Software Requirements Specification (SRS)

AI-Powered Pothole Detection & Reporting System

Version: 1.0
Date: November 18, 2025
Project Status: Production Deployed

Table of Contents

---

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document provides a comprehensive description of the AI-Powered Pothole Detection & Reporting System. It details the functional and non-functional requirements, system architecture, and technical specifications for developers, stakeholders, and maintenance teams.

1.2 Scope

The system is a cloud-based web application that enables citizens to:

- Upload road images for instant AI-powered pothole detection
- Receive confidence-scored analysis results (0-100%)
- Submit geotagged complaints with image evidence to authorities
- Access the platform via secure authentication (JWT, OAuth 2.0)

Key Benefits:

- **For Citizens:** Simplified infrastructure issue reporting, real-time feedback
- **For Municipalities:** Centralized complaint management, visual evidence, location data
- **For Road Safety:** Proactive maintenance, reduced vehicle damage, improved public infrastructure

1.3 Definitions, Acronyms, and Abbreviations

Term	Definition
AI	Artificial Intelligence

Term	Definition
API	Application Programming Interface
CNN	Convolutional Neural Network
CORS	Cross-Origin Resource Sharing
JWT	JSON Web Token
OAuth 2.0	Open Authorization 2.0 standard
OTP	One-Time Password
REST	Representational State Transfer
SPA	Single Page Application
SSL/TLS	Secure Sockets Layer/Transport Layer Security
ML	Machine Learning

1.4 References

- MongoDB Atlas Documentation: https://docs.atlas.mongodb.com/
- React.js Official Documentation: https://react.dev/
- Node.js API Documentation: https://nodejs.org/docs/
- PyTorch Documentation: https://pytorch.org/docs/
- OAuth 2.0 Specification: RFC 6749

1.5 Document Overview

This SRS is organized into seven sections covering system introduction, description, functional/non-functional requirements, interface specifications, architecture, and appendices.

---

2. Overall Description

2.1 Product Perspective

The system operates as a **three-tier cloud-native application**:

1. **Presentation Layer:** React-based responsive web interface (Vercel)
2. **Application Layer:** Node.js/Express REST API server (Render)
3. **AI Inference Layer:** Python/Flask ML microservice (Heroku)
4. **Data Layer:** MongoDB Atlas cloud database

The system integrates with external services:

- Google OAuth 2.0 (authentication)
- Microsoft OAuth 2.0 (authentication)
- Brevo API (transactional email delivery)

2.2 Product Functions

Primary Functions:

1. **User Authentication & Authorization**
    - Email/password registration with OTP verification
    - Google OAuth 2.0 social login
    - Microsoft OAuth 2.0 social login
    - JWT-based session management
2. **AI-Powered Image Analysis**
    - Real-time pothole detection from uploaded images
    - Confidence score calculation (0-100%)
    - Processing time measurement
    - Repair recommendation generation
3. **Complaint Management**
    - Structured complaint submission (location, description, image)
    - Base64 image storage with compression
    - Confidence score archival
    - Complaint retrieval and listing
4. **User Interface**
    - Mobile-responsive design (320px - 4K displays)
    - Real-time feedback and loading states
    - Accessibility compliance (WCAG 2.1 Level AA)
    - Glass-morphism UI with modern aesthetics

2.3 User Classes and Characteristics

User Class	Description	Technical Expertise	Frequency of Use
General Public	Citizens reporting potholes	Low	Occasional
Municipal Staff	Government employees viewing complaints	Medium	Daily
System Administrators	Technical team managing infrastructure	High	As needed
Developers	Software engineers maintaining codebase	Expert	Daily

2.4 Operating Environment

Client-Side Requirements:

- Modern web browser (Chrome 90+, Firefox 88+, Safari 14+, Edge 90+)
- JavaScript enabled

- Internet connection (min 2 Mbps)
- Camera access (for mobile image capture)

Server-Side Environment:

- **Frontend:** Vercel Edge Network (global CDN)
- **Backend:** Render cloud platform (Node.js 18+)
- **AI Service:** Heroku dyno (Python 3.10+)
- **Database:** MongoDB Atlas M0 cluster (shared tier)

Third-Party Services:

- Google OAuth 2.0 API
- Microsoft Identity Platform
- Brevo Email API

2.5 Design and Implementation Constraints

Technical Constraints:

- Maximum request payload: 10 MB (base64 image limit)
- API rate limiting: 100 requests/15 minutes per IP
- Authentication rate limiting: 5 attempts/15 minutes per IP
- MongoDB document size limit: 16 MB
- JWT token expiration: 24 hours
- Image format support: JPEG, PNG, WebP

Regulatory Constraints:

- GDPR compliance for EU users
- Data encryption in transit (TLS 1.3)
- Password hashing (bcrypt, 12 rounds)
- No PII storage without consent

