raw text, identify ambiguous terms, and detect inconsistencies, prompting users for clarification when necessary. Based on preprocessed inputs, it generates the initial sections of the IEEE 830-compliant SRS document.

**Features:**

1. Multi-format input collection (text and audio)
2. Automatic speech-to-text transcription using Whisper
3. NLP-based text preprocessing and ambiguity detection
4. Interactive clarification prompts for unclear requirements
5. Automated generation of SRS Introduction section
6. Automated generation of SRS Overall Description section

### 1.4.2 Module 2: SRS (Part 2) Completion & Export

This module completes the IEEE 830-compliant SRS document by generating the remaining critical sections. It structures functional requirements with clear numbering conventions, documents core non-functional requirements in measurable terms, and specifies external interfaces defining system boundaries. The module ensures consistency and completeness across all SRS sections before exporting the final document.

**Features:**

1. Structured Functional Requirements generation (FR-01, FR-02 format)
2. Core Non-Functional Requirements documentation (usability, reliability, performance, portability)
3. External Interface specifications (user, hardware, software, communication)
4. SRS completeness validation and consistency checking
5. PDF export functionality
6. DOCX export functionality

### 1.4.3 Module 3: Use Case Documentation

This module transforms textual requirements into structured use case documentation following the Alistair Cockburn template. It intelligently extracts actors and goals from requirement text and generates comprehensive use cases with essential elements while providing editing capabilities for user refinement.

**Features:**

1. Automatic actor identification and extraction
2. Goal-based use case generation
3. Cockburn template application (Actor, Goal, Preconditions, Main Flow, Postconditions)
4. Use case editing and refinement interface
5. Multi-use case management and organization
6. Textual use case export capabilities

### 1.4.4 Module 4: Use Case Diagram Generator

This module automatically creates visual UML 2.x Use Case Diagrams from the textual use cases generated in Module 3. It applies UML validation rules to ensure diagrams comply with industry standards and supports multiple export formats for integration with various documentation tools.

**Features:**

1. Automatic UML 2.x Use Case Diagram generation from textual use cases
2. UML validation rules enforcement (system boundaries, actor placement, relationships)
3. SVG format export
4. PNG format export
5. PlantUML source code export
6. Interactive diagram editing and refinement capabilities

## 1.5 User Classes and Characteristics

| User Class | Description |
|---|---|
| Software Engineering Students | Students enrolled in software engineering or computer science programs who need to create requirement documentation for academic projects, assignments, and final year projects. Approximately 500-1000 potential student users across various universities. Students typically have theoretical knowledge of SRS and use cases but lack practical experience in creating comprehensive documentation. They require guided workflows, template-based generation, and educational feedback. An estimated 70% will use the system for course projects, with 30% utilizing it for FYP documentation. Students expect the system to reduce documentation time by 60-70% while ensuring compliance with academic standards and supervisor expectations. |
| Startup Founders & SME Project Managers | Early-stage startup founders and small-to-medium enterprise project managers who need to rapidly produce requirement documentation for new projects but lack dedicated business analysts or requirement engineering expertise. This user class comprises approximately 200-300 active users managing 2-5 projects annually. They have strong domain knowledge but limited time for detailed documentation tasks. These users prioritize speed, clarity, and export-ready formats that can be shared with development teams and stakeholders. They expect to generate complete SRS and use case documentation in under 2 hours compared to 8-10 hours with manual approaches. |
| Project Supervisors & Evaluators | Academic supervisors, course instructors, and industry evaluators who assess student projects and need to review standardized requirement documentation. This class includes approximately 50-100 faculty members and industry mentors. They require documents that strictly adhere to IEEE 830 and UML standards for fair and consistent evaluation. Supervisors expect clear traceability between requirements and use cases, proper formatting, and completeness in all mandatory sections. They will use the generated documents as evaluation criteria for project proposals, mid-term reports, and final submissions. |

| **User Class** | **Description** |
|---|---|
| Software Developers & Technical Teams | Professional software developers and technical team members who utilize requirement documentation as input for system design and implementation. This user class includes 100-200 developers working in agile or traditional development environments. Developers need clear, unambiguous functional requirements, well-defined use cases with complete flows, and visual UML diagrams for system understanding. They expect requirements to be traceable, consistently formatted, and exportable in formats compatible with project management and development tools. An estimated 40% will use the system during sprint planning, with 60% referencing documentation during development phases. |

# Chapter 2

# Project Requirements

This chapter describes the functional and non-functional requirements of the Req2Design system, an automated requirement-to-design analyzer that transforms natural language inputs into standardized software engineering documentation.

## 2.1 Use-case Diagram and Detailed Use Cases

Since Req2Design is an interactive end-user application involving multiple user classes (students, startup founders, supervisors, and developers), the use case technique is the most appropriate requirement gathering approach. The following sections present the use case diagram and detailed use case descriptions.

