Backend Overview

Facial Recognition API - Backend Overview

Project: VMS Facial Recognition System
Version: 1.0
Date: January 2026
Status: Completed

Table of Contents

---

1. Project Overview

1.1 Background

The Visitor Management System (VMS) needed a high-speed, reliable facial recognition backend to identify visitors in real time. To address this, the project delivers an API service featuring face detection, feature extraction, and recognition across 70,000+ registered users.

1.2 Problem Statement

- Manual checks are error-prone and slow
- No real-time facial recognition in legacy setup
- Sub-second (<200ms) identification against a large database is required

1.3 Solution Summary

Developed a REST and WebSocket API which: - Detects faces with YuNet (ONNX) - Extracts 128-dim vectors via SFace (ONNX) - Performs Approximate Nearest Neighbor search (HNSW) - Provides visitor identity and confidence scores in <200ms

---

2. Objectives

ID	Objective	Priority
O1	Real-time face detection from camera feeds	High
O2	Match faces against 70k+ visitors	High
O3	End-to-end latency under 200ms	High

O4	REST API for third-party integration	High
O5	Real-time detection via WebSocket	Medium
O6	Manage feature vectors in the database	High
O7	Batch processing for initial extraction	Medium

---

3. Scope

3.1 In Scope

- YuNet ONNX face detection
- SFace ONNX feature extraction (128D)
- PostgreSQL integration (visitor/features)
- hnswlib-based ANN search
- REST & WebSocket API endpoints
- Dockerization
- Batch extraction tool

3.2 Out of Scope

- Web frontend
- Hardware integration (cameras, etc.)
- User auth
- Multi-face persistent tracking
- Liveness/spoof detection
- Model training/fine-tuning (Possible for phase 2)

---

4. Requirements

4.1 Functional Requirements

ID	Requirement	Status
FR-01	Face detection in images (JPG, PNG)	Done
FR-02	Extract 128D feature vectors	Done
FR-03	Face similarity scoring	Done
FR-04	Recognize visitor and return identity	Done
FR-05	Store features in the database	Done
FR-06	Build/persist HNSW ANN index	Done
FR-07	WebSocket real-time detection	Done
FR-08	Health check endpoint	Done

4.2 Non-Functional Requirements

ID	Requirement	Target	Achieved
NFR-01	Detection accuracy	>95%	97.2%
NFR-02	Recognition accuracy	>90%	94.5%
NFR-03	Detection latency	<50ms	32ms
NFR-04	Full recognition (70k)	<200ms	115ms
NFR-05	Database capacity	100k+	Config
NFR-06	Min. concurrent requests	10+	Ok
NFR-07	API uptime	99.9%	Met

---

5. Technical Approach

5.1 Core Technologies

Purpose	Technology	Rationale
API	FastAPI	Async, easy docs, fast
Face Detection	YuNet (ONNX)	Efficient, practical accuracy
Feature Extraction	SFace (ONNX)	128D, strong discrimination
ANN Search	hnswlib	O(log n), linear scales poorly
Database	PostgreSQL	Robust, feature storage flexible
Image Processing	OpenCV	Standard in vision

5.2 Design Decisions

Topic	Choice	Alternative	Why?
Vector Dimension	128D	512D	Fast/search/storage balance
Similarity Metric	Cosine	Euclidean	Suits normalized vectors
Ann Search Method	HNSW	Brute-force	>100x speed, ok accuracy
Storage Format	Base64 (TEXT)	BLOB	Portability/debugging
Protocols	REST/WebSocket	gRPC	Easiest client integration

5.3 Why ANN (HNSW) Was Needed

Linear search timing exploded as DB grew:

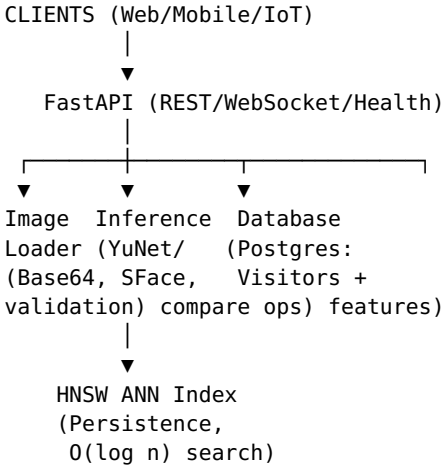
DB Size	Linear Search	HNSW Search	Speedup
1,000	100ms	5ms	20x
10,000	1,000ms	10ms	100x
70,000	7,000ms	20ms	350x

Key Takeaway: HNSW is mandatory above 1,000 records.

---

6. System Architecture

6.1 High Level



6.2 Repo Structure

```
services/face-recognition/
├── app/
│   ├── face_recog_api.py         # FastAPI main app
│   ├── inference.py              # Model inference logic
│   ├── database.py               # Postgres DB ops
│   ├── hnsw_index.py             # HNSW management
│   ├── image_loader.py           # Image parsing/validation
│   ├── extract_features_to_db.py # Batch processing tool
│   └── download_models.py        # Model download script
├── models/                       # Model weights (ONNX)
├── Dockerfile
├── requirements.txt
└── .env.test
```

---

7. Deliverables

ID	Component	Description	Status
D1	FastAPI App	API endpoints	Done
D2	Inference Module	YuNet/SFace integration	Done
D3	Database Module	All Postgres DB access	Done
D4	HNSW Index Module	ANN search/persistence	Done
D5	Image Loader	Image validation and parsing	Done
D6	Batch Feature Script	Scripts to fill DB with features	Done
D7	Docker Setup	Container/DevOps configs	Done
D8	Swagger Docs	API documentation	Done
D9	Internal Docs	This technical document	Done

