Scentinel: Installation Guide

> **Name: Divyansh Jha**

> **Mail Id: jhadivyansh29@gmail.com**

> **Contact No: 9810739995**

```
https://github.com/itsdivyanshjha/Scentinel.git
```

1. TECHNICAL STACK OVERVIEW

Frontend Technologies

- **Next.js 13+**: React-based framework with App Router and server-side rendering
- **TypeScript 5.0+**: Type-safe JavaScript for enhanced development experience
- **Tailwind CSS 3.3+**: Utility-first CSS framework for responsive design
- **React 18+**: Component-based UI library with concurrent features
- **Axios**: HTTP client for API communication
- **React Hook Form**: Performance-optimized form handling
- **React DnD**: Drag-and-drop functionality for perfume ranking

Backend Technologies

- **Flask 2.3+**: Lightweight Python web framework
- **Python 3.10+**: Core programming language
- **PyTorch 2.0+**: Deep learning framework for neural network models
- **scikit-learn 1.3+**: Machine learning utilities and preprocessing
- **gensim 4.3+**: Word2Vec embeddings and NLP processing
- **Flask-JWT-Extended**: JSON Web Token authentication
- **Flask-CORS**: Cross-Origin Resource Sharing support
- **Werkzeug**: Password hashing and security utilities

Database and Storage

- **MongoDB 6.0+**: NoSQL document database
- **pymongo 4.4+**: Python MongoDB driver
- **MongoDB Compass**: Optional GUI for database management

DevOps and Deployment

- **Docker 24.0+**: Containerization platform
- **Docker Compose 2.0+**: Multi-container orchestration
- **Git 2.40+**: Version control system

Machine Learning Stack

- **NumPy 1.24+**: Numerical computing library
- **Pandas 2.0+**: Data manipulation and analysis
- **Matplotlib 3.7+**: Data visualization (optional)
- **Pickle**: Model serialization and storage

---

2. SYSTEM REQUIREMENTS

Minimum Requirements

- **Operating System**: Windows 10 (20H2), macOS 11.0 (Big Sur), Ubuntu 20.04 LTS
- **RAM**: 8GB (4GB available for Docker)
- **Storage**: 10GB free disk space
- **CPU**: Dual-core processor (x64 architecture)
- **Network**: Stable internet connection for initial setup
- **Ports**: 3000, 5000, 27017 available

Recommended Requirements

- **RAM**: 16GB (8GB available for Docker)
- **Storage**: 20GB SSD storage
- **CPU**: Quad-core processor with 3.0GHz+ base frequency
- **Network**: Broadband connection (10+ Mbps)
- **Additional**: GPU support for enhanced ML training (optional)

Performance Targets

- **Container Startup**: < 2 minutes for full system
- **Database Initialization**: < 30 seconds
- **Recommendation Generation**: < 500ms
- **Frontend Load Time**: < 3 seconds
- **Concurrent Users**: 50+ (single instance)

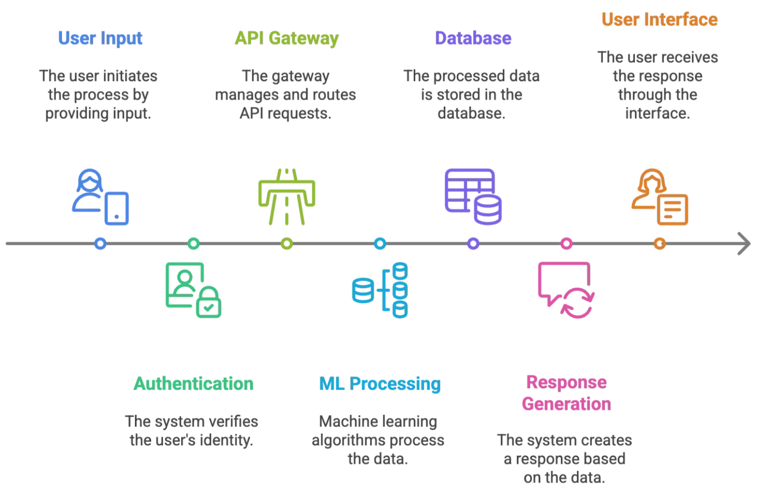
---

3. ARCHITECTURE OVERVIEW

Architecture Principles

- **Microservices Design**: Loosely coupled components for scalability
- **Containerized Deployment**: Docker containers for consistency and portability
- **API-First Approach**: RESTful API design for frontend-backend communication
- **Stateless Authentication**: JWT-based authentication for scalability
- **Modular ML Pipeline**: Pluggable machine learning models

Data Flow Architecture


User Input
The user initiates the process by providing input.
API Gateway
The gateway manages and routes API requests.
Database
The processed data is stored in the database.
User Interface
The user receives the response through the interface.
Authentication
The system verifies the user's identity.
ML Processing
Machine learning algorithms process the data.
Response Generation
The system creates a response based on the data.