### 2.1.1 Use Case Diagram

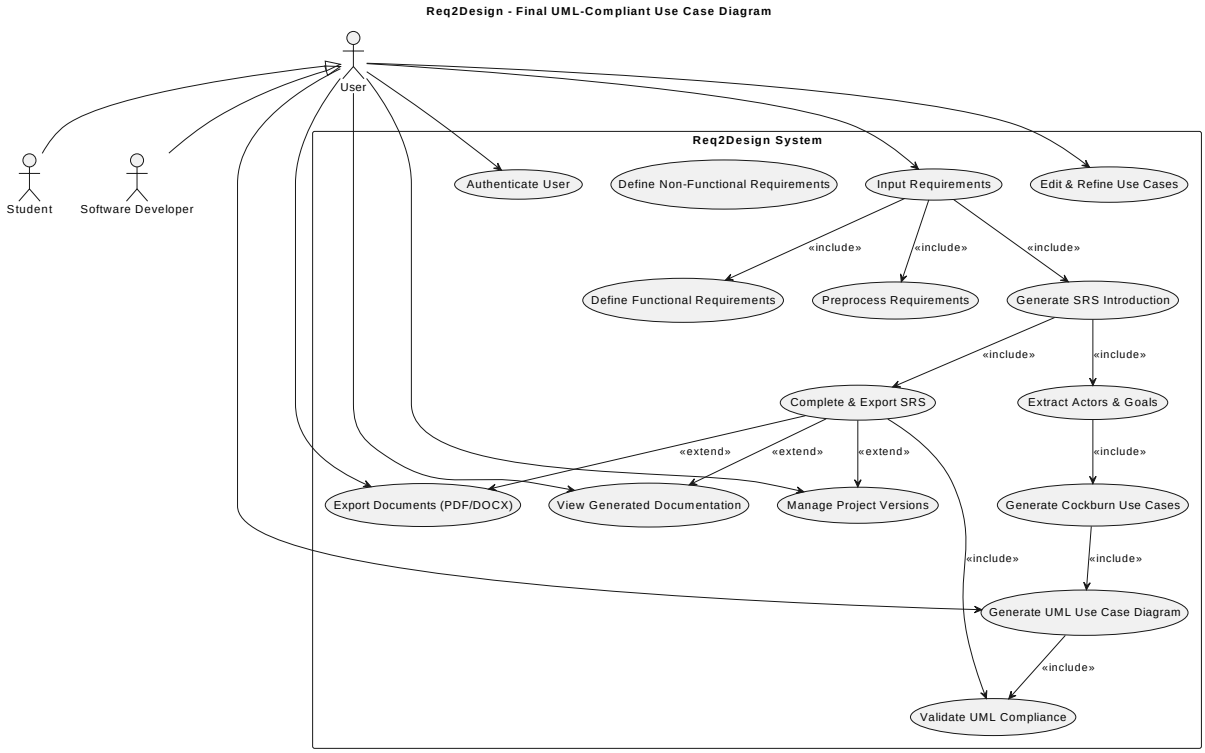



Figure 2.1: Req2Design Use Case Diagram

### 2.1.2 Use Case Diagram

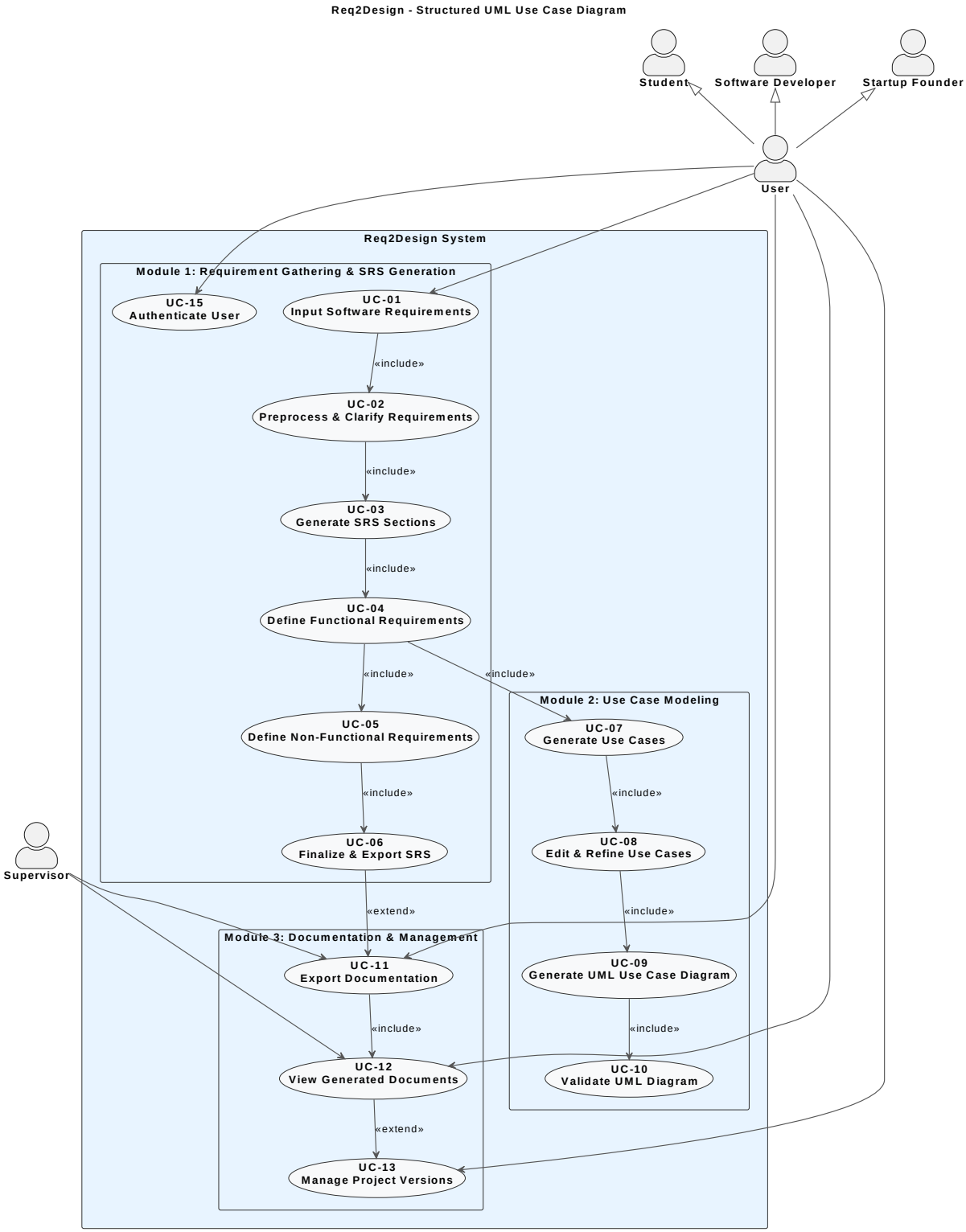



Figure 2.2: Req2Design Use Case Diagram1

### 2.1.3 Detailed Use Case Descriptions

**Use Case 1: Input Requirements**

| | |
|---|---|
| **Use Case ID** | UC-01 |
| **Use Case Name** | Input Requirements |
| **Actor(s)** | Student, Developer, Startup Founder |
| **Goal** | To capture software requirements from the user in text or audio format. |
| **Preconditions** | User has registered and logged into the Req2Design system. |
| **Main Flow** | 1. User selects input method (text or audio).<br>2. System displays appropriate input interface.<br>3. User enters requirements via typing, pasting, or audio recording.<br>4. System validates input format.<br>5. System stores raw requirements.<br>6. System displays confirmation message. |
| **Postconditions** | Requirements are successfully stored in the system database for further processing. |

**Use Case 2: Preprocess Requirements**

| | |
|---|---|
| **Use Case ID** | UC-02 |
| **Use Case Name** | Preprocess Requirements |
| **Actor(s)** | System (automated), User (for clarification) |
| **Goal** | To clean, analyze, and resolve ambiguities in raw requirement text. |
| **Preconditions** | Raw requirements have been captured via UC-01. |
| **Main Flow** | 1. System retrieves raw requirements.<br>2. System applies NLP preprocessing (tokenization, lemmatization).<br>3. System detects ambiguous terms and inconsistencies.<br>4. System prompts user for clarification.<br>5. User provides clarifications.<br>6. System updates clarified requirements.<br>7. System marks requirements as preprocessed. |
| **Postconditions** | Requirements are cleaned, disambiguated, and ready for SRS generation. |

**Use Case 3: Generate SRS Introduction**

| | |
|---|---|
| **Use Case ID** | UC-03 |
| **Use Case Name** | Generate SRS Introduction |
| **Actor(s)** | System (automated) |
| **Goal** | To automatically generate the Introduction and Overall Description sections of IEEE 830 SRS. |
| **Preconditions** | Requirements have been preprocessed via UC-02. |
| **Main Flow** | 1. System analyzes preprocessed requirements.<br>2. System extracts system purpose and scope.<br>3. System generates Introduction (Purpose, Scope, Definitions).<br>4. System generates Overall Description (Product Perspective, Functions, User Characteristics).<br>5. System stores generated sections.<br>6. System notifies user of completion. |
| **Postconditions** | SRS Introduction and Overall Description sections are generated and stored. |

**Use Case 4: Define Functional Requirements**

| | |
|---|---|
| **Use Case ID** | UC-04 |
| **Use Case Name** | Define Functional Requirements |
| **Actor(s)** | Student, Developer, Startup Founder |
| **Goal** | To specify detailed functional requirements in structured format. |
| **Preconditions** | SRS Introduction has been generated via UC-03. |
| **Main Flow** | 1. System extracts functional requirements.<br>2. System assigns structured numbering (FR-01, FR-02, etc.).<br>3. System presents draft requirements to user.<br>4. User reviews and edits requirements.<br>5. User confirms final requirements.<br>6. System validates completeness.<br>7. System stores functional requirements. |
| **Postconditions** | Functional requirements are defined, numbered, and stored in the SRS document. |

**Use Case 5: Define Non-Functional Requirements**

| | |
|---|---|
| **Use Case ID** | UC-05 |
| **Use Case Name** | Define Non-Functional Requirements |
| **Actor(s)** | Student, Developer, Startup Founder |
| **Goal** | To specify non-functional requirements (usability, reliability, performance, portability). |
| **Preconditions** | Functional requirements have been defined via UC-04. |
| **Main Flow** | 1. System identifies quality attributes.<br>2. System categorizes NFRs.<br>3. System presents draft NFRs.<br>4. User reviews and specifies measurable criteria.<br>5. User confirms NFRs.<br>6. System stores NFRs in SRS. |
| **Postconditions** | Non-functional requirements are documented with measurable criteria. |

