AI Shield

Architecture & Design Document

Version 1.0

Generated: November 16, 2025

A Comprehensive PII Detection & Redaction System

Table of Contents

1. Executive Summary

AI Shield is a comprehensive PII (Personally Identifiable Information) detection and redaction system designed to protect sensitive data before it reaches AI/LLM systems. The platform provides multiple interfaces including a modern web application and a Chrome extension for real-time protection.

Key Features:

■	**PII Detection**	Detects emails, phones, SSNs, credit cards, Aadhaar, PAN cards
■	**Audio Processing**	Transcribes and analyzes audio files using AssemblyAI
■	**LLM Integration**	Safe prompt sending via Groq API (Llama 3.1)
■	**Audit Logging**	Complete activity tracking for compliance
■	**Web Interface**	Modern React-based ChatGPT-style UI
■	**Browser Extension**	Real-time protection on ChatGPT, Claude, Gemini

The system operates on a simple yet powerful principle: detect sensitive information, redact it with placeholder tokens, and only then process it through AI systems. This ensures that personally identifiable information never leaves the user's control in its original form.

2. System Overview

Architecture Layers

AI Shield follows a modern three-tier architecture with clear separation of concerns:

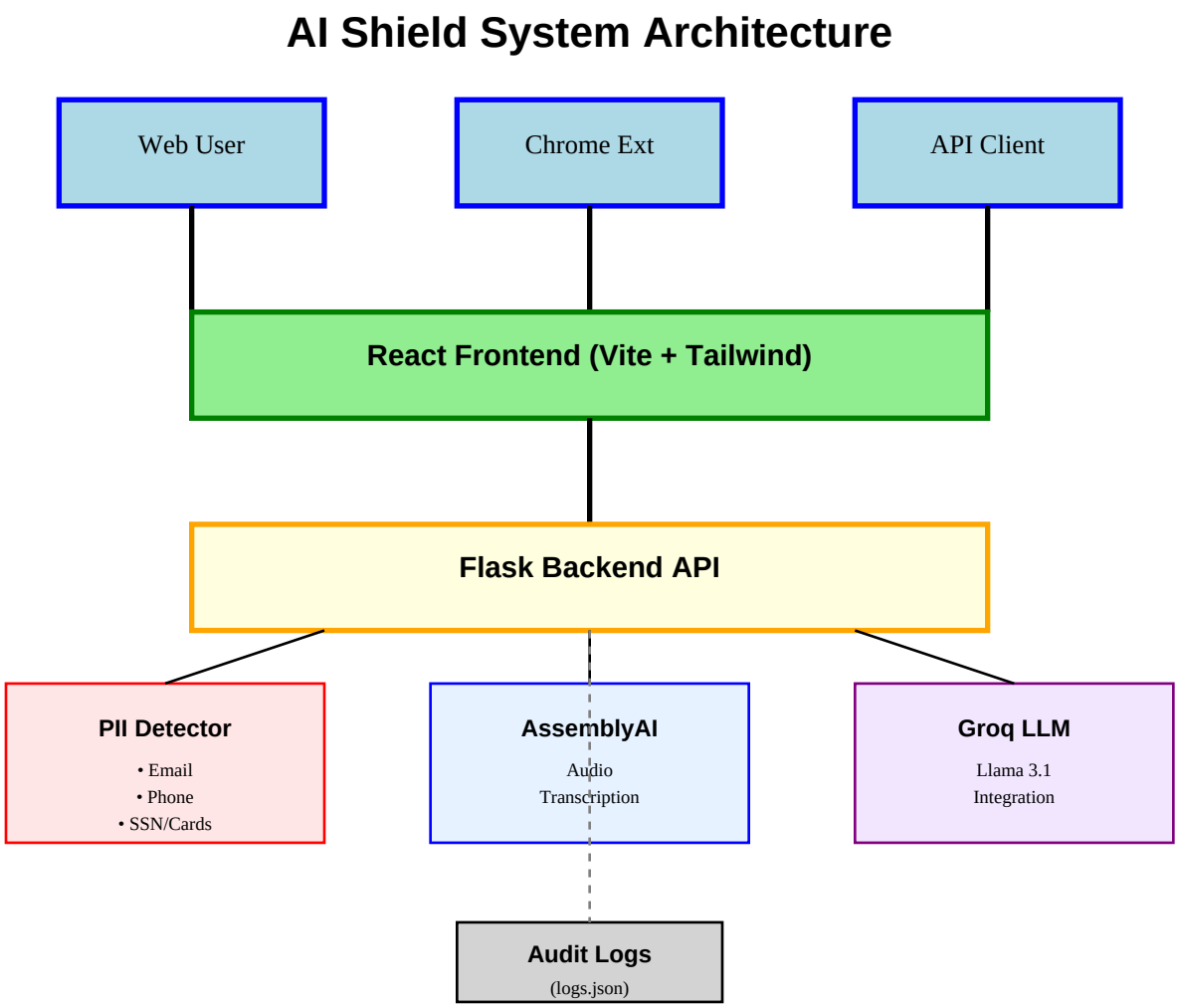
Layer	Technology	Purpose
Presentation	React 18 + Vite + Tailwind	User interface for web and extension
Application	Flask + CORS	REST API server and business logic
Processing	Regex + AssemblyAI + Groq	PII detection, transcription, LLM integration
Storage	JSON File System	Audit logs and configuration

