PPE-WATCHER User & Deployment Manual

An AI-based PPE Detection and Monitoring System using YOLOv8, Fastify, Prisma, React, and SQLite

Author: Hui Sun

University of Canberra – Master of Information Technology and Systems

Date: October 7, 2025

Table of Contents

1． System Overview

1.1 Project Background and Objectives

PPE-WATCHER is an AI-driven safety monitoring system designed to detect the use of personal protective equipment (PPE) such as helmets, masks, gloves, and vests in workplaces or laboratories.
The project aims to enhance occupational safety by automatically identifying PPE violations in real time, reducing manual inspection workload, and providing evidence-based alerts for supervisors.

1.2 System Overview

The system integrates an edge-based AI model (YOLOv8) for object detection, a Fastify backend for data management and communication, a React-MUI frontend for visualization, and an SQLite database for storage.
It provides detection, record logging, alerting, visualization, and configuration functionalities accessible through a web interface.

1.3 Key Features

- Real-time PPE detection and image upload from edge devices.
- Automatic alert generation and notification via email/SMS.
- Dashboard for visual statistics and key performance indicators.
- Configurable system parameters (SMTP, Twilio, PPE types).
- Historical record management and data export.

1.4 Workflow

1. The edge device captures live video using a camera.
2. The YOLOv8 model detects PPE compliance and sends JSON + image data to the backend.
3. The backend validates, stores, and broadcasts the violation event.
4. The notification service sends alerts via SendGrid or Twilio.
5. The frontend updates the dashboard and record list in real time.

1.5 Example Use Cases

- Construction sites detecting workers without helmets.
- Research labs monitoring staff wearing safety goggles.
- Industrial manufacturing detecting gloves or vests compliance.