4. MAJOR COMPONENTS

1. Frontend Application (scentinel-frontend)

Technology: Next.js with TypeScript

Purpose: User interface for perfume ranking and recommendation viewing

Key Features:

- Responsive design supporting desktop and mobile devices
- Drag-and-drop perfume ranking interface
- Real-time recommendation updates
- JWT-based authentication with automatic token refresh
- Dark/light theme support
- Progressive Web App (PWA) capabilities

Directory Structure:

```
frontend/
├── pages/          # Next.js pages and routing
├── components/     # Reusable React components
├── styles/         # CSS and Tailwind configurations
├── contexts/       # React context providers
├── hooks/          # Custom React hooks
├── utils/          # Utility functions
├── public/         # Static assets
└── types/          # TypeScript type definitions
```

2. Backend API Server (scentinel-backend)

Technology: Flask with Python

Purpose: API server, ML processing, and business logic

Key Features:

- RESTful API with comprehensive endpoint coverage
- Three-model ML ensemble (RankNet, DPL, BPR)
- Word2Vec embedding generation and caching
- Secure authentication with password hashing
- Real-time recommendation generation
- Comprehensive error handling and logging

Directory Structure:

```
backend/
├── app/
│   ├── routes/         # API endpoint definitions
│   ├── services/       # Business logic services
│   ├── models/         # ML model implementations
│   ├── utils/          # Utility functions
│   └── data/           # Data processing and storage
├── requirements.txt    # Python dependencies
├── run.py              # Application entry point
└── Dockerfile          # Container configuration
```

3. Database System (scentinel-mongodb)

Technology: MongoDB

Purpose: Data persistence and retrieval

Key Features:

- Document-based storage for flexible schema
- Optimized indexes for recommendation queries
- Aggregation pipelines for complex data processing
- Automatic data validation and constraints
- Backup and recovery capabilities

Collections:

- **users**: User accounts and authentication data
- **perfumes**: Comprehensive perfume database
- **rankings**: User preference rankings
- **recommendations**: Generated recommendation cache

4. Machine Learning Pipeline

Components: RankNet, DPL, BPR models

Purpose: Personalized recommendation generation

Key Features:

- Ensemble model approach for robust predictions

- Dynamic model weighting based on data availability
- Pre-training for cold-start recommendations
- Real-time model fine-tuning
- Performance monitoring and evaluation

5. Containerization Infrastructure

Technology: Docker and Docker Compose

Purpose: Consistent deployment across environments

Key Features:

- Multi-stage builds for optimized container sizes
- Health checks for service monitoring
- Volume mounting for persistent data storage
- Network isolation and security
- Scalable service orchestration

---

5. PRE-INSTALLATION SETUP

1. System Preparation

Windows Setup:

```
# Enable Windows Subsystem for Linux (WSL2) - Recommended
wsl --install

# Enable Hyper-V (Alternative to WSL2)
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V-All

# Update Windows to latest version
# Settings → Update & Security → Windows Update
```

macOS Setup:

```
# Install Xcode Command Line Tools
xcode-select --install

# Install Homebrew (if not already installed)
/bin/bash -c "$(curl -fsSL <https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh>)"
```

Linux (Ubuntu) Setup:

```
# Update package repositories
sudo apt update && sudo apt upgrade -y

# Install essential packages
sudo apt install -y curl wget git software-properties-common
```

2. Docker Installation

Windows (Docker Desktop):

1. Download Docker Desktop from https://docs.docker.com/desktop/windows/install/
2. Run installer with administrator privileges
3. Choose WSL2 backend during installation
4. Restart computer when prompted
5. Launch Docker Desktop and complete setup

macOS (Docker Desktop):

```
# Option 1: Download from website
# Visit <https://docs.docker.com/desktop/mac/install/>

# Option 2: Install via Homebrew
brew install --cask docker

# Launch Docker Desktop from Applications
open /Applications/Docker.app
```

Linux (Ubuntu):

```
# Add Docker's official GPG key
curl -fsSL <https://download.docker.com/linux/ubuntu/gpg> | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

# Add Docker repository
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] <https://download.docker.com/linux/ubuntu> $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Install Docker Engine
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin

# Add user to docker group
sudo usermod -aG docker $USER

# Start Docker service
sudo systemctl start docker
sudo systemctl enable docker

# Log out and back in for group changes to take effect
```

