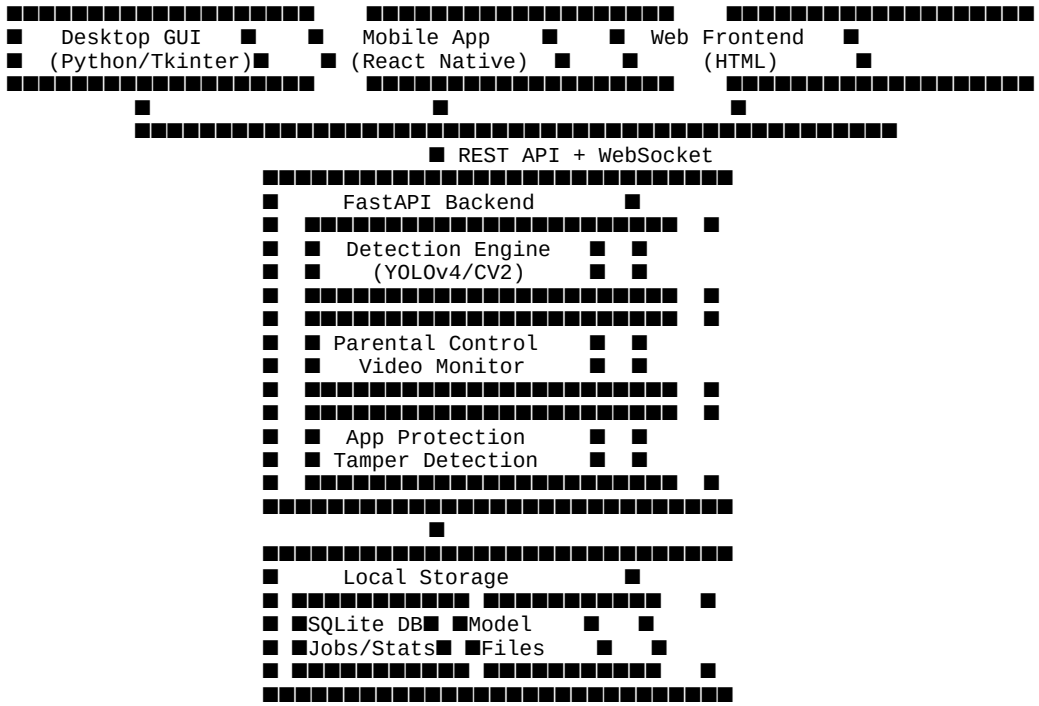
Escvape System Architecture

Overview

Escvape is a comprehensive vaping and smoking detection system that combines AI-powered image/video analysis with parental control and tamper-detection features. The system uses a **client-server architecture** with multiple frontend interfaces connecting to a centralized FastAPI backend.

High-Level Architecture



Core Components

1. Frontend Layer

Desktop Client (`desktop_client.py`)

- **Technology**: Python with Tkinter GUI
- **Features**:
- Image detection interface
- Parental control configuration

- Real-time monitoring controls
- Statistics dashboard
- **Communication**: REST API calls to backend

Mobile App (`mobile-app/App.js`)

- **Technology**: React Native with Expo
- **Features**:
- Camera integration for photo capture
- Gallery image selection
- Monitoring configuration
- Real-time alerts
- **Communication**: REST API + WebSocket for live alerts

Web Frontend (`web_frontend.html`)

- **Technology**: HTML/CSS/JavaScript
- **Features**:
- Browser-based detection interface
- Drag-and-drop image upload
- Real-time WebSocket alerts
- **Communication**: REST API + WebSocket

2. Backend Layer (`api_server.py`)

FastAPI Server

- **Port**: 8000
- **Key Features**:
- RESTful API endpoints
- WebSocket support for real-time alerts
- CORS middleware for cross-origin requests
- Background task processing
- Static file serving

API Endpoints Structure:

```
/detect/*              - Detection endpoints
/self-monitoring/*     - Parental control endpoints
/protection/*          - App protection endpoints
/ws/alerts            - WebSocket for real-time alerts
/                     - Web frontend
```

3. Detection Engine (`main.py`)

SmokingVapingDetector Class

- **Model**: YOLOv4 (You Only Look Once v4)
- **Framework**: OpenCV DNN module
- **Detection Targets**:
- Traditional cigarettes
- Vaping devices (e-cigarettes, vape pens, mods, pods)
- Smoking-related objects

Detection Keywords:

```
['cigarette', 'cigar', 'pipe', 'tobacco', 'smoke',
 'vape', 'vaping', 'e-cigarette', 'vaporizer',
 'mod', 'pod', 'juul', 'puff', 'vapor']
```

Processing Flow:

```
Image Input → Preprocessing → YOLOv4 Inference →
Post-processing → Confidence Filtering → Results
```

4. Parental Control System (`parental_control_api.py`)

Video Monitoring

- **Screen Capture**: Uses PIL ImageGrab for screenshots
- **Detection Frequency**: Configurable interval (default: every few seconds)
- **Monitored Platforms**: YouTube, TikTok, Instagram, Netflix, Safari

Features:

- **Real-time alerts**: WebSocket broadcasts on detection
- **Daily/Weekly reports**: Email summaries (currently disabled)
- **Session tracking**: SQLite database for monitoring sessions
- **Statistics**: Video watch time, detection counts