Business Constraints:

- Free-tier deployment limitations (cold starts on Render)
- Email sending limits: 300 emails/day (Brevo free tier)
- MongoDB storage limit: 512 MB (Atlas free tier)

2.6 Assumptions and Dependencies

Assumptions:

- Users have access to a device with camera or file upload capability
- Internet connectivity is available during usage
- Users provide valid email addresses for OTP verification
- Uploaded images clearly show road conditions

Dependencies:

- MongoDB Atlas uptime (99.95% SLA)

- Vercel CDN availability
- Render platform reliability
- Heroku dyno uptime
- Google/Microsoft OAuth service availability
- Brevo email delivery infrastructure

---

3. System Features and Requirements

3.1 User Authentication

Priority: High
Description: Secure user registration, login, and session management.

3.1.1 Functional Requirements

FR-AUTH-001: The system shall allow users to register with email and password.

- **Input:** Email (valid format), password (min 8 characters)
- **Process:** Validate input, hash password (bcrypt), generate OTP, send verification email
- **Output:** User account created, OTP sent to email
- **Validation:** Email uniqueness check, password strength validation

FR-AUTH-002: The system shall send OTP verification emails via Brevo API.

- **Input:** User email, 6-digit OTP
- **Process:** HTTP POST to Brevo API with SMTP fallback
- **Output:** Email delivered within 30 seconds
- **Validation:** API key validation, rate limit enforcement

FR-AUTH-003: The system shall verify OTP within 10 minutes of generation.

- **Input:** User email, 6-digit OTP
- **Process:** Compare OTP hash, check expiration timestamp
- **Output:** Account activated or error message
- **Validation:** Maximum 3 attempts per OTP

FR-AUTH-004: The system shall support Google OAuth 2.0 login.

- **Input:** Google authorization code
- **Process:** Exchange code for user profile, create/link account
- **Output:** JWT token, redirect to dashboard
- **Validation:** Verify Google OAuth token signature

FR-AUTH-005: The system shall support Microsoft OAuth 2.0 login.

- **Input:** Microsoft authorization code
- **Process:** Exchange code for user profile, create/link account
- **Output:** JWT token, redirect to dashboard
- **Validation:** Verify Microsoft token signature

FR-AUTH-006: The system shall generate JWT tokens with 24-hour expiration.

- **Input:** User ID
- **Process:** Sign payload with HS256 algorithm
- **Output:** JWT token string
- **Validation:** Secret key validation, expiration check

FR-AUTH-007: The system shall validate JWT tokens on protected routes.

- **Input:** Authorization header with Bearer token
- **Process:** Verify signature, check expiration
- **Output:** User context or 401 Unauthorized
- **Validation:** Token structure and signature validation

3.2 Image Upload and AI Detection

Priority: High
Description: Core functionality for pothole detection using deep learning.

3.2.1 Functional Requirements

FR-DETECT-001: The system shall accept image uploads (JPEG, PNG, WebP).

- **Input:** Image file (max 10 MB)
- **Process:** Validate format and size, create FormData
- **Output:** Image preview with base64 encoding
- **Validation:** File type check, size limit enforcement

FR-DETECT-002: The system shall send images to Flask AI service for analysis.

- **Input:** Image file as multipart/form-data
- **Process:** HTTP POST to `/predict` endpoint
- **Output:** JSON response with detection results
- **Validation:** Service availability check, timeout (60 seconds)

FR-DETECT-003: The AI service shall detect potholes using CNN model.

- **Input:** Image tensor (normalized)
- **Process:** Forward pass through PyTorch model
- **Output:** Binary classification (pothole/no pothole)
- **Validation:** Model checkpoint integrity verification

FR-DETECT-004: The system shall calculate confidence scores (0-100%).

- **Input:** Model softmax output
- **Process:** Multiply by 100, round to 2 decimal places
- **Output:** Confidence percentage
- **Validation:** Value range check (0-100)

FR-DETECT-005: The system shall measure prediction time.

- **Input:** Start timestamp, end timestamp
- **Process:** Calculate difference in milliseconds
- **Output:** Processing time in seconds (2 decimal places)
- **Validation:** Reasonable time range (0.5-10 seconds)

FR-DETECT-006: The system shall display detection results in real-time.

- **Input:** API response JSON
- **Process:** Update UI with prediction, confidence, time
- **Output:** Visual feedback with color coding
- **Validation:** Data format validation

FR-DETECT-007: The system shall provide repair recommendations.

- **Input:** Detection result (pothole/no pothole)
- **Process:** Map result to recommendation string
- **Output:** "Immediate Repair Needed" or "No Immediate Action Needed"
- **Validation:** Binary decision logic

3.3 Complaint Submission

Priority: High
Description: Enable users to submit complaints with evidence.