Supported Platforms

Platform	Support Type	Features
Web Application	Full	All features including audio upload
ChatGPT	Extension	Real-time PII warning and sanitization
Claude AI	Extension	Real-time PII warning and sanitization
Google Gemini	Extension	Real-time PII warning and sanitization
Perplexity AI	Extension	Real-time PII warning and sanitization

3. Architecture Diagrams

3.1 High-Level System Architecture



The architecture follows a clean separation between presentation (React frontend), application logic (Flask backend), and external services (AssemblyAI, Groq). All user interactions flow through the frontend, which communicates with the backend via REST APIs. The backend orchestrates PII detection, audio transcription, and LLM integration.

4. Component Details

4.1 Backend Components

Component	File	Responsibility
Main API	app.py	Flask routes, CORS, endpoint handling
PII Detector	pii_detector.py	Regex-based pattern matching for sensitive data
Groq Integration	groq_api.py	LLM API calls using Llama 3.1 8B Instant
AssemblyAI	assembly_api.py	Audio transcription and processing

4.2 Frontend Components

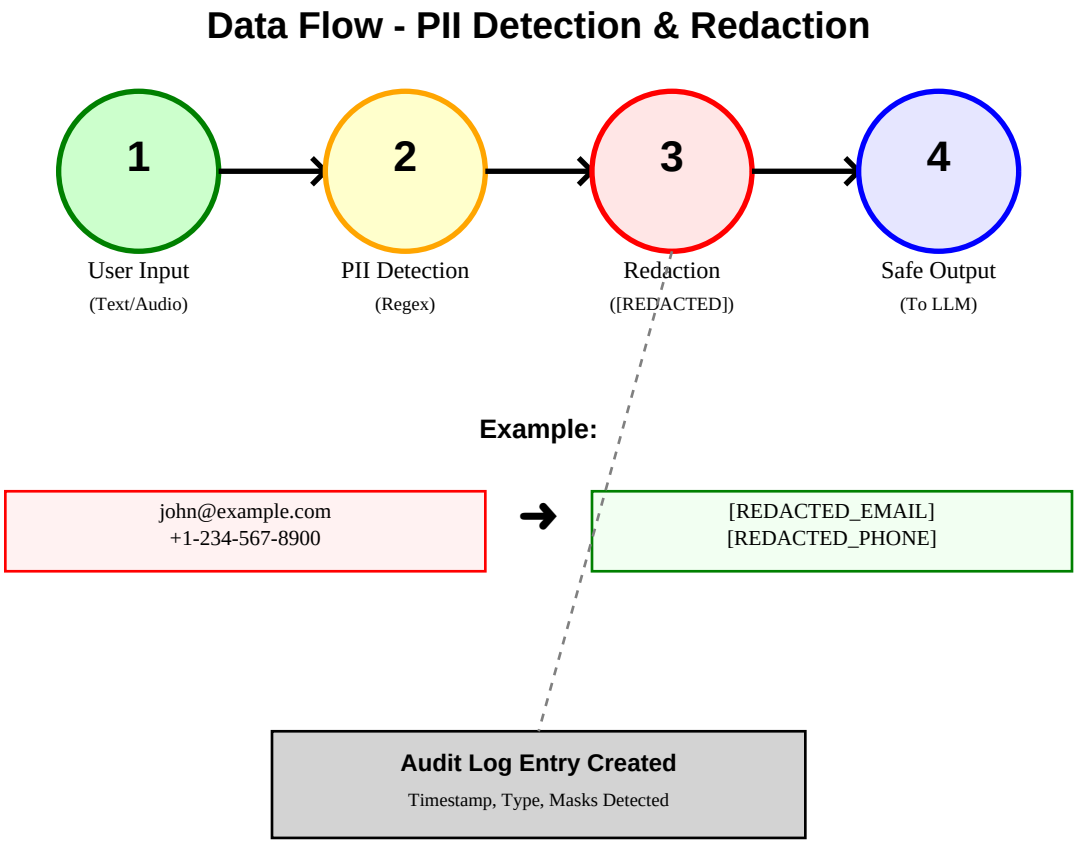
Component	File	Responsibility
Main App	App.jsx	Application shell and routing
Audio Uploader	AudioUploader.jsx	Audio file handling and upload
Redaction Result	RedactionResult.jsx	Display sanitized output
Log Viewer	LogViewer.jsx	Audit trail visualization
AI Shield SDK	aiShieldSdk.js	API wrapper and client library

4.3 Chrome Extension Components

Component	File	Responsibility
Manifest	manifest.json	Extension configuration and permissions
Background	background.js	Service worker for background tasks
Content Script	content_script.js	DOM monitoring and injection
Popup	popup.html/js	Extension popup interface
Sanitizer	libs/sanitizer.js	Client-side PII detection

