Project Proposal

AI-Driven Automated Requirements Engineering System

Members- Devanshi Jain,Sreyas Sharma,Shivang Mangal,Shreyash Shivhare

Mentor- Mitali Chugh Ma'am

1. Introduction

Requirements Engineering (RE) is a critical phase in the Software Development Life Cycle (SDLC), involving the identification, analysis, documentation, and prioritization of stakeholder requirements. In large-scale and agile-driven projects, stakeholders communicate requirements through emails, chat platforms, documents, and meetings, resulting in unstructured, scattered, and continuously evolving data.

Manual extraction and management of requirements from such sources is time-consuming, error-prone, and does not scale efficiently.

This project proposes an AI-driven, desktop-based automated requirements engineering system that extracts, classifies, clusters, prioritizes, and summarizes requirements from multiple stakeholder communication sources using Natural Language Processing (NLP) and Machine Learning (ML) techniques.

2. Objectives of the Proposed System

The primary objectives of the system are:

- Automatically extract software requirements from unstructured text sources
- Classify requirements into functional and non-functional categories
- Cluster similar or related requirements
- Prioritize requirements based on importance indicators
- Generate concise summaries for stakeholder understanding
- Reduce manual effort and improve accuracy in the RE phase

3. Development Methodology

3.1 Agile Scrum Approach

- Development is divided into multiple time-boxed sprints
- Each sprint delivers a working increment (e.g., extraction, clustering, prioritization)
- Continuous stakeholder feedback is incorporated
- Agile is chosen due to:

- Evolving AI models
- Experiment-driven improvements
- Changing stakeholder requirements

3.2 Kanban for Task Management

Kanban principles are used alongside Scrum to:

- Track tasks visually
- Manage AI experimentation workflows
- Improve development transparency

4. System Architecture Overview

Architectural Layers:

1. Desktop Application Layer (Frontend)
2. Backend Orchestration Layer
3. AI / NLP Processing Layer
4. Data Handling and Integration Layer

5. Frontend Design and Methodology

(Electron + React + Vite)

5.1 Electron-Based Desktop Application

- The system is implemented as a cross-platform desktop application using Electron.
- Enables deployment on Windows, Linux, and macOS
- Provides:
  - Native-like UI
  - Local processing capabilities
  - Secure communication with backend services

5.2 User Interface with React and Vite

- React.js is used for its:
  - Component-based architecture
  - Efficient state management

    - Reusability
- Vite is used as the build tool to ensure:
    - Faster development builds
    - Optimized production bundling

5.3 Frontend Capabilities

The UI allows users to:

- Upload documents (PDFs, text files)
- Connect communication platforms (emails, chats)
- Trigger AI analysis
- View:
    - Extracted requirements
    - Classified categories
    - Clustered groups
    - Priority levels
    - Summarized insights

6. Backend Methodology

(Spring Boot – Orchestration Layer)

Responsibilities of the Backend:

- Handle API requests from the Electron frontend
- Manage authentication and authorization
- Fetch data from external communication platforms
- Validate and preprocess textual data
- Communicate with the AI/NLP microservice
- Send structured responses to the frontend

7. Data Collection and Integration Layer

7.1 Email Integration

- Emails are fetched using:
    - IMAP protocol

    - Gmail API

- Features:
    - Secure authentication
    - Extraction of email body text
    - Processing of document attachments (PDF, DOCX)

7.2 Text Messages Integration

- Messages are accessed via:
    - Supported messaging APIs
    - Exported logs (where APIs are unavailable)
- Metadata such as timestamps and sender information are preserved
- Privacy-sensitive fields are handled securely

7.3 Telegram Integration

- Implemented using:
    - Telegram Bot API
    - Telegram client libraries
- Authorized access enables:
    - Message retrieval from chats or groups
    - Text normalization and aggregation

7.4 Unified Text Format

All collected data is:

- Cleaned
- Normalized
- Converted into a common text representation
  before being sent to the AI layer.