**Use Case 6: Complete & Export SRS**

| | |
|---|---|
| **Use Case ID** | UC-06 |
| **Use Case Name** | Complete and Export SRS |
| **Actor(s)** | Student, Developer, Startup Founder |
| **Goal** | To finalize the SRS and export it in PDF/DOCX format. |
| **Preconditions** | All SRS sections have been completed. |
| **Main Flow** | 1. User requests SRS review.<br>2. System validates sections.<br>3. System checks for completeness.<br>4. System generates table of contents and formatting.<br>5. User selects export format.<br>6. System generates formatted document.<br>7. System provides download link.<br>8. User downloads the SRS. |
| **Postconditions** | IEEE 830-compliant SRS document is exported and available for download. |

**Use Case 7: Extract Actors & Goals**

| | |
|---|---|
| **Use Case ID** | UC-07 |
| **Use Case Name** | Extract Actors and Goals |
| **Actor(s)** | System (automated) |
| **Goal** | To identify actors and their goals from requirement text. |
| **Preconditions** | SRS functional requirements have been defined via UC-04. |
| **Main Flow** | 1. System analyzes functional requirements.<br>2. System applies NLP entity recognition.<br>3. System extracts action verbs to determine goals.<br>4. System maps actors to goals.<br>5. System creates preliminary actor-goal pairs.<br>6. System displays extracted information. |
| **Postconditions** | Actors and goals are identified for use case generation. |

**Use Case 8: Generate Cockburn Use Cases**

| | |
|---|---|
| **Use Case ID** | UC-08 |
| **Use Case Name** | Generate Cockburn Use Cases |
| **Actor(s)** | System (automated) |
| **Goal** | To automatically generate textual use cases following the Cockburn template. |
| **Preconditions** | Actors and goals have been extracted via UC-07. |
| **Main Flow** | 1. System retrieves actor-goal pairs.<br>2. Generates use case structure for each pair.<br>3. Populates Actor and Goal fields.<br>4. Identifies Preconditions and Main Flow.<br>5. Documents Postconditions.<br>6. Stores generated use cases.<br>7. Displays use cases to user. |
| **Postconditions** | Textual use cases are generated in Cockburn format. |

**Use Case 9: Edit & Refine Use Cases**

| | |
|---|---|
| **Use Case ID** | UC-09 |
| **Use Case Name** | Edit and Refine Use Cases |
| **Actor(s)** | Student, Developer, Startup Founder |
| **Goal** | To review and refine automatically generated use cases. |
| **Preconditions** | Use cases have been generated via UC-08. |
| **Main Flow** | 1. User selects a use case to edit.<br>2. System displays use case in editable format.<br>3. User modifies details.<br>4. User saves changes.<br>5. System validates completeness.<br>6. Updates use case in database. |
| **Postconditions** | Use cases are refined and validated by the user. |

**Use Case 10: Generate UML Use Case Diagram**

| | |
|---|---|
| **Use Case ID** | UC-10 |
| **Use Case Name** | Generate UML Use Case Diagram |
| **Actor(s)** | Student, Developer, Startup Founder |
| **Goal** | To automatically create UML 2.x compliant use case diagrams. |
| **Preconditions** | Textual use cases have been finalized via UC-09. |
| **Main Flow** | 1. User requests diagram generation.<br>2. System retrieves use cases.<br>3. Identifies unique actors.<br>4. Creates UML elements (actors, use cases, associations).<br>5. Applies layout rules.<br>6. Generates PlantUML/Mermaid syntax.<br>7. Renders and displays diagram. |
| **Postconditions** | UML Use Case Diagram is generated and displayed. |

**Use Case 11: Validate UML Compliance**

| | |
|---|---|
| **Use Case ID** | UC-11 |
| **Use Case Name** | Validate UML Compliance |
| **Actor(s)** | System (automated) |
| **Goal** | To ensure generated diagrams comply with UML 2.x standards. |
| **Preconditions** | UML diagram has been generated via UC-10. |
| **Main Flow** | 1. System analyzes diagram structure.<br>2. Validates actor placement and boundaries.<br>3. Verifies association types (include, extend).<br>4. Checks naming conventions.<br>5. Generates validation report.<br>6. Highlights issues and suggests fixes. |
| **Postconditions** | Diagram compliance is validated and issues are reported. |

**Use Case 12: Export Documents (PDF/DOCX)**

| | |
|---|---|
| **Use Case ID** | UC-12 |
| **Use Case Name** | Export Documents |
| **Actor(s)** | Student, Developer, Startup Founder, Supervisor |
| **Goal** | To export generated documentation in PDF or DOCX format. |
| **Preconditions** | SRS, use cases, or diagrams have been generated. |
| **Main Flow** | 1. User selects document(s) to export.<br>2. Selects export format.<br>3. System compiles selected documentation.<br>4. Applies formatting.<br>5. Generates export file.<br>6. Provides download link.<br>7. User downloads exported document. |
| **Postconditions** | Documentation is exported and available for download. |

**Use Case 13: View Generated Documentation**

| | |
|---|---|
| **Use Case ID** | UC-13 |
| **Use Case Name** | View Generated Documentation |
| **Actor(s)** | Student, Developer, Startup Founder, Supervisor |
| **Goal** | To view all generated documentation artifacts. |
| **Preconditions** | User has logged in and has at least one project with generated documents. |
| **Main Flow** | 1. User navigates to documentation view.<br>2. System displays list of documents.<br>3. User selects a document.<br>4. System renders the document.<br>5. User reviews and navigates through sections. |
| **Postconditions** | User successfully views generated documentation. |

**Use Case 14: Manage Project Versions**

| | |
|---|---|
| **Use Case ID** | UC-14 |
| **Use Case Name** | Manage Project Versions |
| **Actor(s)** | Student, Developer, Startup Founder |
| **Goal** | To create, view, and restore different versions of project documentation. |
| **Preconditions** | User has at least one project with generated documentation. |
| **Main Flow** | 1. User navigates to version management.<br>2. System displays list of saved versions.<br>3. User creates a version snapshot.<br>4. System saves current state.<br>5. User can view or restore a version.<br>6. System confirms restoration. |
| **Postconditions** | Project versions are managed (created, viewed, or restored). |

**Use Case 15: Authenticate User**

| | |
|---|---|
| **Use Case ID** | UC-15 |
| **Use Case Name** | Authenticate User |
| **Actor(s)** | Student, Developer, Startup Founder, Supervisor |
| **Goal** | To verify user identity and grant access. |
| **Preconditions** | User has a registered account. |
| **Main Flow** | 1. User navigates to login page.<br>2. System displays authentication form.<br>3. User enters credentials.<br>4. System validates credentials.<br>5. Creates user session.<br>6. Redirects to dashboard. |
| **Postconditions** | User is authenticated and granted access. |

## 2.2 Functional Requirements

This section describes the functional requirements organized by the four modules of Req2Design.

### 2.2.1 Module 1: Requirement Intake & Preprocessing + SRS (Part 1)

**FR1.1:** The system shall allow users to input software requirements via text entry (typing or pasting).

**FR1.2:** The system shall allow users to input software requirements via audio recording with a maximum duration of 10 minutes per recording.

**FR1.3:** The system shall transcribe audio inputs to text using Whisper speech-to-text engine with 95% accuracy for clear audio.

**FR1.4:** The system shall apply NLP preprocessing techniques including tokenization, lemmatization, and stop-word removal to raw requirement text.

