Software Requirements Specification (SRS)

Project Title: Traffic Sign Recognition Demo
Version: 1.0
Date: 25-08-2025
Status: Final Submission

Authors & Contributors

Role in Scrum Team	Name	SRN	Agile Contribution Area
Product Owner / Mentor	Dr. Swetha P	–	Acts as Product Owner guiding vision, clarifying requirements, and ensuring alignment with learning objectives.
Scrum Master	Adishree Gupta	PES1UG23CS024	Facilitates sprints, removes blockers, ensures Agile ceremonies (planning, review, retrospective), manages documentation.
Developer	Akshat	PES1UG23CS048	Focuses on model training, backend APIs, and sprint-wise integration of ML modules.
Developer	Monica M	PES1UG24CS813	Designs and iterates UI/UX, integrates recognition results into user stories, and provides visual feedback to the team.
Developer	Aditya Sharma	PES1UG23CS035	Handles dataset curation, preprocessing tasks, and sprint-based testing of ML pipelines.

Revision History

Version	Date	Author	Change Summary	Approval
1.0	25-08-2025	Dr. Swetha P	SRS with diagrams embedded	–

Approvals

Role	Name	Signature / Email	Date
Course Coordinator	Prof .Rajesh Banginwar		**25/08/2025**

1. Introduction

1.1 Purpose

This document specifies the requirements for the **Traffic Sign Recognition System (TSRS)**, developed as part of the Software Engineering course.

Our project demonstrates how **Software Engineering principles** — including requirements engineering, SDLC models, Agile methodology, testing, and security — can be applied to a **real-world challenge**:

According to the World Health Organization, over 1.3 million people die each year in road accidents. A significant portion of these accidents are linked to **ignored or misunderstood traffic signs**.

TSRS uses AI to recognize traffic signs from uploaded images, illustrating how disciplined software engineering can contribute to **road safety, driver training, and smart city transport systems**.

1.2 Scope

- **User:** Uploads an image (PNG/JPG).
- **System:** Preprocesses the image, classifies it with a trained ML model, and displays recognition results with confidence scores.
- **Admin:** Monitors logs, checks performance, and retrains the ML model.

Applications in practice:

- Driving school tools for learner awareness.
- Input into **autonomous vehicle systems**.
- Smart traffic enforcement in cities.

1.3 Audience

- **Faculty & Evaluators:** Assess application of SE concepts.
- **Developers/Researchers:** Extend the project to autonomous driving domains.
- **Students:** Understand integration of SE and AI.

1.4 Motivation & Vision

Just as **CrowdStrike applied structured SE to cybersecurity**, our project applies **Agile + SSDDLC practices** to road safety.

Vision:

- Deliver a working AI demo that proves ML can recognize road signs.
- Use **Agile methodology** to handle iterative model refinements and UI updates.
- Ensure **security is embedded** in every sprint via SSDDLC.
- Inspire peers to view SE as a **practical, evolving framework** for solving problems.

2. Overall Description

2.1 Product Perspective

TSRS is a **web-based system** comprising:

- **Frontend UI** → Image upload, result display.
- **Backend ML Engine** → Classification service.
- **Database** → Recognition logs and reports.

It reflects **industry projects in autonomous driving and intelligent transport systems**.

2.2 Major Product Functions

- Authenticate users.
- Upload traffic sign images.
- Preprocess (resize, normalize).
- Classify into ≥50 categories.
- Display results + confidence score.
- Generate downloadable reports.
- Admin dashboard for monitoring & retraining.

2.3 User Roles

- **User:** Upload → Recognize → Download report.
- **Admin:** Monitor logs → Retrain ML model → Track usage.

2.4 Operating Environment

- **Client:** Chrome/Edge browser.
- **Backend:** Python (Flask/FastAPI).
- **ML:** TensorFlow/PyTorch.
- **Database:** MySQL.

2.5 Constraints

- Only open-source licensed datasets allowed.
- Max upload size: 5MB.
- Accuracy target ≥ 90%.
- All communication via HTTPS.

3. Software Engineering Foundations

3.1 Why Software Engineering?

SE ensures clarity, discipline, and reliability. It prevents projects from being ad hoc by enforcing:

- Traceable requirements.
- Systematic design + testing.
- Team collaboration.