3. Git Installation and Configuration

Windows:

```
# Download and install from <https://git-scm.com/download/win>
# Or install via chocolatey
choco install git
```

```
# Configure Git
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

macOS:

```
# Install via Homebrew
brew install git

# Configure Git
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

Linux:

```
# Install Git
sudo apt install -y git

# Configure Git
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

---

6. INSTALLATION METHODS

Method 1: Quick Start (Recommended)

Best for: Users wanting immediate deployment

Time Required: 10-15 minutes

Prerequisites: Docker, Docker Compose, Git

```
# Clone repository
git clone <https://github.com/itsdivyanshjha/Scentinel.git>
cd Scentinel

# Start all services
docker-compose up --build

# Initialize database (in new terminal)
docker-compose exec backend python init_db.py
```

Method 2: Development Setup

Best for: Developers and contributors

Time Required: 30-45 minutes

Prerequisites: Docker, Node.js, Python, MongoDB

```
# Clone repository
git clone <https://github.com/itsdivyanshjha/Scentinel.git>
cd Scentinel

# Backend setup
cd backend
python -m venv venv
source venv/bin/activate  # Windows: venv\\Scripts\\activate
pip install -r requirements.txt

# Frontend setup
cd ../frontend
npm install

# Start services individually
# Terminal 1: MongoDB
docker run -d -p 27017:27017 --name scentinel-mongo mongo:latest

# Terminal 2: Backend
cd backend && python run.py

# Terminal 3: Frontend
cd frontend && npm run dev
```

Method 3: Production Deployment

Best for: Server deployment and scaling

Time Required: 45-60 minutes

Prerequisites: Docker, Docker Compose, Reverse Proxy

```
# Clone repository
git clone <https://github.com/itsdivyanshjha/Scentinel.git>
cd Scentinel

# Create production environment file
cp .env.example .env.production

# Edit production settings
nano .env.production

# Deploy with production configuration
docker-compose -f docker-compose.prod.yml up --build -d

# Set up reverse proxy (Nginx example)
sudo apt install nginx
# Configure nginx with provided configuration
```

7. OPERATING SYSTEM SPECIFIC INSTRUCTIONS

Windows Installation

Prerequisites Installation:

```
# Install Chocolatey package manager (optional but recommended)
Set-ExecutionPolicy Bypass -Scope Process -Force
[System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityP
rotocol -bor 3072
iex ((New-Object System.Net.WebClient).DownloadString('<https://community.chocolatey.org/install.
ps1>'))

# Install required software via Chocolatey
choco install git docker-desktop -y

# Alternative: Manual installation
# Git: <https://git-scm.com/download/win>
# Docker Desktop: <https://docs.docker.com/desktop/windows/install/>
```

Scentinel Installation:

```
# Open PowerShell as Administrator
# Clone repository
git clone <https://github.com/itsdivyanshjha/Scentinel.git>
cd Scentinel

# Verify Docker is running
docker --version
docker-compose --version

# Build and start services
docker-compose up --build

# In new PowerShell window, initialize database
docker-compose exec backend python init_db.py
```

Windows-Specific Configuration:

```
# Configure Windows Defender firewall (if needed)
New-NetFirewallRule -DisplayName "Scentinel Frontend" -Direction Inbound -Port 3000 -Protocol TC
P -Action Allow
New-NetFirewallRule -DisplayName "Scentinel Backend" -Direction Inbound -Port 5000 -Protocol TC
P -Action Allow

# Set Docker memory limit (Docker Desktop Settings)
# Recommended: 4GB minimum, 8GB optimal
```

macOS Installation

Prerequisites Installation:

```
# Install Xcode Command Line Tools
xcode-select --install

# Install Homebrew
/bin/bash -c "$(curl -fsSL <https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh>)"

# Install required software
brew install git
brew install --cask docker

# Start Docker Desktop
open /Applications/Docker.app
```

Scentinel Installation:

```
# Open Terminal
# Clone repository
git clone <https://github.com/itsdivyanshjha/Scentinel.git>
cd Scentinel

# Verify Docker installation
docker --version
docker-compose --version

# Build and start services
docker-compose up --build

# In new terminal tab, initialize database
docker-compose exec backend python init_db.py
```

macOS-Specific Configuration:

```
# Configure Docker resource limits (Docker Desktop Preferences)
# Memory: 4GB minimum, 8GB recommended
# CPU: 2 cores minimum, 4 cores recommended