3.3.1 Functional Requirements

FR-COMPLAINT-001: The system shall accept complaint submissions with metadata.

- **Input:** Location (string), description (string), image (base64), confidence (float)
- **Process:** Validate input fields, extract user context
- **Output:** Complaint document in MongoDB
- **Validation:** Required field checks, max length limits

FR-COMPLAINT-002: The system shall store base64-encoded images.

- **Input:** Base64 string from preview
- **Process:** Validate encoding, calculate size
- **Output:** Image data in complaint document
- **Validation:** Size limit (16 MB MongoDB limit)

FR-COMPLAINT-003: The system shall store AI confidence scores.

- **Input:** Raw confidence value (0-1)
- **Process:** Store as Number type
- **Output:** Confidence field in database
- **Validation:** Range check (0-1)

FR-COMPLAINT-004: The system shall associate complaints with user accounts.

- **Input:** JWT token from Authorization header
- **Process:** Extract user ID, link to complaint

- **Output:** User reference in complaint document
- **Validation:** Token validity and user existence

FR-COMPLAINT-005: The system shall generate timestamps for complaints.

- **Input:** Submission moment
- **Process:** Create ISO 8601 timestamp
- **Output:** createdAt field with timezone
- **Validation:** Valid date format

FR-COMPLAINT-006: The system shall retrieve complaints by ID.

- **Input:** Complaint MongoDB ObjectId
- **Process:** Query database with ID
- **Output:** Full complaint object with image
- **Validation:** Valid ObjectId format, existence check

FR-COMPLAINT-007: The system shall list all complaints for authenticated users.

- **Input:** User JWT token
- **Process:** Query complaints by user ID
- **Output:** Array of complaint objects
- **Validation:** Authorization check

3.4 User Interface

Priority: Medium
Description: Responsive, accessible web interface.

3.4.1 Functional Requirements

FR-UI-001: The system shall display a responsive navigation bar.

- **Components:** Logo, app name, navigation links, logout button
- **Responsive breakpoints:** 320px, 768px, 1024px, 1440px
- **Validation:** Active link highlighting

FR-UI-002: The system shall provide a homepage with feature overview.

- **Sections:** Hero banner, features grid, workflow diagram, statistics
- **Animations:** Fade-in, slide-up, hover effects
- **Validation:** Content loading states

FR-UI-003: The system shall display login/signup forms with validation.

- **Fields:** Email (email format), password (min 8 chars)
- **Validation:** Real-time inline validation
- **Feedback:** Success/error messages (3-second timeout)

FR-UI-004: The system shall show OTP verification page.

- **Components:** 6-digit input fields, resend button, countdown timer

- **Behavior:** Auto-focus next field, submit on 6 digits
- **Validation:** Numeric input only

FR-UI-005: The system shall render pothole detection interface.

- **Components:** File upload, preview, detection results, complaint form
- **States:** Idle, uploading, processing, results, error
- **Validation:** Loading indicators, error boundaries

FR-UI-006: The system shall display complaint submission form.

- **Fields:** Location (text), description (textarea)
- **Behavior:** Disabled when no detection, enabled after detection
- **Validation:** Required field enforcement

FR-UI-007: The system shall show about/contact pages.

- **Content:** Team information, mission statement, contact form
- **Validation:** Contact form email validation

FR-UI-008: The system shall use consistent theming.

- **Colors:** CSS variables (--color-bg, --color-accent, etc.)
- **Typography:** System font stack with fallbacks
- **Validation:** WCAG AA contrast ratios

3.5 Security Features

Priority: High
Description: Protection against common web vulnerabilities.

3.5.1 Functional Requirements

FR-SEC-001: The system shall implement rate limiting on authentication endpoints.

- **Limits:** 5 requests per 15 minutes per IP
- **Action:** Return 429 Too Many Requests
- **Headers:** RateLimit-* headers with remaining count

FR-SEC-002: The system shall implement rate limiting on API endpoints.

- **Limits:** 100 requests per 15 minutes per IP
- **Action:** Return 429 Too Many Requests
- **Headers:** RateLimit-Limit, RateLimit-Remaining

FR-SEC-003: The system shall hash passwords with bcrypt.

- **Algorithm:** bcrypt with 12 salt rounds
- **Process:** Hash on registration, compare on login
- **Validation:** Minimum password complexity

FR-SEC-004: The system shall sanitize NoSQL queries.

- **Method:** express-mongo-sanitize middleware
- **Process:** Remove `$` and `.` from input
- **Validation:** Replace prohibited characters with `_`

FR-SEC-005: The system shall set security HTTP headers.