5. Data Flow

5.1 PII Detection & Redaction Flow



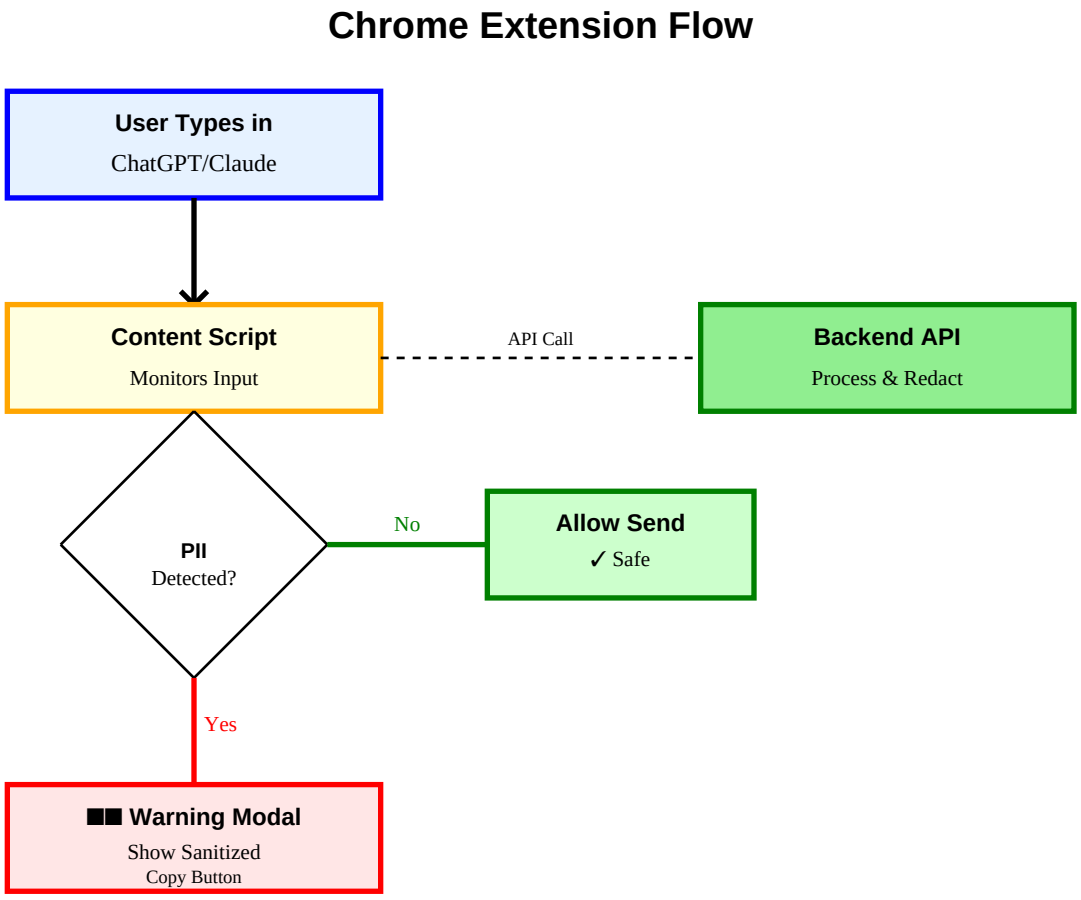
5.2 Processing Steps

Step	Action	Output
1	User submits text or audio	Raw input data
2	Backend applies regex patterns	List of detected PII entities
3	Replace PII with [REDACTED_TYPE] tokens	Sanitized text
4	Log detection event to audit trail	Log entry with timestamp
5	Return sanitized text to user	Safe data for LLM processing

6	(Optional) Send to Groq LLM	AI response with no PII exposure

6. Chrome Extension Architecture

6.1 Extension Flow Diagram



6.2 Extension Workflow

The Chrome extension operates seamlessly in the background, monitoring user input on supported AI platforms. When a user types a message containing PII, the content script intercepts the input before submission, sends it to the backend API for analysis, and displays a warning modal if sensitive data is detected. Users can then copy the sanitized version with a single click.

6.3 Supported AI Platforms

✓	ChatGPT	chatgpt.com	Full support with submit button monitoring
✓	Claude	claude.ai	Full support with submit button monitoring
✓	Gemini	gemini.google.com	Full support with submit button monitoring
✓	Perplexity	perplexity.ai	Full support with submit button monitoring

7. Security Features

7.1 PII Detection Patterns

AI Shield employs sophisticated regex patterns to detect various types of sensitive information:

Type	Pattern Description	Example	Redaction Token
Email	Standard email format	user@example.com	[REDACTED_EMAIL]
Phone (US)	US phone formats	(123) 456-7890	[REDACTED_PHONE]
Phone (Intl)	International format	+91-98765-43210	[REDACTED_PHONE]
SSN	US Social Security	123-45-6789	[REDACTED_SSN]
Credit Card	Major card issuers	4532-1234-5678-9010	[REDACTED_CC]
Aadhaar	12-digit Indian ID	1234 5678 9012	[REDACTED_AADHAAR]
PAN Card	Indian tax ID	ABCDE1234F	[REDACTED_PAN]

7.2 Security Measures

✓ **PII Detection Before Processing**: All text is scanned before sending to external APIs

✓ **Comprehensive Audit Logging**: Every detection event is logged with timestamp and details

✓ **CORS Protection**: Backend enforces CORS policies for API security

✓ **Environment-based Configuration**: Sensitive API keys stored in environment variables