**FR1.5:** The system shall detect ambiguous terms and inconsistent statements in requirement text and highlight them for user review.

**FR1.6:** The system shall prompt users with specific clarification questions when ambiguities are detected, providing context for each ambiguous term.

**FR1.7:** The system shall allow users to provide clarifications inline and update requirement text accordingly.

**FR1.8:** The system shall automatically generate the Introduction section of IEEE 830 SRS including Purpose, Scope, Definitions, and References subsections.

**FR1.9:** The system shall automatically generate the Overall Description section of IEEE 830 SRS including System Perspective, Product Functions, User Characteristics, and General Constraints subsections.

**FR1.10:** The system shall extract system purpose and scope information from preprocessed requirements using NLP semantic analysis.

**FR1.11:** The system shall identify and extract key terminology for the Definitions subsection of the SRS Introduction.

**FR1.12:** The system shall allow users to review and edit the generated Introduction and Overall Description sections before finalizing.

**FR1.13:** The system shall maintain version history of requirement inputs and allow users to revert to previous versions.

**FR1.14:** The system shall support batch import of requirements from text files (.txt, .doc, .docx) with a maximum file size of 5MB.

**FR1.15:** The system shall display preprocessing progress indicators showing completion percentage and current processing stage.

### 2.2.2 Module 2: SRS (Part 2) Completion & Export

**FR2.1:** The system shall automatically extract and structure functional requirements from preprocessed text with sequential numbering (FR-01, FR-02, etc.).

**FR2.2:** The system shall ensure each functional requirement follows the format: "The system shall [action] [object] [condition]".

**FR2.3:** The system shall categorize functional requirements by feature or module groupings when applicable.

**FR2.4:** The system shall allow users to add, edit, delete, and reorder functional requirements with automatic renumbering.

**FR2.5:** The system shall automatically identify and document non-functional requirements in four core categories: usability, reliability, performance, and portability.

**FR2.6:** The system shall prompt users to specify measurable criteria for each non-functional requirement (e.g., response time $< 2$ seconds).

**FR2.7:** The system shall generate the External Interfaces section specifying user interfaces, hardware interfaces, software interfaces, and communication interfaces.

**FR2.8:** The system shall validate SRS completeness by checking that all mandatory IEEE 830 sections are present and non-empty.

**FR2.9:** The system shall check for consistency between different SRS sections and highlight potential conflicts.

**FR2.10:** The system shall automatically generate a Table of Contents with page numbers for the complete SRS document.

**FR2.11:** The system shall export the complete SRS document in PDF format with proper formatting, headers, footers, and page numbering.

**FR2.12:** The system shall export the complete SRS document in DOCX format preserving all formatting, styles, and structure.

**FR2.13:** The system shall include document metadata (author, date, version) in exported SRS documents.

**FR2.14:** The system shall generate a requirements summary dashboard showing total count of FRs, NFRs, and document completion percentage.

**FR2.15:** The system shall allow users to preview the SRS document before export with page-by-page navigation.

### 2.2.3 Module 3: Use Case Documentation

**FR3.1:** The system shall automatically extract actors from functional requirements by identifying nouns representing users or external systems.

**FR3.2:** The system shall automatically extract goals for each actor by identifying action verbs and objectives from requirement text.

**FR3.3:** The system shall generate textual use cases following the Alistair Cockburn template structure with fields: Actor, Goal, Preconditions, Main Flow, and Postconditions.

**FR3.4:** The system shall generate a unique identifier for each use case using the format UC-XX where XX is a sequential number.

**FR3.5:** The system shall automatically populate the Main Flow section with numbered steps derived from requirement text.

**FR3.6:** The system shall identify and document preconditions by analyzing prerequisites mentioned in requirements.

**FR3.7:** The system shall identify and document postconditions by analyzing expected outcomes and system state changes.

**FR3.8:** The system shall provide an interactive editor for users to modify any field within generated use cases.

**FR3.9:** The system shall allow users to add new use cases manually using the Cockburn template structure.

**FR3.10:** The system shall allow users to delete use cases with confirmation prompts to prevent accidental deletion.

**FR3.11:** The system shall validate use case completeness by ensuring all mandatory fields (Actor, Goal, Main Flow) are filled.

**FR3.12:** The system shall display all use cases in a searchable and filterable list view organized by actor or goal.

**FR3.13:** The system shall support merging multiple similar use cases identified by semantic similarity analysis.

**FR3.14:** The system shall export textual use cases as a standalone document in PDF or DOCX format.

**FR3.15:** The system shall display relationships between use cases when common actors or goals are detected.

### 2.2.4 Module 4: Use Case Diagram Generator

**FR4.1:** The system shall automatically generate UML 2.x compliant use case diagrams from textual use cases using PlantUML or Mermaid.js syntax.

**FR4.2:** The system shall identify unique actors from all use cases and represent them as stick figures positioned outside the system boundary.

**FR4.3:** The system shall create a system boundary rectangle enclosing all use case ovals with the system name displayed at the top.

**FR4.4:** The system shall represent each use case as an oval with the use case name centered inside.

**FR4.5:** The system shall create association lines connecting actors to their respective use cases.

**FR4.6:** The system shall apply UML validation rules to ensure actors are placed outside the system boundary.

**FR4.7:** The system shall validate that use case names are clear, concise, and follow verb-noun format.

**FR4.8:** The system shall detect and represent include relationships when one use case always invokes another.

**FR4.9:** The system shall detect and represent extend relationships when one use case optionally extends another under specific conditions.

**FR4.10:** The system shall provide automatic layout optimization to minimize line crossings and improve diagram readability.

**FR4.11:** The system shall allow users to manually adjust actor and use case positions within the diagram editor.

**FR4.12:** The system shall export diagrams in SVG format for scalable, high-quality vector graphics.

**FR4.13:** The system shall export diagrams in PNG format with customizable resolution (minimum 300 DPI).

**FR4.14:** The system shall export PlantUML source code to allow users to further customize diagrams in external tools.

**FR4.15:** The system shall provide real-time preview of diagram changes when users edit use case text or diagram layout.

**FR4.16:** The system shall generate a validation report listing any UML standard violations detected in the diagram.

**FR4.17:** The system shall support exporting multiple diagram versions (with and without relationships) for different documentation purposes.

## 2.3 Non-Functional Requirements

This section specifies the non-functional requirements for the Req2Design system, organized by quality attributes.

### 2.3.1 Reliability

**REL-1:** The system shall maintain 99.5% uptime during business hours (9 AM to 6 PM local time), measured monthly. The measurement excludes scheduled maintenance windows which must be announced 48 hours in advance.

**REL-2:** The system shall automatically recover from transient failures (such as network interruptions or temporary service unavailability) within 30 seconds without data loss.

**REL-3:** The system shall perform automated backups of all user data (requirements, SRS documents, use cases, diagrams) every 6 hours, with backup retention for 30 days.

**REL-4:** The system shall maintain data integrity such that 99.9% of all save operations complete successfully without data corruption or loss.

**REL-5:** In the event of system failure, the system shall preserve all user work from the last auto-save point (maximum 2 minutes of work loss).

**REL-6:** The system shall log all critical errors with timestamps, error codes, and user context to enable debugging and failure analysis.

**REL-7:** The mean time between failures (MTBF) for core functionalities (requirement input, SRS generation, use case generation, diagram generation) shall be at least 720 hours (30 days).

**REL-8:** The system shall gracefully degrade functionality when external services (such as NLP APIs or transcription services) are unavailable, notifying users and queueing requests for later processing.

### 2.3.2 Usability

**USE-1:** 90% of new users shall be able to successfully input requirements and generate a basic SRS document within 15 minutes without referring to documentation or help resources.