- **Headers:** Helmet.js default set
- **CSP:** Content-Security-Policy with restricted sources
- **HSTS:** Strict-Transport-Security with 1-year max-age

FR-SEC-006: The system shall enforce CORS policies.

- **Origin:** Vercel production domain only
- **Credentials:** Allow cookies and auth headers
- **Methods:** GET, POST, PUT, DELETE, OPTIONS

FR-SEC-007: The system shall validate JWT signatures.

- **Algorithm:** HS256 with 256-bit secret
- **Validation:** Signature, expiration, issuer
- **Action:** Reject invalid tokens with 401

FR-SEC-008: The system shall use HTTPS for all communications.

- **TLS Version:** 1.3 (minimum 1.2)
- **Certificates:** Automatic via Vercel/Render/Heroku
- **Validation:** Redirect HTTP to HTTPS

---

4. External Interface Requirements

4.1 User Interfaces

4.1.1 Homepage (`/`)

- **Layout:** Fixed navbar, hero section, features grid, workflow, footer
- **Responsive:** Mobile (320px), Tablet (768px), Desktop (1024px+)
- **Interactions:** Scroll animations, hover effects, CTA buttons

4.1.2 Login Page (`/login`)

- **Components:** Email input, password input, submit button, OAuth buttons
- **Validation:** Real-time inline error messages
- **States:** Idle, loading, success, error

4.1.3 Signup Page (`/signup`)

- **Components:** Email input, password input, confirm password, submit button
- **Validation:** Password match, email format, password strength
- **States:** Idle, loading, OTP sent

4.1.4 OTP Verification (`/verify-otp`)

- **Components:** 6 OTP input boxes, resend button, countdown timer
- **Behavior:** Auto-advance on digit entry, auto-submit on completion
- **States:** Active, verifying, success, expired

4.1.5 Pothole Detection (`/pothole`)

- **Components:** File upload area, image preview, detection results, complaint form
- **Workflow:** Upload → Detect → Review → Submit Complaint
- **States:** Empty, uploaded, detecting, results, submitting

4.1.6 OAuth Callback (`/auth/callback`)

- **Components:** Loading spinner, status messages
- **Process:** Extract token from URL, store in localStorage, redirect
- **States:** Processing, success, error

4.2 Hardware Interfaces

No direct hardware interfaces. The system runs entirely in the cloud and accesses user hardware (camera, storage) via browser APIs.

Browser APIs Used:

- File API (image upload)
- Camera API (mobile image capture)
- LocalStorage API (token persistence)
- Fetch API (HTTP requests)

4.3 Software Interfaces

4.3.1 MongoDB Atlas Database

- **Interface Type:** MongoDB Wire Protocol
- **Connection:** Mongoose ODM (v8.0+)
- **Connection String:** `mongodb+srv://` with credentials
- **Operations:** CRUD on User, Complaint, OTP collections
- **Data Format:** BSON documents

4.3.2 Google OAuth 2.0

- **Interface Type:** HTTPS REST API
- **Endpoints:**
  - Authorization: `https://accounts.google.com/o/oauth2/v2/auth`
  - Token: `https://oauth2.googleapis.com/token`
  - User Info: `https://www.googleapis.com/oauth2/v2/userinfo`
- **Authentication:** OAuth 2.0 Authorization Code Flow
- **Scopes:** `profile`, `email`

4.3.3 Microsoft Identity Platform

- **Interface Type:** HTTPS REST API
- **Endpoints:**
  - Authorization: `https://login.microsoftonline.com/common/oauth2/v2.0/authorize`
  - Token: `https://login.microsoftonline.com/common/oauth2/v2.0/token`
  - User Info: `https://graph.microsoft.com/v1.0/me`
- **Authentication:** OAuth 2.0 Authorization Code Flow
- **Scopes:** `user.read`

4.3.4 Brevo Email API

- **Interface Type:** HTTPS REST API
- **Endpoint:** `https://api.brevo.com/v3/smtp/email`
- **Authentication:** API Key in `api-key` header
- **Method:** POST with JSON payload
- **Rate Limit:** 300 emails/day (free tier)

4.3.5 Flask AI Service

- **Interface Type:** HTTPS REST API
- **Endpoint:** `https://pothole-detection-ai-4562ae5b30dc.herokuapp.com/predict`
- **Method:** POST with multipart/form-data
- **Input:** Image file under `file` field
- **Output:** JSON with `is_pothole`, `confidence`, `prediction_time`

4.4 Communications Interfaces

Protocol: HTTPS over TCP/IP
Data Format: JSON (REST API), FormData (file uploads)
Compression: Gzip/Brotli (automatic via CDN)
Encoding: UTF-8

API Response Format:

```
{
  "success": true,
  "message": "Operation completed",
  "data": { ... }
}
```

Error Response Format:

```
{
  "success": false,
  "error": "Error message",
  "statusCode": 400
}
```

---

5. Non-Functional Requirements

5.1 Performance Requirements

NFR-PERF-001: The system shall load the homepage within 2 seconds on 4G connection.