3.2 Comparing SDLC Models

Model	Pros	Cons
Waterfall	Linear, simple, strong documentation	Inflexible, late testing
Spiral	Risk-driven, iterative	Expensive, complex for small teams
V-Model	Testing embedded at each stage	Costly, rigid
Agile	Flexible, iterative, user-focused	Needs discipline & collaboration

3.3 Why Agile and SSDDLC Together

Our project requires **continuous adaptation**: ML models must be retrained, datasets refined, and UI adjusted frequently. Agile suits this environment because it supports:

- **Incremental delivery:** Each sprint adds working functionality.
- **Continuous feedback:** Recognition accuracy is checked early.
- **Team collaboration:** Daily scrums align efforts.

But adaptability is not enough. Road safety systems must also be **secure**. That is why we combined Agile with the **Secure Software Development Life Cycle (SSDDLC)**:

- **Agile** → provides speed and flexibility.
- **SSDDLC** → ensures **security at every phase** (requirements → design → implementation → testing → deployment).

Benefits of Agile + SSDDLC:

- Early threat modeling for uploads.
- Shift-left security → validation during development.
- Security checklists in sprint reviews.
- Secure coding & encrypted storage.

Thus, Agile gave us **adaptability**, while SSDDLC gave us **assurance**.

4. Agile Development Approach

4.1 Scrum Process

- **Backlog:** FRs, NFRs, Security.
- **Sprint Planning:** Feature selection.
- **Sprints (2 weeks):**
    - Sprint 1: Authentication + Upload.

    - Sprint 2: Preprocessing + Validation.
    - Sprint 3: ML Model + Recognition.
    - Sprint 4: Admin Dashboard + Reports + Testing.
- **Daily Scrum:** 10-minute syncs.
- **Review:** Demo to peers/faculty.
- **Retrospective:** Team improvement.

4.2 Security in Agile

- Sprint 1: HTTPS, login security.
- Sprint 2: File-type validation.
- Sprint 3: RBAC for User/Admin.
- Sprint 4: Log encryption, session timeouts.

5. Requirements Engineering

5.1 Elicitation & Analysis

Sources: brainstorming, labs, case studies, personas.

5.2 Functional Requirements (15 FRs)

IDs TSRS-F-001 to TSRS-F-015 → authentication, upload, preprocessing, classification, reporting, admin.

5.3 Non-Functional Requirements (5 NFRs)

Performance ≤ 3s, Accuracy ≥ 90%, Availability ≥ 99%, Secure uploads, Responsive UI.

5.4 Security Objectives

- Protect files.
- Enforce encryption.
- Prevent malicious uploads.

5.5 Security Requirements (5)

HTTPS-only, file validation, RBAC, encrypted DB, session timeout.

5.6 Personas & Scenarios

- *Riya (18, driving learner)* → uploads to learn traffic signs.
- *Arun (researcher)* → tests datasets.

6. System Features

Example:

- **TSRS-F-006:** System shall classify uploaded signs into ≥50 categories.
- **Acceptance:** ≥90% accuracy on benchmark dataset.

7. Non-Functional Requirements

- **Performance:** ≤3s per recognition.
- **Reliability:** ≥99% uptime.
- **Usability:** WCAG-compliant UI.
- **Security:** PCI-DSS style compliance.

8. Security

- Lifecycle integrated into sprints.
- Regular threat modeling.
- Early “shift-left” validation.
- SSDDLC applied across phases.