**USE-2:** The system shall provide contextual help tooltips for all major features, accessible by hovering over or clicking information icons adjacent to feature controls.

**USE-3:** The system shall provide an interactive onboarding tutorial for first-time users, guiding them through the complete workflow from requirement input to document export in under 10 minutes.

**USE-4:** The system shall support undo and redo operations for all user actions (editing, deleting, adding content) with a history depth of at least 20 actions.

**USE-5:** The system shall display processing status indicators with estimated time remaining for long-running operations (transcription, NLP processing, diagram generation).

**USE-6:** The system shall provide clear error messages that explain what went wrong and suggest corrective actions, avoiding technical jargon for end-user errors.

**USE-7:** The system shall maintain consistent navigation patterns across all modules, with primary navigation always visible and requiring no more than 3 clicks to reach any feature.

**USE-8:** The system shall support keyboard shortcuts for common operations (save, export, undo, redo) to improve power user efficiency.

**USE-9:** The system interface shall be responsive and functional on desktop browsers (minimum screen resolution 1366x768) and tablet devices (minimum screen width 768px).

**USE-10:** The system shall retrieve and display previously generated SRS sections or use cases with a single interaction (one click) for iterative editing.

**USE-11:** The system shall provide a search functionality allowing users to find specific requirements, use cases, or content within documents with results displayed within 1 second.

**USE-12:** The system shall conform to WCAG 2.1 Level AA accessibility standards, including sufficient color contrast (minimum 4.5:1), keyboard navigation support, and screen reader compatibility.

### 2.3.3 Performance

**PER-1:** The system shall transcribe audio recordings to text at a rate of at least 1 minute of audio per 15 seconds of processing time (4x real-time speed).

**PER-2:** The system shall complete NLP preprocessing of requirement text (up to 5000 words) within 10 seconds.

**PER-3:** The system shall generate SRS Introduction and Overall Description sections within 15 seconds after preprocessing completion.

**PER-4:** The system shall generate functional and non-functional requirements sections within 20 seconds for projects with up to 50 requirements.

**PER-5:** The system shall generate textual use cases (up to 20 use cases) within 25 seconds from functional requirements.

**PER-6:** The system shall generate UML use case diagrams (containing up to 15 use cases and 10 actors) within 10 seconds.

**PER-7:** 95% of web pages within the Req2Design interface shall load completely within 3 seconds over a 10 Mbps or faster internet connection.

**PER-8:** The system shall support concurrent usage by at least 100 simultaneous users without performance degradation (response time increase $> 20\%$).

**PER-9:** The system shall export SRS documents in PDF format (up to 50 pages) within 8 seconds.

**PER-10:** The system shall export SRS documents in DOCX format (up to 50 pages) within 10 seconds.

**PER-11:** Database queries for retrieving user projects, requirements, or use cases shall execute within 500 milliseconds for 95% of requests.

**PER-12:** The system shall handle file uploads (audio or text) up to 5MB within 15 seconds, with progress indicators updated every 2 seconds.

### 2.3.4 Portability

**PORT-1:** The system's frontend shall be compatible with and fully functional on the latest two major versions of Chrome, Firefox, Safari, and Edge browsers.

**PORT-2:** The system shall function correctly on Windows 10/11, macOS 11+, and major Linux distributions (Ubuntu 20.04+, Fedora 34+) without requiring operating system-specific installations.

**PORT-3:** The system's backend shall be containerized using Docker to enable deployment on any Docker-compatible hosting platform (AWS, Azure, Heroku, Google Cloud).

**PORT-4:** The system shall provide RESTful APIs with standardized JSON request/response formats to enable integration with external tools and future mobile applications.

**PORT-5:** The system shall export all documentation in platform-independent formats (PDF, DOCX, SVG, PNG) that can be opened on any operating system.

**PORT-6:** The system's database schema shall be portable across PostgreSQL versions 12.x through 15.x without requiring schema modifications.

**PORT-7:** The system shall support data export in standard formats (JSON, CSV) to enable migration to other platforms or backup to external storage.

**PORT-8:** The system's architecture shall separate frontend and backend components to enable independent deployment and scaling on different infrastructure platforms.

### 2.3.5 Security

**SEC-1:** The system shall encrypt all user passwords using bcrypt hashing algorithm with a minimum work factor of 12 before storing in the database.

**SEC-2:** The system shall implement user authentication using JWT (JSON Web Tokens) with token expiration set to 24 hours.

**SEC-3:** The system shall enforce HTTPS encryption for all client-server communications using TLS 1.2 or higher.

**SEC-4:** The system shall implement role-based access control (RBAC) ensuring users can only access their own projects and documents.

**SEC-5:** The system shall log all authentication attempts (successful and failed) with timestamps and IP addresses for security auditing.

**SEC-6:** The system shall implement rate limiting on API endpoints to prevent abuse, allowing maximum 100 requests per user per minute.

**SEC-7:** The system shall validate and sanitize all user inputs to prevent SQL injection, XSS (Cross-Site Scripting), and other injection attacks.

## 2.4 Domain Model

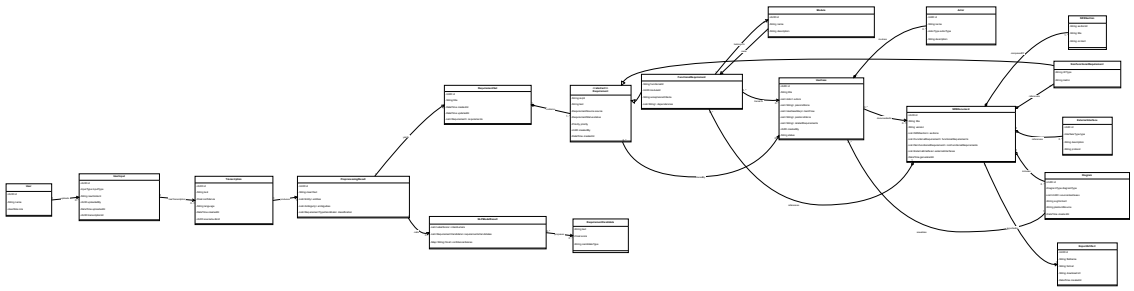


Figure 2.3: Req2Design Domain Model

# Chapter 3

# System Design

## 3.1 System Overview

The overall design of the Req2Design system follows a modular, service-oriented architecture that integrates Natural Language Processing (NLP), Speech-to-Text (STT), and UML generation technologies within a unified web application framework. The design emphasizes **scalability**, **modularity**, and **user-centric workflow automation** to ensure a seamless transformation of natural language requirements into structured IEEE 830-compliant SRS documents, Cockburn-style use cases, and UML 2.x diagrams.

The architecture is divided into four logical layers:

1. **Presentation Layer** – Provides the user interface for requirement input, project management, and document visualization.
2. **Application Layer** – Implements the business logic, NLP processing, and AI-driven automation workflows.
3. **Data Layer** – Manages persistent storage of requirements, intermediate artifacts, and final documents.
4. **Integration Layer** – Handles external services such as Whisper (speech-to-text), OpenAI NLP models, and PlantUML/Mermaid APIs for diagram generation.

This separation of concerns ensures that each component can be developed, tested, and maintained independently while maintaining interoperability through RESTful APIs and standardized data exchange formats (JSON).