- **Metric:** First Contentful Paint (FCP) < 2s
- **Measurement:** Lighthouse performance score > 90

NFR-PERF-002: The AI detection shall complete within 10 seconds.

- **Typical:** 1-3 seconds for average image
- **Maximum:** 10 seconds timeout
- **Measurement:** 95th percentile < 5s

NFR-PERF-003: The backend API shall respond within 500ms for read operations.

- **Measurement:** Average response time < 500ms
- **Threshold:** 99th percentile < 1000ms

NFR-PERF-004: The system shall support 100 concurrent users without degradation.

- **Measurement:** Load testing with 100 virtual users
- **Threshold:** < 5% error rate

NFR-PERF-005: The database queries shall complete within 100ms.

- **Measurement:** MongoDB query profiler
- **Threshold:** < 100ms for indexed queries

5.2 Safety Requirements

NFR-SAFE-001: The system shall not store plain-text passwords.

- **Implementation:** bcrypt hashing with salt
- **Validation:** Code review, security audit

NFR-SAFE-002: The system shall not expose sensitive data in logs.

- **Implementation:** Redact JWT tokens, passwords, API keys
- **Validation:** Log inspection

NFR-SAFE-003: The system shall sanitize all user inputs.

- **Implementation:** express-mongo-sanitize, Helmet CSP
- **Validation:** Penetration testing

NFR-SAFE-004: The system shall handle errors gracefully without crashing.

- **Implementation:** Try-catch blocks, error boundaries
- **Validation:** Error injection testing

5.3 Security Requirements

NFR-SEC-001: The system shall encrypt all data in transit.

- **Standard:** TLS 1.3 (minimum TLS 1.2)
- **Validation:** SSL Labs A+ rating

NFR-SEC-002: The system shall enforce strong password policies.

- **Rules:** Minimum 8 characters, no common passwords
- **Validation:** Password strength meter

NFR-SEC-003: The system shall expire JWTs after 24 hours.

- **Implementation:** JWT `exp` claim with timestamp
- **Validation:** Token expiration testing

NFR-SEC-004: The system shall rate-limit authentication attempts.

- **Implementation:** 5 requests per 15 minutes per IP
- **Validation:** Rate limit testing

NFR-SEC-005: The system shall validate OAuth tokens from Google/Microsoft.

- **Implementation:** Signature verification with public keys
- **Validation:** Invalid token rejection testing

NFR-SEC-006: The system shall protect against XSS attacks.

- **Implementation:** Content-Security-Policy headers
- **Validation:** XSS payload injection testing

NFR-SEC-007: The system shall protect against CSRF attacks.

- **Implementation:** SameSite cookies, token validation
- **Validation:** CSRF attack simulation

5.4 Software Quality Attributes

5.4.1 Availability

- **Target:** 99.5% uptime (43 hours downtime/year)
- **Measurement:** Uptime monitoring (UptimeRobot)
- **Recovery:** Automatic restart on crash (platform-managed)

5.4.2 Maintainability

- **Code Quality:** ESLint/Prettier enforcement
- **Documentation:** Inline comments, README files

- **Modularity:** Separation of concerns (MVC pattern)
- **Testing:** Unit tests with >70% coverage target

5.4.3 Usability

- **Learnability:** First-time users complete detection in < 3 minutes
- **Efficiency:** Returning users submit complaint in < 1 minute
- **Error Recovery:** Clear error messages with recovery actions
- **Accessibility:** WCAG 2.1 Level AA compliance

5.4.4 Scalability

- **Horizontal:** Deploy additional backend/AI instances
- **Vertical:** Upgrade dyno/instance sizes
- **Database:** MongoDB sharding for >100k documents
- **CDN:** Vercel edge caching for static assets

5.4.5 Reliability

- **MTBF:** Mean Time Between Failures > 720 hours (30 days)
- **MTTR:** Mean Time To Repair < 4 hours
- **Error Rate:** < 1% of requests fail
- **Data Integrity:** MongoDB transactions for critical operations

5.4.6 Portability

- **Browser Support:** Chrome, Firefox, Safari, Edge (last 2 versions)
- **Device Support:** Desktop, tablet, mobile (iOS, Android)
- **Platform Support:** Cloud-agnostic (can migrate providers)

5.5 Business Rules

BR-001: Users must verify email via OTP before accessing detection features.
BR-002: Only authenticated users can submit complaints.
BR-003: Images must be analyzed by AI before complaint submission.
BR-004: Complaints must include location, description, and image evidence.
BR-005: OAuth users are automatically verified (no OTP required).
BR-006: Email addresses must be unique across all authentication methods.
BR-007: JWT tokens are stateless and cannot be revoked (expire after 24h).