9. Quality & Testing

9.1 Testing Types

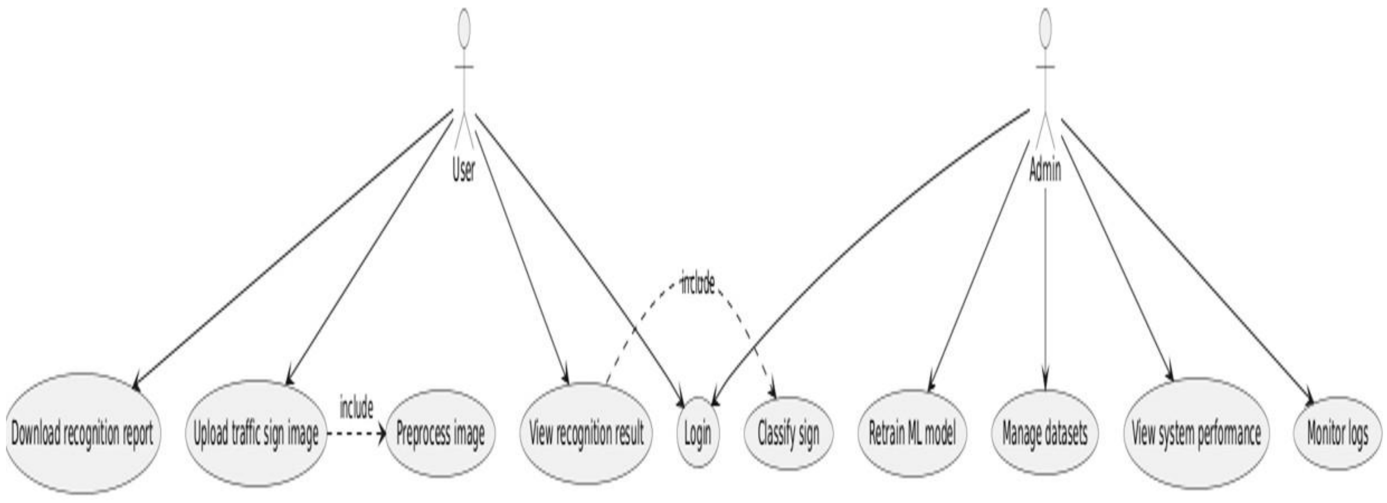
- Unit: preprocessing.
- Integration: ML + API.
- System: upload → classify → display.
- Security: malicious files, HTTPS validation.

9.2 Security Validation Plan

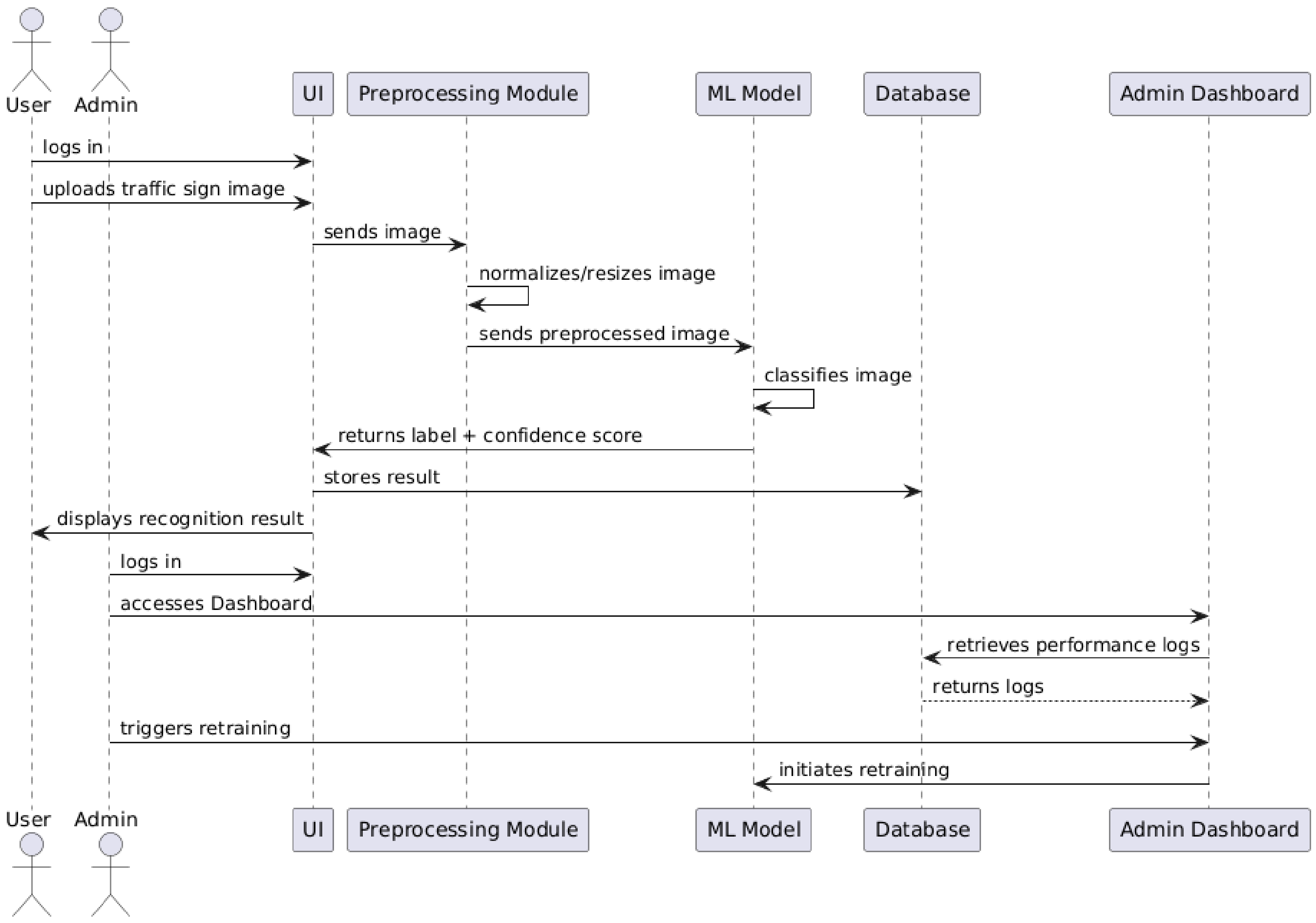
- Test HTTPS.
- Test invalid uploads.
- Test session timeout.