✓ **Client-side Validation**: Frontend validates input before API calls

✓ **User Consent Workflow**: Extension requires user action before processing

✓ **No Data Retention**: Original text is never stored on servers

✓ **Local Processing Option**: Extension can work with local backend

8. API Endpoints

8.1 Endpoint Overview

Method	Endpoint	Purpose	Authentication
POST	/process_text	Detect and redact PII in text	None
POST	/process_audio	Transcribe audio and detect PII	None
POST	/ask_llm	Send sanitized prompt to LLM	Groq API Key
GET	/logs	Retrieve audit trail	None

8.2 Request/Response Examples

POST /process_text

```
Request:
{ "text": "My email is john@example.com" }

Response:
{
"original": "My email is john@example.com",
"redacted_text": "My email is [REDACTED_EMAIL]",
"masks": [{"type": "EMAIL", "text": "john@example.com"}]
}
```

POST /ask_llm

```
Request:
{ "sanitized": "Tell me about [REDACTED_NAME]" }

Response:
{
"choices": [{"message": {"content": "I'd be happy to help..."}}]
}
```

9. Technology Stack

9.1 Backend Technologies

Technology	Version	Purpose
Python	3.8+	Core programming language
Flask	Latest	Web framework for REST API
Flask-CORS	Latest	Cross-origin resource sharing
Groq SDK	Latest	LLM API integration (Llama 3.1 8B)
AssemblyAI	Latest	Audio transcription service
Regex	Built-in	Pattern-based PII detection

9.2 Frontend Technologies

Technology	Version	Purpose
React	18	UI library and component framework
Vite	Latest	Build tool and development server
Tailwind CSS	Latest	Utility-first CSS framework
JavaScript/JSX	ES6+	Programming language
Node.js	16+	JavaScript runtime environment

9.3 Extension Technologies

Technology	Version	Purpose
Manifest V3	3	Chrome extension platform
JavaScript	ES6+	Core programming language
Chrome APIs	Latest	Browser integration (storage, tabs, runtime)
Service Worker	Latest	Background processing
Content Scripts	Latest	DOM monitoring and injection

10. Deployment Guide

10.1 Prerequisites

✓ Python 3.8 or higher installed

✓ Node.js 16 or higher installed

✓ Groq API Key (for LLM features)

✓ AssemblyAI API Key (for audio transcription)

✓ Chrome browser (for extension)

10.2 Backend Deployment Steps

1. Navigate to the backend directory

2. Create a Python virtual environment: `python -m venv venv`

3. Activate the virtual environment

4. Install dependencies: `pip install -r requirements.txt`

5. Copy .env.example to .env and add API keys

6. Run the server: `python app.py`

7. Backend will be available at http://localhost:5000

10.3 Frontend Deployment Steps

1. Navigate to the frontend directory

2. Install dependencies: `npm install`

3. (Optional) Configure API URL in .env

4. Run development server: `npm run dev`

5. Access application at http://localhost:5173

6. For production: `npm run build`

10.4 Chrome Extension Installation

1. Ensure backend is running on localhost:5000

2. Open Chrome and navigate to chrome://extensions/

3. Enable 'Developer mode' toggle (top right)

4. Click 'Load unpacked' button

5. Select the 'ai-shield-extension' folder

6. Extension icon will appear in Chrome toolbar

7. Configure backend URL in extension options if needed

10.5 Environment Variables

Variable	Location	Description
GROQ_API_KEY	backend/.env	API key for Groq LLM service
ASSEMBLYAI_API_KEY	backend/.env	API key for AssemblyAI transcription
PORT	backend/.env	Backend server port (default: 5000)
VITE_API_URL	frontend/.env	Backend API URL (default: localhost:5000)

For questions, issues, or contributions, please visit the GitHub repository or contact the TechnoSleuths team. This system is built with security and privacy as top priorities, ensuring that sensitive information is always protected when interacting with AI systems.