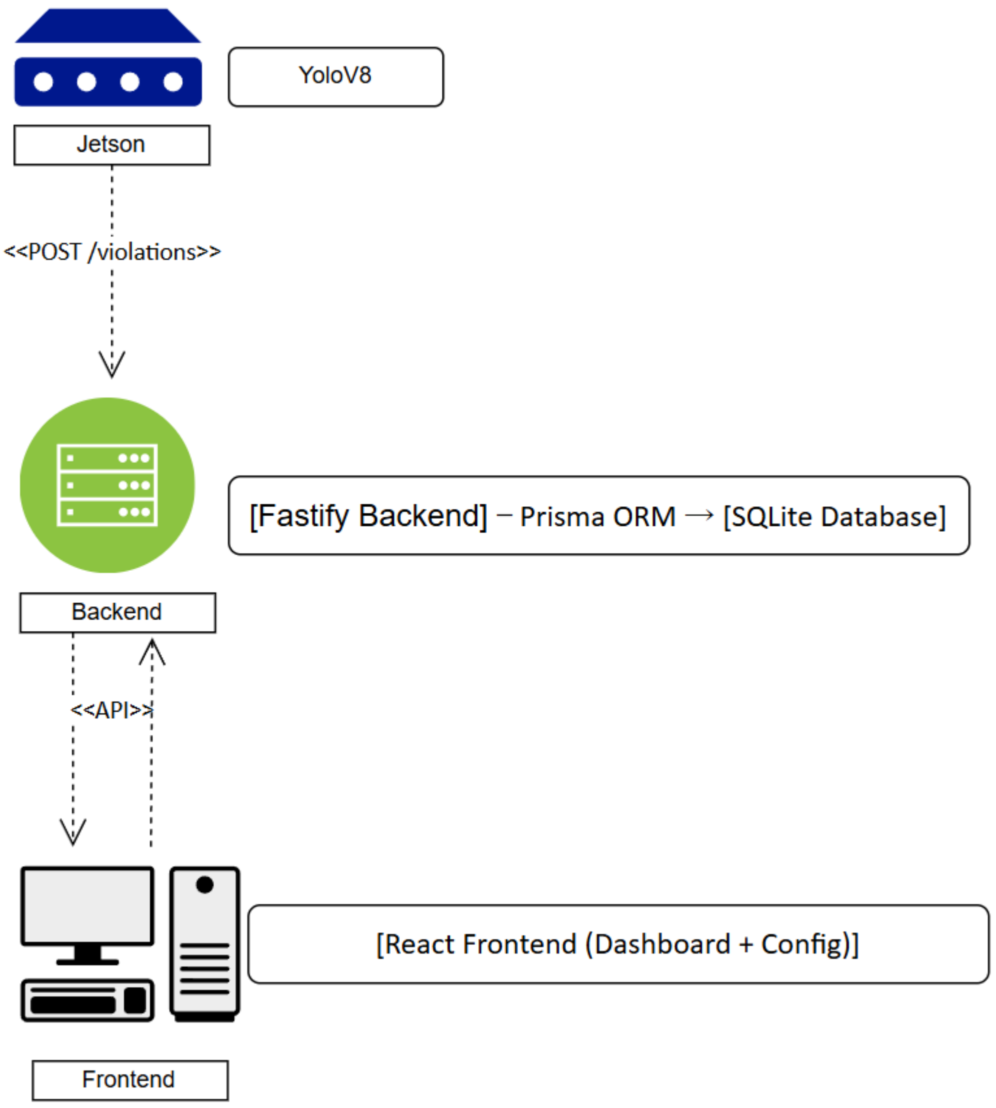
2． System Architecture

2.1 Component Modules

PPE-WATCHER consists of four key components:

- **Frontend (React + MUI)** – provides the user interface.
- **Backend (Fastify + Prisma)** – handles APIs, database operations, and notifications.
- **Database (SQLite)** – stores violations, configurations, and contacts.
- **Edge AI (YOLOv8)** – performs PPE detection and uploads events.

2.2 Local Architecture Diagram


YoloV8
Jetson
<<POST /violations>>
[Fastify Backend] – Prisma ORM → [SQLite Database]
Backend
<<API>>
[React Frontend (Dashboard + Config)]
Frontend

2.3 Component Descriptions

- **Frontend:** Built with React, Material UI, and Axios for API calls.
- **Backend:** Fastify framework with Prisma ORM and Pino logging.
- **Database:** SQLite file located in /backend/prisma/dev.db.
- **Edge Device:** Python script using OpenCV and YOLOv8 for PPE detection.

3. Environment and Requirements

3.1 Development Requirements

- Node.js v18 or higher
- npm v9 or higher
- SQLite3
- Python 3.8+ (for YOLOv8)
- Git, VSCode, or WebStorm

3.2 Runtime Requirements

- OS: Windows 10+, macOS, or Linux
- RAM: Minimum 8GB
- Storage: 1GB free space
- Network: Localhost or LAN connection

3.3 Software Dependencies

- Backend: Fastify, Prisma, Nodemailer, Twilio, Pino
- Frontend: React, MUI, Axios, React Router
- AI: Ultralights YOLOv8, OpenCV, Requests

3.4 Network and Port Configurations

- Backend: http://localhost:8080
- Frontend: http://localhost:3000
- SQLite: local file, no network port required

Note:
If you change the backend API port or base URL, remember to update the corresponding endpoint in the frontend client configuration (for example, in src/api/client.ts). Otherwise, the frontend may fail to connect to the backend.

3.5 Folder Structure Overview

```
PPE-WATCHER/
├── backend/              # Fastify API Server (Node.js + Prisma)
```

```
|   ├── src/              # Backend source code
|   ├── prisma/
|   |   └── dev.db         # SQLite local database
|   └── package.json        # Backend dependencies & scripts
|
├── frontend/              # React + MUI Frontend
|   ├── src/              # React source code
|   ├── build/             # Compiled static files (for S3/CloudFront)
|   └── package.json        # Frontend dependencies & scripts
|
└── ai-edge/              # Edge AI (Jetson / Raspberry Pi)
    └── upload_yolo.py       # YOLOv8 model upload and alert script
```

3.6 Environment Variables Configuration

The PPE-WATCHER backend uses a minimal set of environment variables to define essential runtime settings such as the server port, authentication secret, and database path.
All variables are stored in a .env file located in the /backend/ directory.
These values should be customized before deployment, and the .env file must **not** be committed to version control.