10. UML Diagrams

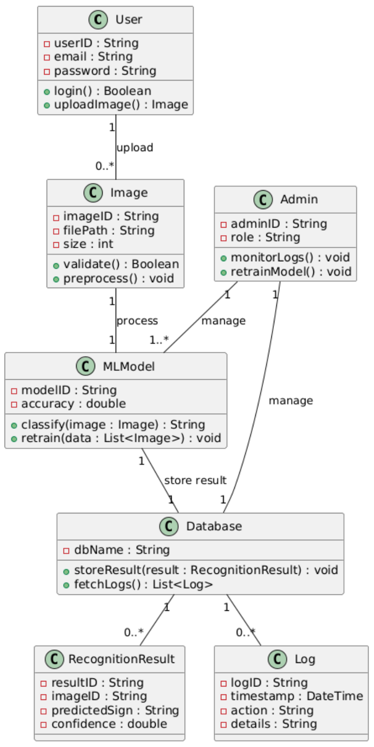
- **Use Case Diagram** → User/Admin interactions.



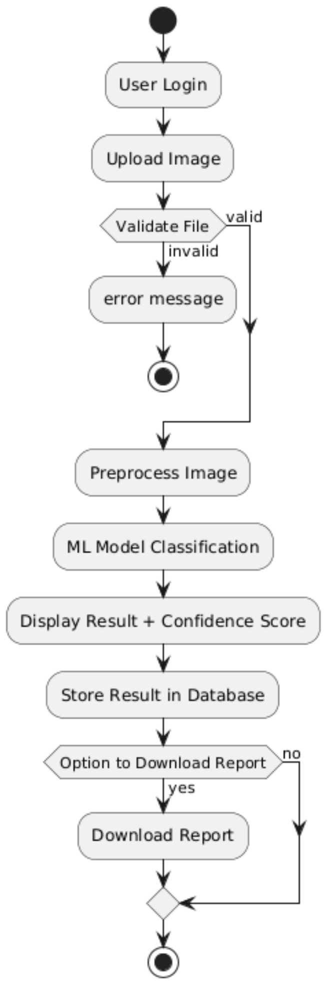
- **Sequence Diagram** → Upload → Preprocess → Classify → Result.



- **Class Diagram** → User, Admin, Image, MLModel, Database.



- **Activity Diagram** → End-to-end workflow.



11. Requirements Traceability Matrix (RTM)

Req ID	Requirement	Sprint	Module	Test Case	Status
TSRS-F-001	User login	1	Auth Module	TC-Auth-01	N
TSRS-F-003	Image upload	1	Upload Module	TC-Upload-01	N
TSRS-F-006	Sign classification	3	ML Inference	TC-Rec-01	N
TSRS-NF-001	Response $\leq$ 3s	3	Backend	TC-Perf-01	N
TSRS-SR-002	File validation	2	Upload Module	TC-Sec-01	N

12. Reflection & Conclusion

Our journey with **TSRS** highlighted how **Agile and SSDLC** can work hand in hand:

- **Labs & Case Studies** (like CrowdStrike) showed us how SE powers real industries.
- **SDLC exploration** revealed Agile as the natural choice for iterative ML.
- **Scrum practices** made our teamwork adaptive and effective.
- **Requirements Engineering & Testing** ensured clarity and completeness.
- **Security integration** taught us that robust software = safe software.

We chose Agile model as it matched our project's needs. And by embedding SSDLC, we proved that security and agility can coexist. TSRS shows how structured, iterative engineering can turn classroom learning into **meaningful real-world impact** for road safety.