---

6. System Architecture

6.1 Architectural Overview

The system follows a **microservices architecture** with clear separation of concerns:

```
```

```
┌─────────────────────────────────────────────────────────────┐
│                         CLIENT LAYER                          │
│  React SPA (Vercel CDN) - Global Edge Distribution           │
└─────────────────────────────────────────────────────────────┘
             │                                   │
             │ HTTPS (JSON)                      │ HTTPS (FormData)
             │                                   │
┌────────────▼───────────────────┐   ┌───────────▼─────────────┐
│   APPLICATION LAYER             │   │   AI INFERENCE LAYER     │
│   Node.js/Express REST API      │   │   Python/Flask Service   │
│   (Render Cloud Platform)       │   │   (Heroku Dyno)          │
│                                 │   │                          │
│   - Authentication              │   │   - Image Processing     │
│   - Authorization               │   │   - CNN Model Inference  │
│   - Complaint Management        │   │   - Confidence Scoring   │
│   - Email Integration           │   │                          │
└────────────────────────────────┘   └──────────────────────────┘
             │
             │ MongoDB Wire Protocol
             │
┌────────────▼───────────────────┐
│        DATA LAYER               │
│    MongoDB Atlas Cloud          │
│                                 │
│    - User Collection            │
│    - Complaint Collection       │
│    - OTP Collection             │
└────────────────────────────────┘
```

6.2 Component Descriptions

6.2.1 Frontend (React SPA)

- **Technology:** React 18, React Router v6
- **Styling:** CSS Modules with CSS Variables
- **State Management:** React Hooks (useState, useEffect)
- **Build Tool:** Create React App
- **Deployment:** Vercel Edge Network
- **CDN:** Global distribution with edge caching

Key Components:

- `App.js` - Root component with routing
- `Layout.js` - Page wrapper with navbar/footer
- `LoginPage.js` - Authentication form
- `SignupPage.js` - Registration form
- `VerifyOTPPage.js` - OTP verification
- `PotholePage.js` - Detection and complaint interface
- `OAuthCallbackPage.js` - OAuth redirect handler

6.2.2 Backend API (Node.js/Express)

- **Technology:** Node.js 18, Express 4
- **Authentication:** Passport.js with JWT strategy
- **Database ORM:** Mongoose 8
- **Email Service:** @getbrevo/brevo SDK
- **Security:** Helmet, express-rate-limit, express-mongo-sanitize
- **Deployment:** Render Web Service

Directory Structure:

```
backend/
├── config/
│   ├── db.js              # MongoDB connection
│   ├── passport.js        # OAuth strategies
│   └── loadEnv.js         # Environment variables
├── middleware/
│   └── auth.js            # JWT verification
├── models/
│   ├── User.js            # User schema
│   ├── Complaint.js       # Complaint schema
│   └── OTP.js             # OTP schema
├── routes/
│   ├── authRoutes.js      # Registration/login
│   ├── oauthRoutes.js     # Google/Microsoft OAuth
│   ├── complaintRoutes.js # Complaint CRUD
│   └── contactRoutes.js   # Contact form
├── services/
│   └── emailService.js    # Brevo integration
└── server.js              # Application entry point
```

6.2.3 AI Inference Service (Python/Flask)

- **Technology:** Python 3.10, Flask 3, PyTorch 2.0
- **Model:** Custom CNN (Convolutional Neural Network)
- **Input:** JPEG/PNG images (any resolution)
- **Output:** JSON with classification results
- **Deployment:** Heroku Eco Dyno

Workflow:

1. Receive image via POST `/predict`
2. Preprocess image (resize, normalize)
3. Convert to tensor
4. Forward pass through CNN
5. Calculate softmax probabilities
6. Return JSON response

Response Schema:

```
{
  "is_pothole": true,
  "confidence": 0.9576,
  "prediction_time": 1.23
}
```

6.2.4 Database (MongoDB Atlas)

- **Technology:** MongoDB 7.0
- **Tier:** M0 Sandbox (512 MB storage)
- **Deployment:** AWS us-east-1
- **Backup:** Automatic daily snapshots

Collections:

`users`

```
{
  _id: ObjectId,
  email: String (unique, indexed),
  password: String (bcrypt hash),
  name: String,
  verified: Boolean,
  authProvider: String (local|google|microsoft),
  googleId: String,
  microsoftId: String,
  createdAt: Date,
  updatedAt: Date
}
```

`complaints`

```
{
  _id: ObjectId,
  userId: ObjectId (ref: User),
  location: String,
  description: String,
  imageData: String (base64),
  confidence: Number (0-1),
  status: String (pending|resolved),
  createdAt: Date,
  updatedAt: Date
}
```