Backend (.env file in /backend/)

```
PORT=8080
JWT_SECRET=your-secure-random-string
DATABASE_URL="file:./dev.db"
```

Parameter Descriptions

- **PORT** – Defines the HTTP port used by the Fastify backend (default: 8080).
- **JWT_SECRET** – A long, random string used to sign and verify JSON Web Tokens for authentication.
- **DATABASE_URL** – Path to the SQLite database used by Prisma. The default value points to backend/prisma/dev.db.

Notes

- Update the PORT value if the backend server runs on a different port; remember to change the frontend API base URL accordingly.
- The JWT_SECRET should be unique and confidential to ensure secure authentication.

- For production environments, consider using a dedicated .env.production file with environment-specific values.

4 Installation and Deployment

4.1 Backend Installation

This section installs and runs the PPE-WATCHER backend service.
It installs all dependencies, applies the Prisma database migration, and starts the Fastify server in development mode.

```
cd backend
npm install
npx prisma migrate dev
npx prisma generate
npm run dev
```

Note: Before running command `npx prisma migrate dev` and the backend, ensure that a valid .env file exists in the `/backend/` directory.
It must include at least PORT, JWT_SECRET, and DATABASE_URL.

4.2 Frontend Installation

This section sets up the React-based frontend user interface.
It installs all required dependencies and launches the application locally for testing and configuration.

```
cd frontend
npm install
npm run start
```

4.3 Edge Device Configuration (Jetson)

This section explains how to prepare a Jetson device to run the detector and upload results to the backend. Use **setup.txt** to install and configure the Jetson environment, then run **jetson_run.py** on the device.

Files in this package

setup.txt — Consolidated, engine-first setup for JetPack 6.x (Python 3.10, Ultralytics, Jetson wheels, cuSPARSELt, GStreamer, TensorRT), plus optional camera activation (Jetson-IO) and exFAT USB steps.

jetson_run.py —Run program for Jetson. It reads frames from the CSI camera via GStreamer, runs a two-stage pipeline (Stage-A person detector → Stage-B PPE detector), applies a windowed vote, saves snapshots/CSV, and can upload violations to the backend.

4.3.1Prerequisites (run once per device)

Complete the environment setup using setup.txt.

Install Python 3.10, optional yoloenv venv, Ultralytics, Jetson PyTorch wheels, cuSPARSELt, GStreamer, and enable TensorRT (Export TensorRT engine).

Activate the CSI camera (first use only).

If preview works, the CSI camera is ready.

4.3.2 Model preparation

The .pt files should be Convert to TensorRT first, and you should save them in ~/ (see engine-export step in setup.txt- [9]). Keep both engines in your home directory or update paths at the top of jetson_run.py.

Use a person detector at ~/yolov8m.engine for Stage-A.

Use PPE model for Stage-B (recommended for production).

4.3.3 Network connection

The edge device (Jetson) and the machine running the frontend are connected to the same local network.

Set the IP in jetson_run.py. (BACKEND_URL — HTTP endpoint for uploads. If unset, defaults to http://<local-ip>:8080/violations.)

4.3.4 Run the Jetson program

```
# active the env

source ~/yoloenv/bin/activate

sudo systemctl restart nvargus-daemon

# run the jetson_run.py
```

```
python ~/jetson_run.py
```

4.3.5 Troubleshooting

Camera busy or not found →

```
sudo systemctl restart nvargus-daemon  # confirm nvarguscamerasrc plugin.
```

No detections →

Verify both engines exist and match expected classes; lower CONF_A/CONF_B

Low FPS → prefer TensorRT engines, reduce IMGSZ_A/IMGSZ_B (e.g., 512), avoid rendering preview windows.

4.3.6 Runtime Parameters (jetson_run.py)

Where to change: open jetson_run.py and edit the constants at the top of the file.

Models & I/O:

STAGE_A_PATH: ~/yolov8s.engine

TensorRT engine path for Stage-A (person detector).