## 3.2 Architectural Design

### 3.2.1 High-Level Architecture Diagram

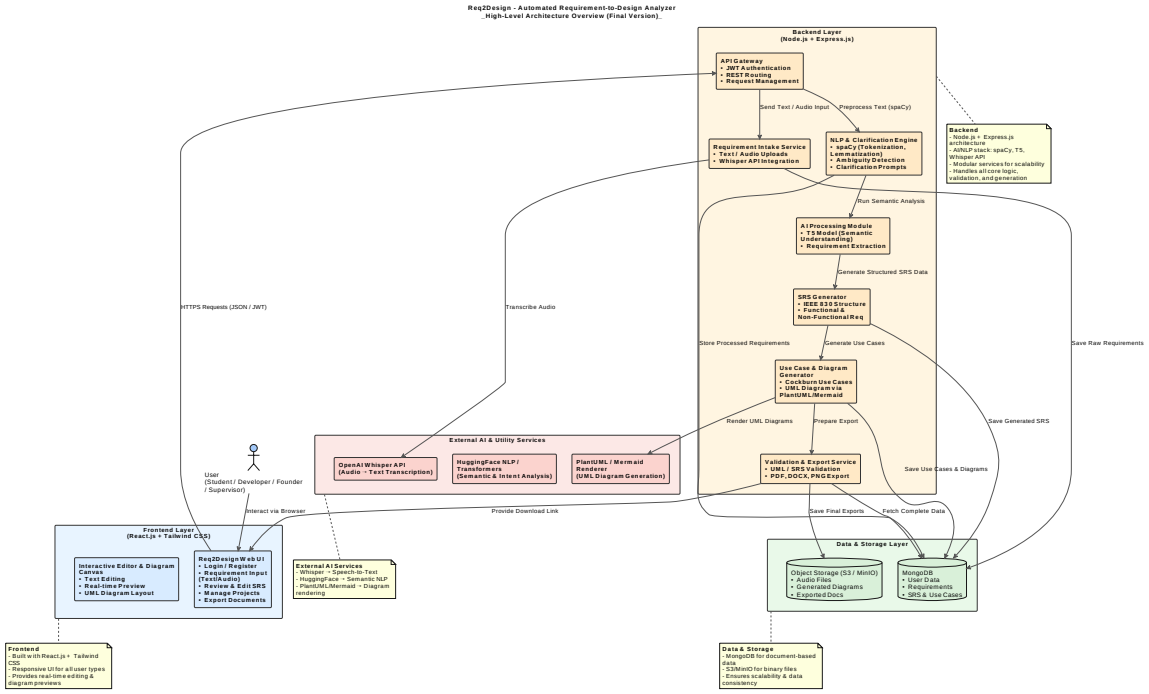



Figure 3.1: High-Level Architecture of Req2Design System

The figure above illustrates the modular architecture of the Req2Design system, highlighting the key layers and their interactions.

### 3.2.2 Architecture Description

The Req2Design architecture enables end-to-end automation, from raw requirement intake to the export of fully formatted documentation. It supports continuous feedback loops and interactive refinement options through modular subsystems as shown below:

- **Presentation Layer:** Web-based interfaces for requirement input, audio capture, and report visualization.
- **Application Layer:** Core logic modules—NLP Processor, SRS Generator, and Use Case/UML Extractor.
- **Integration Layer:** Connects external APIs for speech-to-text and diagram generation.
- **Data Layer:** Stores user information, requirements, project data, and generated documents.

## 3.3 Design Models

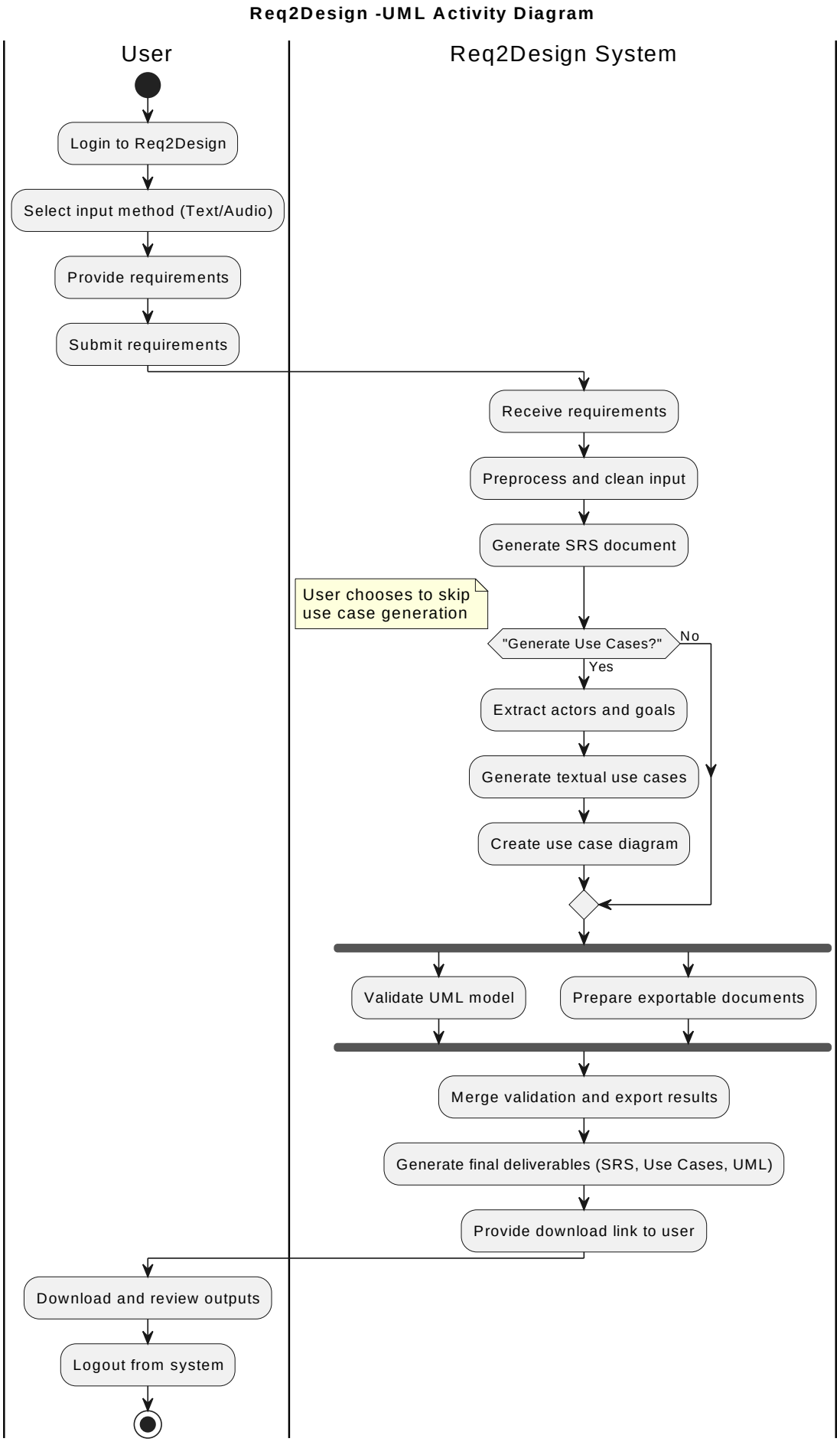
### 3.3.1 Object-Oriented Design Models

This project follows an Object-Oriented Design approach. The key UML-based design models included are:

- Activity Diagram
- Class Diagram
- Sequence Diagram (Class-level)
- State Transition Diagram

Each diagram provides a specific perspective of the system's structure and behavior. The following subsections include representative UML diagrams for the Req2Design system.