`otps`

```
{
  _id: ObjectId,
  email: String (indexed),
  otp: String (hashed),
  expiresAt: Date (TTL index: 10 minutes),
  createdAt: Date
}
```

6.3 Data Flow Diagrams

6.3.1 User Registration Flow

```
User → Frontend → Backend → MongoDB (Create User)
                     ↓
             Brevo API (Send OTP Email)
                     ↓
User → Frontend → Backend → MongoDB (Verify OTP)
```

6.3.2 Pothole Detection Flow

```
User → Frontend (Upload Image)
        ↓
    Flask AI Service (Detect Pothole)
        ↓
    Frontend (Display Results)
        ↓
    Backend (Submit Complaint)
        ↓
    MongoDB (Store Complaint)
```

6.3.3 OAuth Authentication Flow

```
User → Frontend → Backend → Google/Microsoft (Redirect)
                                 ↓
                     User Authorizes (OAuth Consent)
                                 ↓
               Google/Microsoft → Backend (Auth Code)
                                 ↓
                     Backend (Exchange for Token)
                                 ↓
                     MongoDB (Create/Link User)
                                 ↓
                     Frontend (JWT + Redirect)
```

6.4 Security Architecture

Defense in Depth Strategy:

1. **Network Layer:** HTTPS/TLS 1.3 encryption
2. **Application Layer:** Rate limiting, input sanitization
3. **Authentication Layer:** JWT with secure secrets, OAuth validation
4. **Data Layer:** Password hashing, NoSQL injection prevention
5. **Transport Layer:** CORS policies, secure cookies

Security Headers (Helmet.js):

- `Strict-Transport-Security: max-age=31536000`
- `X-Content-Type-Options: nosniff`
- `X-Frame-Options: DENY`
- `X-XSS-Protection: 1; mode=block`
- `Content-Security-Policy: default-src 'self'`

6.5 Deployment Architecture

Production Environment:

Component	Platform	Region	URL
Frontend	Vercel	Global CDN	https://pothole-detect.vercel.app
Backend	Render	US East	https://pothole-detection-backend.onrender.com
AI Service	Heroku	US	https://pothole-detection-ai-4562ae5b30dc.herokuapp.com
Database	MongoDB Atlas	AWS us-east-1	Cluster connection string

Environment Variables:

Backend:

- `MONGO_URI` - MongoDB connection string
- `JWT_SECRET` - JWT signing key (256-bit)
- `SESSION_SECRET` - Session encryption key
- `BREVO_API_KEY` - Email API key
- `GOOGLE_CLIENT_ID` / `GOOGLE_CLIENT_SECRET` - OAuth credentials
- `MICROSOFT_CLIENT_ID` / `MICROSOFT_CLIENT_SECRET` - OAuth credentials
- `FRONTEND_URL` - Vercel production URL
- `NODE_ENV` - `production`

Frontend:

- `REACT_APP_BACKEND_URL` - Backend API URL
- `REACT_APP_AI_SERVICE_URL` - Flask service URL

AI Service:

- (No environment variables required)

---

7. Appendices

7.1 Technology Stack Summary

Layer	Technology	Version	Purpose
Frontend	React	18.2+	UI framework
	React Router	6.20+	Client-side routing
	Axios	1.6+	HTTP client
	CSS Modules	-	Scoped styling
Backend	Node.js	18 LTS	Runtime environment
	Express	4.18+	Web framework
	Mongoose	8.0+	MongoDB ODM
	Passport.js	0.7+	Authentication middleware
	@getbrevo/brevo	3.0+	Email API SDK
	bcryptjs	2.4+	Password hashing
	jsonwebtoken	9.0+	JWT implementation
	Helmet	7.1+	Security headers
	express-rate-limit	7.1+	Rate limiting
AI Service	Python	3.10+	Programming language
	Flask	3.0+	Web microframework
	PyTorch	2.0+	Deep learning framework
	torchvision	0.15+	Image transformations
	Pillow	10.0+	Image processing
Database	MongoDB	7.0	NoSQL database
DevOps	Vercel	-	Frontend hosting
	Render	-	Backend hosting
	Heroku	-	AI service hosting
	Git/GitHub	-	Version control

7.2 API Endpoints Reference

Authentication Endpoints

Method	Endpoint	Description	Auth Required
POST	`/signup`	User registration	No
POST	`/verify-otp`	OTP verification	No
POST	`/resend-otp`	Resend OTP email	No
POST	`/login`	User login	No
GET	`/auth/google`	Google OAuth initiation	No
GET	`/auth/google/callback`	Google OAuth callback	No
GET	`/auth/microsoft`	Microsoft OAuth initiation	No
GET	`/auth/microsoft/callback`	Microsoft OAuth callback	No
GET	`/logout`	User logout	Yes