STAGE_B_PATH: ~/best.engine

TensorRT engine path for Stage-B (PPE detector).

OUT_DIR: ~/ppe_events_window

Root output directory for events (images + CSV).

IMG_DIR: ~/ppe_events_window/images

Folder for saved event snapshots (annotated JPEGs).

CSV_PATH: ~/ppe_events_window/events.csv

CSV log for events (timestamp, track_id, missing items, box, image path).

REQUIRED_ITEMS: ["boots","gloves","goggles","helmet","mask","vest"]

The set of PPE items required; Stage-B detections are mapped against this list to compute "missing".

Camera capture:

CAM_W: 1280

CSI capture width. Must match the GStreamer caps.

CAM_H: 720

CSI capture height. Must match the GStreamer caps.

CAM_FPS: 30

CSI capture framerate. Must match the GStreamer caps.

Note: mismatches can cause no video or color issues.

Inference:

IMGSZ_A: 640

Input size for Stage-A (person). Smaller → faster but slightly less accurate. Use multiples of 32.

IMGSZ_B: 640

Input size for Stage-B (PPE). Smaller → faster but slightly less accurate. Use multiples of 32.

CONF_A: 0.35

Confidence threshold for Stage-A (higher → fewer false positives; lower → higher recall).

CONF_B: 0.25

Confidence threshold for Stage-B (higher → fewer false positives; lower → higher recall).

PPE vote logic:

RUN_PPE_EVERY: 2

Run Stage-B every N frames (stride). Larger value → fewer PPE inferences → higher FPS but slower reaction.

VOTE_WINDOW_SEC: 1.5

Sliding window length (seconds) used to compute the recent ratio of “missing” samples.

ENTER_RATIO: 0.70

Threshold of “missing” ratio to enter the VIOLATION state (higher → stricter to enter).

EXIT_RATIO: 0.40

Threshold of “missing” ratio to return to OK (lower → exits sooner).

EVENT_COOLDOWN_SEC: 3.0

Minimum time between two events **for the same track** (per-person cooldown, not global).

Global silence:

GLOBAL_SILENCE_SEC: 10.0 (for test)

After one violation event is triggered, suppress **all tracks** from triggering again during this global silence window. Set to 0.0 to disable.

Intended to simulate the time a detected subject needs to re-don PPE; tune to match real-world behavior.

silence.active: False (initial)

Runtime flag indicating whether global silence is currently active; a message is printed when entering silence.

silence.until: 0.0 (initial)

Runtime UNIX timestamp when global silence ends; once the time is reached, silence ends (optionally logs) and triggering resumes.

Logging & heartbeat:

HEARTBEAT_SEC: 5.0

Prints a heartbeat (e.g., [INF] persons=…) every N seconds to indicate live status.

Backend & networking:

BACKEND_URL: http://<local-ip>:8080/violations (default if unset)

HTTP endpoint for uploads. If not provided, the program builds a default URL from the local IP.

Override: export BACKEND_URL=http://<BACKEND_IP>:8080/violations

Tracking:

IOU matching threshold (hardcoded): 0.3

Minimum IoU for associating current boxes to existing tracks.

Track expiry (hardcoded): 2.0s

Tracks not matched for longer than this duration are removed (treated as exited).

4.4 Production Build and Start

This section prepares the PPE-WATCHER system for production deployment.
The backend is compiled with TypeScript to generate executable JavaScript files, then started using Node.js.
The frontend is built into static files optimized for production, which can be hosted on services such as AWS S3, Nginx, or any web server.

```
cd backend
npm run build
npm run start
cd frontend
npm run build
```

5 Frontend Features

5.1 Login and Authentication

Users log in using credentials verified via JWT.
Session tokens are stored securely in browser memory.

5.2 Main Layout (AppShell)

- **Top Bar:** Title and notification icon.
- **Main Navigation:** Dashboard, History, Setting
- **Side Navigation:** Links to Overview, Violations, Config, etc.
- **Footer:** Copyright and status info.