Monitoring Thread:

```
Screen Capture → Detection →
Confidence Check → Alert Broadcast →
Database Logging → Repeat
```

5. App Protection System (`app_protection.py`)

Security Features:

- **File Integrity Monitoring**: SHA256 hashing of critical files
- **Heartbeat System**: Regular status checks (5-minute intervals)
- **Deletion Detection**: Monitors app directory existence

- **Tamper Alerts**: Notifications on file modifications

Protected Files:

```
['main.py', 'api_server.py', 'parental_control_api.py',
 'desktop_client.py', 'app_protection.py',
 'models/yolov4.weights', 'models/yolov4.cfg']
```

6. Alert System (`alerts.py`)

WebSocket Alert Manager

- **Real-time Broadcasting**: Pushes alerts to all connected clients
- **Thread-safe**: Cross-thread event loop integration
- **Alert Types**: Detection alerts, system status, tamper warnings

Data Layer

SQLite Databases

1. jobs.db - Batch Processing

```
- batch_jobs: Job metadata and status
- job_results: Individual detection results
```

2. monitoring.db - Parental Control

```
- monitoring_sessions: Active/historical sessions
- video_detections: Detection events with timestamps
- daily_stats: Aggregated statistics
```

3. protection.db - App Security

```
- app_status: Heartbeat and protection status
- deletion_alerts: Tampering events
- file_integrity: Hash checksums for critical files
```

Communication Protocols

REST API

- **Format**: JSON
- **Authentication**: None (local deployment)
- **Endpoints**: ~15 endpoints across detection, monitoring, and protection

WebSocket

- **Protocol**: WS (ws://localhost:8000/ws/alerts)
- **Purpose**: Real-time alert broadcasting
- **Message Format**: JSON with detection metadata

Deployment Model

Local Deployment

- **Backend**: Runs on localhost:8000
- **Storage**: Local SQLite databases
- **Processing**: On-device AI inference
- **Privacy**: No cloud uploads, all data stays local

Cross-Platform Support

- **Desktop**: Windows, macOS, Linux
- **Mobile**: iOS, Android (via React Native)
- **Web**: Any modern browser

Key Design Patterns

Modular Architecture: Separate modules for detection, monitoring, and protection

Background Processing: Async tasks for long-running operations

Event-Driven Alerts: WebSocket-based real-time notifications

Database Persistence: SQLite for all state management

Thread Safety: Proper synchronization for concurrent operations

Security & Privacy

- ■ **Local Processing**: All AI inference happens on-device
- ■ **No Cloud Storage**: Images never leave the device
- ■ **File Integrity**: SHA256 checksums for tamper detection
- ■ **Encrypted Storage**: SQLite databases with integrity checks
- ■ **GDPR Compliant**: No external data transmission

Component Interactions

Image Detection Flow

```
User Upload → API Server → Detection Engine →
YOLOv4 Processing → Results → Database → Response
```

Video Monitoring Flow

```
Start Monitoring → Background Thread → Screen Capture →
Detection Engine → Confidence Check → Alert Broadcast →
Database Logging → Repeat
```

App Protection Flow

```
Enable Protection → Calculate File Hashes →
Heartbeat Thread → Periodic Checks →
Detect Changes → Alert Parent → Log Event
```

File Structure

```
escvape/
■■■ main.py                    # Detection engine
■■■ api_server.py              # FastAPI backend
■■■ parental_control_api.py    # Monitoring system
■■■ app_protection.py          # Security system
■■■ alerts.py                  # Alert manager
■■■ desktop_client.py          # Desktop GUI
■■■ web_frontend.html          # Web interface
■■■ mobile-app/
■   ■■■ App.js                 # React Native app
■   ■■■ package.json
■■■ models/
■   ■■■ yolov4.weights         # AI model weights
■   ■■■ yolov4.cfg             # Model configuration
■   ■■■ coco.names             # Class labels
■■■ jobs.db                    # Batch processing DB
■■■ monitoring.db              # Monitoring data DB
■■■ protection.db              # Security data DB
■■■ requirements.txt           # Python dependencies
■■■ README.md                  # Documentation
```

Technology Stack

Backend

- **Python 3.x**: Core language
- **FastAPI**: Web framework
- **OpenCV**: Computer vision
- **YOLOv4**: Object detection
- **SQLite**: Database
- **WebSockets**: Real-time communication

Frontend

- **Tkinter**: Desktop GUI (Python)
- **React Native**: Mobile app
- **Expo**: Mobile development platform
- **HTML/CSS/JS**: Web interface

AI/ML

- **YOLOv4**: Deep learning model
- **COCO Dataset**: Pre-trained weights
- **OpenCV DNN**: Inference engine

Performance Characteristics

Detection Speed

- **Image Analysis**: ~1-3 seconds per image
- **Video Monitoring**: Real-time (configurable interval)
- **Batch Processing**: Parallel processing supported

Resource Usage

- **Model Size**: ~245MB (YOLOv4 weights)
- **Memory**: ~500MB-1GB during inference
- **CPU**: Optimized for CPU inference
- **GPU**: Optional CUDA support

Future Enhancements

Cloud Sync: Optional cloud backup for reports

Advanced Analytics: ML-based trend analysis

Multi-device Support: Centralized parent dashboard

Enhanced Detection: Custom-trained models for vaping devices

Mobile Optimization: On-device ML inference for mobile

Document Version: 1.0

Last Updated: October 25, 2025

System Version: 1.0.0