Complaint Endpoints

Method	Endpoint	Description	Auth Required
POST	`/api/complaints`	Submit complaint	Yes
GET	`/api/complaints`	List all complaints	Yes
GET	`/api/complaints/:id`	Get complaint by ID	Yes

AI Service Endpoints

Method	Endpoint	Description	Auth Required
POST	`/predict`	Detect pothole in image	No
GET	`/health`	Health check	No

Contact Endpoints

Method	Endpoint	Description	Auth Required
POST	`/contact`	Send contact message	No

7.3 Error Codes

Status Code	Error Code	Description
400	INVALID_INPUT	Request validation failed
401	UNAUTHORIZED	Missing or invalid JWT token
403	FORBIDDEN	Insufficient permissions

Status Code	Error Code	Description
404	NOT_FOUND	Resource does not exist
409	CONFLICT	Email already registered
413	PAYLOAD_TOO_LARGE	Image exceeds 10 MB limit
429	TOO_MANY_REQUESTS	Rate limit exceeded
500	INTERNAL_ERROR	Server error
502	BAD_GATEWAY	AI service unavailable
503	SERVICE_UNAVAILABLE	Temporary outage
504	GATEWAY_TIMEOUT	AI processing timeout

7.4 Glossary

Base64 Encoding: Binary-to-text encoding scheme for image storage
bcrypt: Password hashing algorithm with salt
CNN: Convolutional Neural Network for image classification
Cold Start: Delay when inactive cloud service restarts
CORS: Cross-Origin Resource Sharing security mechanism
Dyno: Heroku application container
Edge Network: Geographically distributed CDN nodes
Glass-morphism: UI design style with frosted glass effect
JWT: JSON Web Token for stateless authentication
Middleware: Software layer between request and route handler
OAuth 2.0: Open standard for token-based authorization
ODM: Object Document Mapper (Mongoose for MongoDB)
OTP: One-Time Password for email verification
PyTorch: Open-source machine learning framework
Rate Limiting: Restricting request frequency per client
REST: Representational State Transfer architecture
Softmax: Neural network activation function for probabilities
SPA: Single Page Application
TLS: Transport Layer Security encryption protocol
TTL Index: Time-To-Live database index for auto-deletion

7.5 Testing Strategy

Unit Testing

- **Frontend:** Jest + React Testing Library
- **Backend:** Mocha/Chai or Jest
- **Target Coverage:** 70%+ for critical paths

Integration Testing

- API endpoint testing with Supertest
- Database integration tests with MongoDB Memory Server
- OAuth flow testing with mock providers

End-to-End Testing

- Selenium/Cypress for user journey testing
- Test scenarios: Registration → Detection → Complaint

Performance Testing

- Load testing with Apache JMeter or Artillery
- Target: 100 concurrent users, < 5% error rate

Security Testing

- OWASP ZAP vulnerability scanning
- Manual penetration testing
- Dependency vulnerability scanning (npm audit)

7.6 Maintenance and Support

Monitoring:

- Uptime monitoring: UptimeRobot (5-minute checks)
- Error tracking: Sentry or LogRocket
- Performance monitoring: Vercel Analytics, Render Metrics

Backup Strategy:

- MongoDB automatic daily snapshots (7-day retention)
- Git repository backups (GitHub)
- Environment variable backups (encrypted file)

Update Schedule:

- Security patches: Within 24 hours of release
- Dependency updates: Monthly review
- Feature releases: Bi-weekly sprints

Support Channels:

- Email: support@pothole-detection.com
- GitHub Issues: Bug reports and feature requests
- Documentation: README and SRS updates

7.7 Future Enhancements

Phase 2 Features:

- Admin dashboard for municipal authorities

- Real-time notifications via WebSocket
- Geolocation-based complaint mapping
- Mobile native apps (React Native)
- Batch image processing
- Historical analytics and reporting

Technical Improvements:

- Migrate to TypeScript for type safety
- Implement Redis caching layer
- Add Elasticsearch for complaint search
- Containerize with Docker/Kubernetes
- Implement CI/CD pipeline (GitHub Actions)
- Add GraphQL API alternative

Model Enhancements:

- Retrain CNN with larger dataset
- Implement ensemble learning
- Add severity classification (mild/moderate/severe)
- Support video processing
- Edge deployment for offline detection

Document Approval

Role	Name	Signature	Date
Project Lead	[Your Name]	________	November 18, 2025
Technical Architect	[Your Name]	________	November 18, 2025
QA Lead	[Your Name]	________	November 18, 2025

Document Version: 1.0
Last Updated: November 18, 2025
Next Review Date: December 18, 2025

This SRS document is a living document and will be updated as the project evolves. All stakeholders should refer to the latest version on GitHub.