5.3 Module Descriptions

The PPE-WATCHER web interface is organized into three main menus: **Dashboard**, **History**, and **Settings**.
Each menu contains several sub-modules designed to provide monitoring, alert management, and configuration capabilities for the system.
The table below summarizes all modules and their key functionalities.

System Module Overview

Main Menu	**Sub-Menu**	**Description**
Dashboard	**Overview**	Displays real-time summary cards showing the total number of PPE violations, open vs. resolved cases, and recent activity. Provides a quick overview of system health and compliance.
	Alert	Shows the most recent alerts generated by edge devices. Each alert includes timestamp, detected PPE type, and status. Users can acknowledge or investigate alerts directly.
History	**Violation**	Displays a full list of violation records with filters for type, date, and status. Users can resolve violations or view detailed snapshots captured by edge cameras.
	Trends	Presents graphical analytics and time-series charts to identify violation patterns and frequency trends over selected time ranges.
	Notification	Shows a chronological log of email and SMS notifications sent by the system, including type (alert/resolution) and timestamp.
	Bookmark	Allows users to save specific violation records for quick access. Useful for tracking high-priority or recurring safety issues.
Settings	**Alert Setting**	Configures how alerts are delivered (Email/SMS) and determines active notification channels. Supports test sending for verification.
	Contacts	Manages recipient list for notifications. Users can add, edit, or delete contact information used for alert delivery.
	Sender Config	Defines system sender settings including SMTP credentials, Twilio SMS API, and sender email address. Supports secure testing and saving configuration.
	Security	Allows administrators to manage user credentials, passwords, and access permissions. Includes logout and session management functions.

6 Backend Services

6.1 API Overview

Provides RESTful endpoints for CRUD operations and alert triggers.

6.2 Routing and Module Structure

Table 6.2 – Backend Libraries and Routes

Folder (Path)	**File**	**Description**
src/lib	prisma.ts	Initializes and exports the Prisma client for SQLite database connection and queries.

Folder (Path)	File	Description
	notifier.ts	Provides a unified notification service to send Email (via SendGrid SMTP) and SMS (via Twilio API).
src/routes	_utils.ts	Shared helper functions and constants used across different route handlers.
	auth.ts	Handles user authentication, token verification, and access control.
	user.ts	Manages user account information. Only one admin-type user exists and must be added directly through the database.
src/routes	violation.ts	Provides full CRUD operations for PPE violations, including image snapshot storage, status updates, and resolution handling.
	notification.ts	Manages system-generated notifications for Email and SMS alerts.
	contact.ts	Allows adding, editing, and deleting alert recipients (email and phone contacts).
	config.ts	Handles system configuration for SMTP, Twilio, PPE detection types, and sender email address.
	bookmark.ts	Allows users to bookmark important violation records for quick access.
	events.ts	Manages Server-Sent Events (SSE) for real-time communication with the frontend.

6.3 Major Endpoints

- POST /violations – Upload detection result.
- GET /violations – Retrieve records.
- PATCH /violations/:id/resolve – Mark record as resolved.
- POST /config – Update configuration values.

6.4 Notification Logic

When a violation is created:

1. The backend saves it to SQLite.
2. A notification entry is generated.
3. Email or SMS is sent based on configuration.

6.5 Data Persistence (SQLite)

Prisma ORM manages models, migrations, and queries.
Data is stored in backend/prisma/dev.db.

7 Database and Data Management

7.1 Database Overview

SQLite is used as the local database for lightweight, file-based storage.
It stores essential system data such as user accounts, contact lists, violation records, and notification logs.

This approach simplifies deployment and eliminates the need for a separate database server. The database file (dev.db) is located under the backend/prisma directory and can be easily backed up or restored when needed.