8. AI and NLP Methodology

(Flask-Based Python Microservice)

8.1 Flask NLP Microservice

- All AI and NLP logic is implemented as a Python microservice
- Flask is used due to:

- Lightweight design
- Easy REST API exposure
- Seamless ML integration

8.2 NLP Techniques Employed

a) Text Preprocessing

- Noise removal
- Sentence segmentation
- Tokenization
- Text normalization

b) Requirement Extraction

- Rule-based linguistic patterns (e.g., *shall, must, should*)
- Named Entity Recognition (NER) to identify:
  - Actors
  - Actions
  - System components

c) Requirement Classification

- Functional vs Non-Functional classification
- Machine learning classifiers using NLP features

d) Requirement Clustering

- TF-IDF vectorization
- K-Means clustering
- Topic modeling as future enhancement

e) Requirement Prioritization

- Keyword-based importance
- Frequency-based scoring
- Requirement type weighting

f) Text Summarization

- Extractive summarization
- Generation of concise stakeholder-friendly summaries

9. System Workflow

1. User interacts with the Electron–React UI
2. Backend fetches and preprocesses data
3. Cleaned text is sent to Flask AI service
4. AI service performs:
    - Extraction
    - Classification
    - Clustering
    - Prioritization
    - Summarization
5. Results are returned to frontend for visualization