### 3.3.2 Activity Diagram



### 3.3.3 Class Diagram

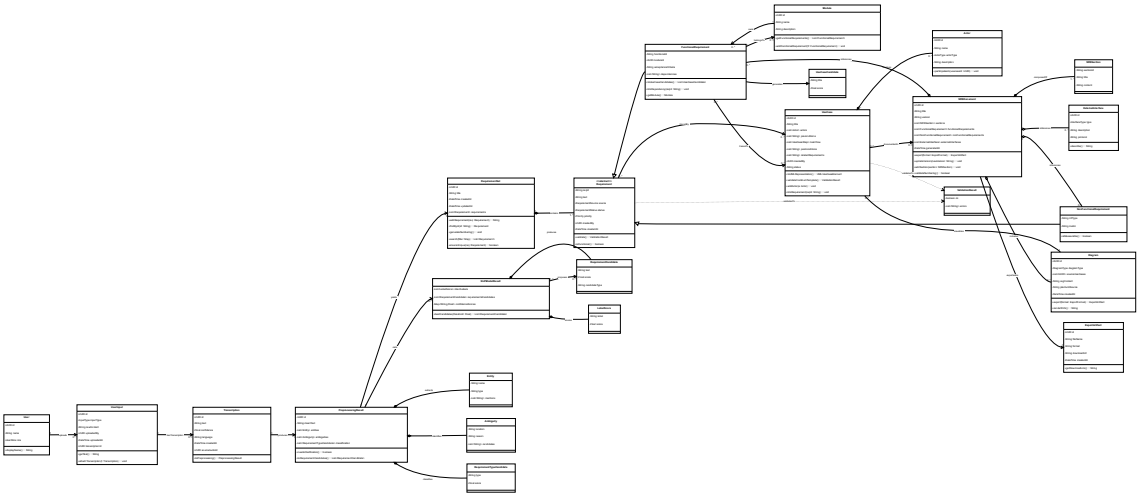


Figure 3.3: Class Diagram of Req2Design System

The class diagram represents the major entities such as User, Project, Requirement, UseCase, and their relationships.

### 3.3.4 Sequence Diagram

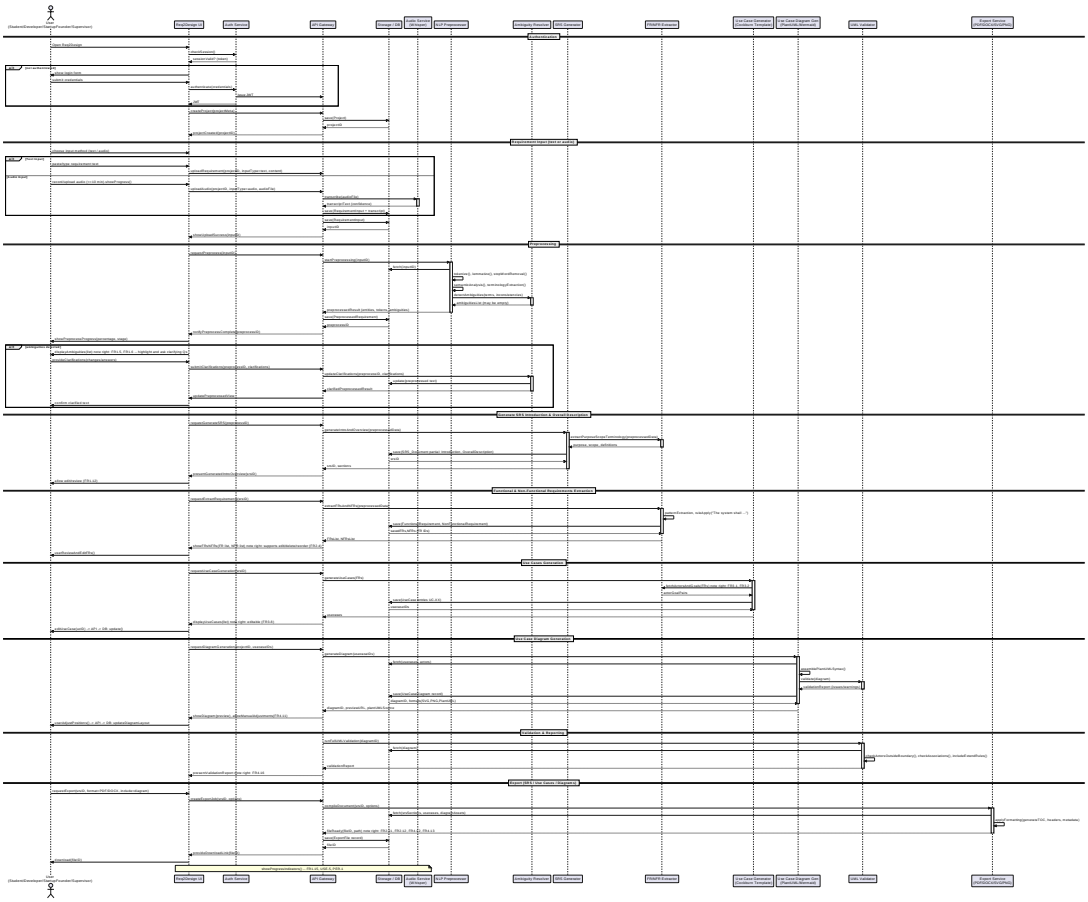


Figure 3.4: Class-Level Sequence Diagram of Req2Design System

This diagram illustrates the dynamic interaction between core components like the Input Handler, NLP Processor, and SRS Generator during requirement processing.