7.2 Main Table Structures

Table name	Description
user	User information
violation	Detected PPE violations
notification	Notifications of violation and it’s status
contact	Notification recipients
System Config	SMTP/SMS configurations

7.3 Export and Import

To ensure data safety, the PPE-WATCHER system allows manual backup and restoration of the SQLite database.
The following shell commands can be executed from the project root directory to back up or recover the current database file.
Backup Command：

```
cp backend/prisma/dev.db backups/dev-$(date +%F).db
```

This command copies the current database file (dev.db) from the backend folder to the backups directory.
The $(date +%F) expression automatically inserts the current date in the filename (for example, dev-2025-10-07.db),
ensuring that each backup file is uniquely named by date.
It allows you to maintain multiple historical versions of the database safely.

Restore Command：

```
cp backups/dev-xxxx.db backend/prisma/dev.db
```

This command replaces the active database with the selected backup file, effectively restoring the system to a previous state.
Both commands must be run from the root of the project, and the backups folder must exist beforehand.
These commands work on Linux, macOS, or Windows with Git Bash or WSL.
Because the cp command overwrites files without confirmation, it should be used carefully when restoring data.

Note:
The two cp commands are used to quickly back up and restore the SQLite database — the first creates a dated copy, and the second restores a selected version.

7.4 Prisma Studio and IDE Database Tools

The database can be visualized and managed using two methods:

1. **Prisma Studio** – Run `npx prisma studio` in the backend directory to open a web interface for browsing and editing records.
2. **Integrated Development Environment (IDE)** – Modern IDEs such as WebStorm or VS Code provide built-in database management tools that can directly open the dev.db file under backend/prisma/.

8 Configuration and Notification

8.1 Email Configuration (SendGrid)

- smtp_host: smtp.sendgrid.net
- smtp_port: 587
- smtp_user: apikey
- smtp_password: SG.xxxxx
- smtp_sender: no-reply@domain.com

8.2 SMS Configuration (Twilio)

- sms_provider: twilio
- sms_sid: ACxxxx
- sms_token: secret
- sms_from: +1234567890
-

Note: The specific parameters may vary depending on the email or SMS service provider. Please refer to your provider's documentation for accurate configuration details.

8.3 System Parameters

Includes PPE detection types, sender email configuration, and notification service credentials (SMTP and Twilio).

Detection types and alert delivery methods (SMS, Email, or both) can be managed under **Settings → Alert Setting**,
while sender email and notification credentials can be configured under **Settings → Sender Config**.

8.4 Notification Triggers

Triggered when new violations are added or status changes occur.

8.5 Troubleshooting Configuration Issues

Check API credentials, ports, and verified sender numbers.

If the problem persists, review the response messages or error logs returned from the backend service providers (e.g., Twilio or SendGrid) for more details.

9 Maintenance and Backup

9.1 SQLite Backup and Restore

Database backup and restoration are essential for data safety.
For detailed procedures and shell commands, refer to **Section 7.3: Export and Import**.
It describes how to copy, version, and restore the SQLite database using manual or automated methods.

9.2 Logs and Monitoring

The backend integrates **Pino** logger within the **Fastify** framework for real-time logging and monitoring.
All API requests, responses, and system errors are automatically recorded in the console output or log file.
Logs include timestamps, request paths, response codes, and error messages, which help administrators debug and monitor system activity.

Example log output:

```
[INFO] (Server) Request received at /violations - 200 OK
[ERROR] (Notifier) Email sending failed: Authentication error
```

9.3 Server Restart and PM2 Management

To manage and restart the backend server in a production environment, it is recommended to use **PM2** (Process Manager 2).
PM2 keeps the backend running continuously, even after terminal closure or system reboot.

Installation:

```
npm install -g pm2
```

Basic Commands:

```
pm2 start dist/server.js     # Start the compiled backend service
pm2 logs                     # View real-time server logs
pm2 restart all              # Restart all running services
```

Note:
If PM2 is not installed, please download it first using the above installation command.
PM2 is an optional but recommended tool for stable backend management during long-term or production operation.

9.4 Security and Update Recommendations

Keep all dependencies and system libraries updated to ensure stability and security.
When upgrading packages, **verify compatibility between dependencies** to prevent runtime issues.

Regularly **rotate API keys and credentials** for email and SMS providers to reduce the risk of unauthorized access.
It is also recommended to back up configuration files before applying major updates or dependency changes.