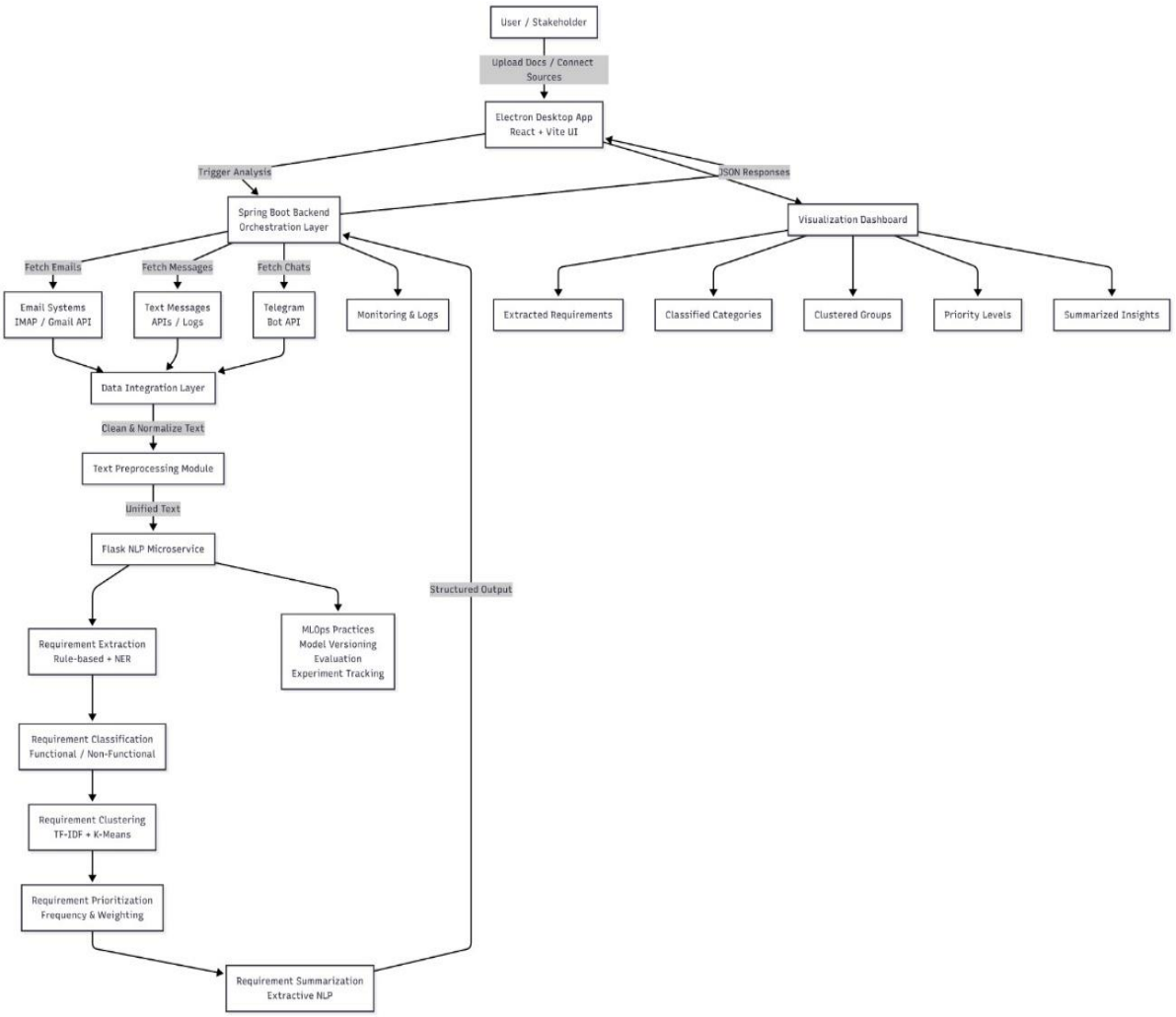
10. DevOps and AI Lifecycle Practices

The project adopts MLOps-inspired practices, including:

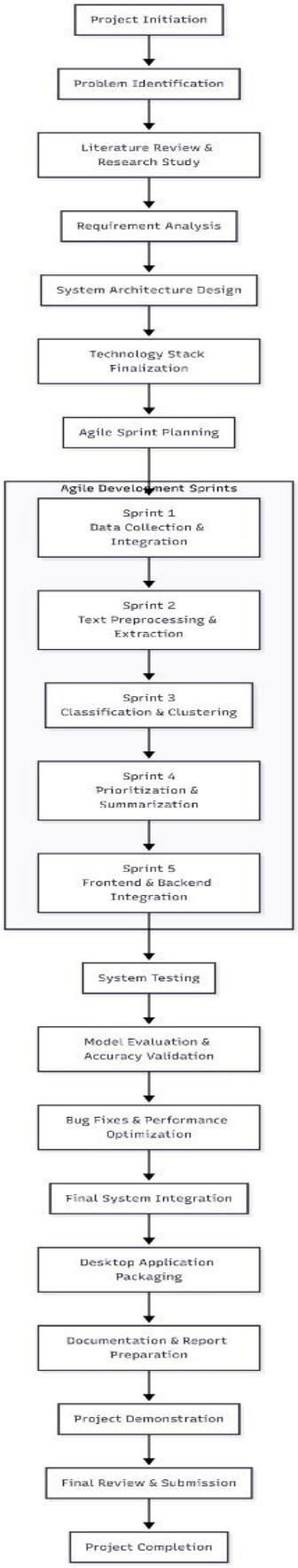
- Modular AI pipelines
- Dataset and model versioning
- Reproducible experiments
- Evaluation-driven improvements

This ensures long-term scalability and maintainability.

11.Flow Diagram



12.Development Cycle



13.Future Scope

The system can be extended to include:

- Real-time message ingestion
- Ambiguity detection in requirements
- Conflict and dependency analysis
- Stakeholder-aware prioritization
- Requirement Traceability Matrix (RTM)
- Transformer-based NLP models (BERT, etc.)

14.Conclusion

The proposed system delivers a practical, scalable, and industry-aligned solution to automate the Requirements Engineering phase using AI. By combining:

- Electron-based desktop UI
- Spring Boot orchestration backend
- Flask-based NLP microservices
- Agile Scrum development methodology

the solution effectively addresses the challenges of modern, communication-heavy software projects.