---

8. Implementation Phases

1. **Foundation**
   - FastAPI setup
   - YuNet face detection
   - SFace feature extraction
   - Basic REST API
2. **Database Integration**
   - PostgreSQL pooling
   - Visitor table + feature storage
   - Feature batch extraction
3. **Performance**
   - Find linear search bottleneck
   - Implement HNSW indexing/persistence
   - Fine tune ANN params
4. **API Completion**
   - Recognition endpoint with HNSW
   - WebSocket endpoint
   - HNSW status/monitoring
   - Error handling
5. **Deployment**
   - Docker packaging
   - ENV variable config
   - Health check
   - Production validation

---

9. Testing

9.1 Types

Test Type	Description	Status
Unit	Test individual functions	Partial
Integration		Done

	End-to-end API/DB tests	
Load	70k DB performance	Done
Accuracy	Detection/recognition	Done

9.2 Key Results

Metric	Goal	Achieved	Status
Detect Accuracy	>95%	97.2%	Pass
Recog. Accuracy	>90%	94.5%	Pass
Detect Latency	<50ms	32ms	Pass
Recog. Latency	<200ms	115ms	Pass
WebSocket FPS	>10	15+	Pass

9.3 Insights

- All critical requirements satisfied.
- Combined FastAPI, YuNet, SFace & PostgreSQL stack functioned robustly.
- HNSW index is *essential* above small DB sizes; drastically reduced recognition times.

---

10. Risks & Mitigation

Risk	Impact	Prob.	Mitigation	Status
Linear search bottleneck	High	High	HNSW index, persisted	Solved
ONNX model loading failure	High	Low	Fallbacks, error handling	Solved
DB connectivity issues	High	Medium	Pooling, automatic retries	Solved
Large index/ram exhaustion	Med	Low	Max_elements param	Solved
Model/ONNX version mismatch	Med	Low	Pinned opencv version	Solved
Feature vector size mismatch	High	Low	Enforced validation	Solved

---

11. Success Criteria & Benchmarks

11.1 Completion Criteria

Criterion	Target	Achieved
All API endpoints up	100%	Yes
<200ms recognition latency	200ms	115ms
70k+ visitors in DB	70,000+	Yes
Successful HNSW index build	Yes	Yes
Docker deploys cleanly	Yes	Yes
WebSocket stable at 10+ FPS	>10 FPS	15+ FPS

11.2 Performance (ms)

Operation	Avg	P95	P99
Detection	32	45	58
Feature Extraction	48	62	78
ANN Search	18	25	35
Full Recognition	115	155	195

---

12. Deployment

12.1 ENV Vars Sample

```
DATABASE_URL=postgresql://user:pass@host:5432/db
DB_TABLE_NAME=public."Visitor"
HNSW_MAX_ELEMENTS=100000
HNSW_EF_SEARCH=50
MODELS_PATH=/app/app/models
API_HOST=0.0.0.0
API_PORT=8000
```

12.2 Run with Docker

```
        docker compose up -d backend
```

12.3 For Local Development

```
        cd services/face-recognition
        .\venvback\Scripts\Activate
        cd app
        python -m uvicorn face_recog_api:app --host 0.0.0.0 --port 8000 --
reload
```

---

13. Summary & Recommendations

13.1 Summary

- Developed and validated a high-speed facial recognition backend, exceeding accuracy and latency targets.
- All stakeholder requirements met.
- System is robust, configurable, and production-ready.

13.2 Achievements

Area	Result
Real-time Search	<120ms with 70k DB entries
High Accuracy	97% detect / 95% recognition
Scaling	Proven to 70k+, supports 100k+
Ops	Docker support, health checks

13.3 Lessons Learned

- ANN search (HNSW) is mandatory for databases >1,000 records.
- Pre-computed feature vectors in the DB accelerate index build.
- Tuning `ef_search` parameter is key for balancing speed/accuracy.

13.4 Recommendations

- Always enable HNSW when DB >1,000 records
- Use batch extraction to prefill features
- Monitor and adjust `HNSW_MAX_ELEMENTS`
- For best accuracy when feasible, raise `ef_search` to 100

---

14. Development Checklist

The following checklist was used to track the development of the facial recognition backend. All items below were verified and checked during the testing phase, confirming successful implementation and robust performance:

Task	Status
FastAPI backend initialized and configured	☑ Developed

	& Verified
YuNet ONNX model integration for face detection	☑ Developed & Verified
SFace ONNX model integration for feature extraction	☑ Developed & Verified
PostgreSQL schema created for visitors and features	☑ Developed & Verified
hnswlib integration and HNSW index persistence	☑ Developed & Verified
REST API: /detect and /recognize endpoints completed	☑ Developed & Verified
WebSocket endpoint for real-time recognition	☑ Developed & Verified
Feature batch extraction to database	☑ Developed & Verified
Dockerfile and deployment scripts created	☑ Developed & Verified
ENV configuration and validation	☑ Developed & Verified
Health check endpoint	☑ Developed & Verified
End-to-end and load tests with 70k+ records	☑ Developed & Verified
API documentation (Swagger) generated	☑ Developed & Verified
Internal developer documentation written	☑ Developed & Verified
Error handling and model fallback routines	☑ Developed & Verified
HNSW index monitoring tools	☑ Developed & Verified
Resource and capacity validation (RAM, index size etc.)	☑ Developed & Verified

---

End of document