10 Security and Access Control

10.1 Authentication Mechanism

The PPE-WATCHER backend uses **JWT (JSON Web Token)** for authentication.
Each user session is represented by a signed token generated during login.
The token includes encoded user information and an expiration time, ensuring secure and stateless authentication.
JWTs are stored client-side (in browser memory or local storage) and must be included in the Authorization header of each request.
This design avoids storing session data on the server and supports scalable, multi-instance deployments.

10.2 Role Definitions

The system contains a single role: **User**.
This role has full administrative privileges, including the ability to view, update, and configure all system modules.
User accounts are created directly in the database; there is no frontend interface for user registration or management.

10.3 API Security

All protected API routes require a valid **Bearer token** in the HTTP header for access.
Requests without valid authentication are automatically rejected with a 401 Unauthorized response.
Access control middleware verifies token validity and user permissions before allowing data retrieval or modification.
Additionally, API endpoints are restricted by CORS configuration, limiting access to trusted origins only.
Example header:

```
Authorization: Bearer <JWT_token>
```

10.4 Data Encryption and Environment Variables

Sensitive credentials such as database connection strings, SMTP passwords, and Twilio tokens are stored securely in the `.env` file.
These variables are loaded via the **dotenv** package during runtime and are never hard-coded in source files.
The `.env` file is excluded from version control using `.gitignore` to prevent accidental leaks.
All communication between frontend and backend occurs over **HTTPS**, ensuring encryption in transit.

11 Version and Future Plans

11.1 Current Version Info

Version: v1.0.0
Date: October 2025

11.2 New Features

Integrated email/SMS alerts, configuration panel, and dashboard statistics.

11.3 Known Issues

- Limited to single camera feed.
- Manual configuration of SMTP/SMS required.

11.4 Future Development Plan

- Multi-device support
- Cloud-based model synchronization
- Enhanced user management

12 Appendix

12.1 Project Folder Structure

```
PPE-Watcher/
├─── backend/                 # Backend service (Node.js + Fastify + Prisma)
│    ├─── node_modules/          # Backend dependencies
│    ├─── prisma/               # Prisma data models and migration files
│    ├─── src/                 # Source code (API routes, logic, utilities)
│    ├─── uploads/              # Uploaded images or log files
│    ├─── .env                 # Environment variables configuration
│    ├─── .gitignore            # Git ignore rules
│    ├─── package.json           # Backend dependencies and scripts
│    ├─── package-lock.json       # Dependency lock file
│    └─── tsconfig.json          # TypeScript compiler configuration
│
├─── documents/                # Project documentation
│    ├─── ai-edge/              # Edge device scripts and AI models
│    ├───video_links.txt       # This file contains cloud links to demo and presentation
│    ├─── PPE-WATCHER Frontend Functional Manual.docx
│    └─── PPE-WATCHER_User_Deployment_Manual_v1.1.docx
│
├─── frontend/                 # Frontend project (React + TypeScript + MUI)
│    ├─── node_modules/          # Frontend dependencies
│    ├─── public/               # Static resources (HTML, icons, etc.)
│    ├─── src/                 # Source code (components, pages, APIs)
│     ├──── .eslintrc.json          # ESLint configuration
│     ├──── .prettierrc             # Prettier formatting configuration
│     ├──── package.json            # Frontend dependencies and scripts
│     ├──── package-lock.json       # Dependency lock file
│     └──── tsconfig.json           # TypeScript compiler configuration
│
├──── .gitignore                    # Global Git ignore file
└──── README.md                     # Project description and setup guide
```

12.2 API Examples

Endpoint	Request
POST /violations	***Json*** {"file": "<binary-image-file>", "kinds": ["no_helmet", "no_vest"]}

Uploads a new PPE violation record detected by the edge device.
This endpoint accepts multipart form data containing an image file and a JSON-encoded list of violation types.
All other fields (id, ts, status, snapshotUrl) are generated automatically by the backend.

12.3 Contact and Support Info

PPE_WATCHER TEAM
u3276283@uni.canberra.edu.au