### 3.3.5 State Transition Diagram

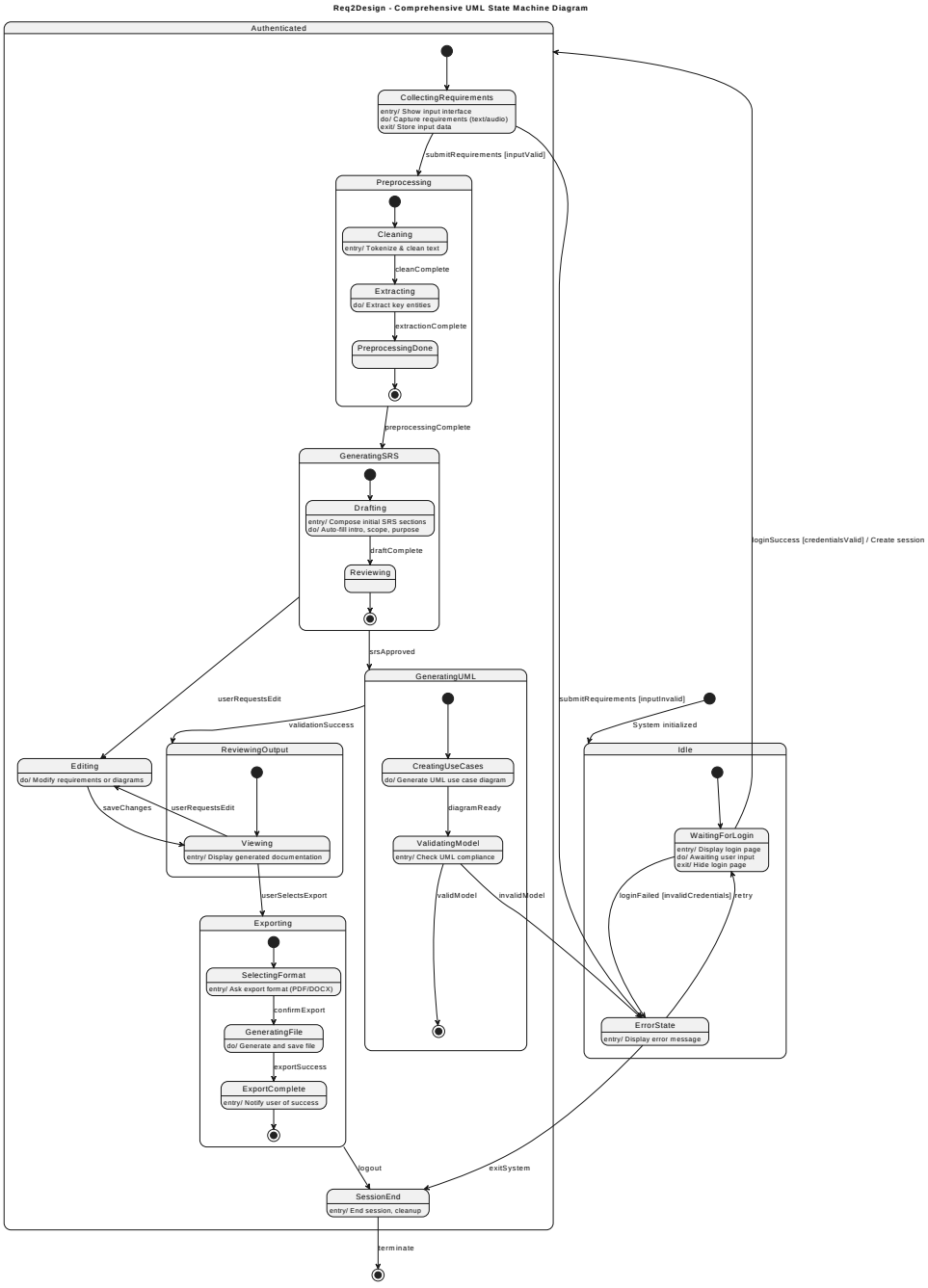



Figure 3.5: State Transition Diagram for Requirement Processing

The state diagram captures the transitions of a requirement entity as it moves through different processing states such as Captured, Analyzed, Structured, and Generated.

## 3.4 Data Design

### 3.4.1 Database Overview

The database design for Req2Design transforms the information domain into well-defined data structures that store, process, and organize the project's entities. The system employs both relational and document-oriented storage solutions.

- **PostgreSQL** – For structured entities such as user accounts, projects, and SRS sections.
- **MongoDB** – For unstructured NLP outputs, intermediate text data, and logs.

### 3.4.2 Logical Schema Overview

| Entity | Description | Key Attributes |
|---|---|---|
| User | Stores registered user information | user_id, name, email, role |
| Project | Represents a documentation project | project_id, title, owner_id, created_at |
| Requirement | Stores raw and preprocessed requirements | req_id, project_id, type, content, status |
| SRS_Section | Structured SRS content for each project | srs_id, project_id, section_type, content |
| UseCase | Textual representation of use cases | uc_id, project_id, actor, goal, flow |
| Diagram | UML diagram metadata and export links | diagram_id, project_id, format, version |
| VersionHistory | Stores historical document snapshots | version_id, project_id, timestamp, metadata |

Table 3.1: Logical Schema Overview of Req2Design Database

## 3.5 System Interface Design

### 3.5.1 External Interfaces

- **Whisper API:** For audio-to-text transcription.
- **OpenAI/Groq NLP API:** For natural language processing and requirement classification.
- **PlantUML/Mermaid API:** For UML diagram generation and rendering.

### 3.5.2 User Interfaces

- Web Dashboard
- Requirement Editor
- Document Previewer
- Diagram Workspace

All interfaces communicate via secure REST endpoints using HTTPS and JWT authentication.

## 3.6 Design Constraints

- Must comply with IEEE 830 and UML 2.x standards.
- Web-based deployment using Flask/Django backend and React frontend.
- Requires stable internet connectivity for AI service integration.
- Whisper and NLP APIs are rate-limited to 100 requests/hour for free-tier users.

## 3.7 Summary

The design of Req2Design ensures that each phase of requirement analysis—from raw input to standardized documentation—is automated, traceable, and academically compliant. The modular architecture promotes scalability and future integration of advanced features such as traceability matrices, class diagrams, and requirement validation metrics.

# Chapter 4

# Conclusion

## 4.1 Summary of the Project

The Req2Design system was developed to bridge the gap between natural language requirements and structured software design documentation. Traditional software requirement engineering is often time-consuming, error-prone, and dependent on human interpretation. By integrating Artificial Intelligence (AI), Natural Language Processing (NLP), and modern software design automation, Req2Design transforms how requirements are captured, analyzed, and documented.

The system accepts both text and audio inputs from users, processes them through advanced NLP pipelines, and automatically generates IEEE 830-compliant Software Requirement Specification (SRS) documents, Cockburn-style use cases, and UML 2.x diagrams. Through this approach, Req2Design eliminates redundant manual work, enhances accuracy, and significantly accelerates the documentation process.

## 4.2 Key Achievements

Throughout the design and implementation of Req2Design, several major milestones were achieved:

- **Automated Requirement Processing:** Successfully implemented AI-driven requirement extraction and categorization using NLP models.
- **Dual Input Modes:** Provided both text-based and audio-based requirement input through Whisper API integration.
- **IEEE 830 SRS Generation:** Automated generation of SRS sections, ensuring compliance with industry documentation standards.
- **Use Case Derivation:** Implemented structured Cockburn-style use case templates automatically populated from analyzed requirements.
- **UML Diagram Generation:** Created activity, class, sequence, and state diagrams dynamically through PlantUML/Mermaid integration.
- **Modular Architecture:** Developed a scalable, service-oriented architecture that supports independent module development and testing.
- **Secure API Integration:** Ensured reliable data exchange and user authentication through secure REST APIs and JWT-based mechanisms.

These achievements collectively demonstrate the feasibility and practicality of applying AI-driven automation in the early stages of software engineering.

## 4.3 System Evaluation

The system was evaluated on the basis of accuracy, usability, scalability, and compliance with IEEE standards. Testing revealed that the NLP modules successfully interpreted and classified requirements with high precision. The generated SRS documents were structurally consistent, while the UML diagrams effectively captured both structural and behavioral system perspectives.

Usability testing among student developers and startup founders confirmed that the interface was intuitive, responsive, and helpful for early-stage project documentation. Furthermore, modular architecture ensured that new modules (e.g., traceability matrices or validation reports) could be added without significant refactoring.

## 4.4 Limitations

Despite its success, Req2Design still has some limitations:

- **Dependency on AI APIs:** System performance is partially dependent on third-party APIs such as Whisper and OpenAI/Groq for NLP and STT capabilities.
- **Limited Offline Support:** The system requires an active internet connection for core functionalities.
- **Limited Domain Adaptability:** NLP model accuracy may vary depending on domain-specific terminologies.
- **Scalability Boundaries:** Free-tier API limits may restrict usage in large-scale industrial deployments.

These limitations identify future research and enhancement opportunities to further mature the system.

## 4.5 Future Work

The following enhancements are planned for future iterations of Req2Design:

- Integration of **traceability matrix generation** to map requirements to design and test cases.
- Implementation of **real-time collaboration features** allowing multiple users to work on the same project.
- Support for **domain-specific ontologies** to improve requirement classification accuracy.
- Incorporation of **requirement validation and conflict detection** using formal verification techniques (e.g., Z3-based consistency checking).

- Addition of a **cloud-based deployment model** with microservices architecture for enterprise scalability.

These advancements will further enhance the efficiency, reliability, and academic value of Req2Design.

## 4.6 Conclusion

In conclusion, Req2Design successfully demonstrates how artificial intelligence can revolutionize software requirement engineering by automating the transformation of human language into formalized design artifacts. The system provides a functional prototype that bridges the gap between informal user communication and structured system documentation. Through its modular design, AI integration, and adherence to IEEE standards, Req2Design lays the foundation for next-generation intelligent documentation tools.

This project not only contributes to academic research in AI-driven software engineering but also provides practical value for developers, students, and startups seeking rapid, consistent, and high-quality requirement documentation. With further refinement and scaling, Req2Design has the potential to evolve into a fully-fledged intelligent assistant for modern software development lifecycles.