# Add Docker to PATH (if needed)
echo 'export PATH="/Applications/Docker.app/Contents/Resources/bin:$PATH"' >> ~/.zshrc
source ~/.zshrc
```

Linux (Ubuntu) Installation

Complete Setup Script:

```
#!/bin/bash
# Scentinel Ubuntu Installation Script

# Update system
```

```
sudo apt update && sudo apt upgrade -y

# Install prerequisites
sudo apt install -y curl wget git software-properties-common apt-transport-https ca-certificates gnup
g lsb-release

# Install Docker
curl -fsSL <https://download.docker.com/linux/ubuntu/gpg> | sudo gpg --dearmor -o /usr/share/keyri
ngs/docker-archive-keyring.gpg
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] <https://downlo
ad.docker.com/linux/ubuntu> $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin

# Configure Docker
sudo usermod -aG docker $USER
sudo systemctl start docker
sudo systemctl enable docker

# Install Docker Compose (if not included)
sudo curl -L "<https://github.com/docker/compose/releases/latest/download/docker-compose-$>(un
ame -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose

# Clone and setup Scentinel
git clone <https://github.com/itsdivyanshjha/Scentinel.git>
cd Scentinel

# Note: Log out and back in, then run:
echo "Please log out and back in, then run:"
echo "cd Scentinel && docker-compose up --build"
```

Manual Installation:

```
# Step-by-step installation
sudo apt update
sudo apt install -y git curl

# Install Docker (see Docker installation section above)

# Clone repository
git clone <https://github.com/itsdivyanshjha/Scentinel.git>
cd Scentinel

# Start services
docker-compose up --build

# Initialize database (new terminal)
docker-compose exec backend python init_db.py
```

Linux-Specific Configuration:

```
# Configure firewall (UFW)
sudo ufw allow 3000/tcp  # Frontend
sudo ufw allow 5000/tcp  # Backend (optional for external access)

# Set up log rotation for Docker containers
sudo nano /etc/logrotate.d/docker-containers
# Add configuration for log management

# Configure system limits for Docker
echo 'vm.max_map_count=262144' | sudo tee -a /etc/sysctl.conf
sudo sysctl -p
```

8. ADDITIONAL SOFTWARE REQUIREMENTS

Essential Tools

1. **Web Browser**: Chrome, Firefox, Safari, or Edge (latest versions)
2. **Text Editor**: VS Code, Sublime Text, or Atom (for configuration editing)
3. **Terminal**: Command line interface for your operating system

Development Tools (Optional)

```
# Node.js (for frontend development)
# Windows: Download from nodejs.org
# macOS: brew install node
# Linux: sudo apt install nodejs npm

# Python (for backend development)
# Windows: Download from python.org
# macOS: brew install python@3.10
# Linux: sudo apt install python3.10 python3.10-venv

# MongoDB Compass (database GUI)
# Download from mongodb.com/products/compass
```

Monitoring and Management Tools

```
# Docker Desktop Dashboard (included with Docker Desktop)
# Portainer (web-based Docker management)
docker run -d -p 9000:9000 --name portainer --restart always -v /var/run/docker.sock:/var/run/dock
er.sock portainer/portainer-ce

# ctop (command-line container monitoring)
# Linux/macOS: Available via package managers
# Windows: Download binary from github.com/bcicen/ctop
```

9. CONFIGURATION AND ENVIRONMENT SETUP

Environment Variables Configuration

Backend Configuration (.env file):

```
# Create backend/.env file
cd backend
cat > .env << EOF
# Flask Configuration
FLASK_ENV=development
FLASK_DEBUG=True
SECRET_KEY=your-secret-key-here-change-in-production

# Database Configuration
MONGO_URI=mongodb://scentinel-mongodb:27017/scentinel
MONGO_DB_NAME=scentinel

# JWT Configuration
JWT_SECRET_KEY=your-jwt-secret-key-here
JWT_ACCESS_TOKEN_EXPIRES=86400

# ML Model Configuration
MODEL_CACHE_DIR=app/data/models
EMBEDDING_CACHE_DIR=app/data/embeddings

# Performance Configuration
RECOMMENDATION_CACHE_TIMEOUT=3600
MAX_RECOMMENDATIONS=50

# Logging Configuration
LOG_LEVEL=INFO
LOG_FILE=app.log
EOF
```

Frontend Configuration (.env.local file):

```
# Create frontend/.env.local file
cd frontend
cat > .env.local << EOF
# API Configuration
NEXT_PUBLIC_API_URL=http://localhost:5000
NEXT_PUBLIC_FRONTEND_URL=http://localhost:3000

# Application Configuration
NEXT_PUBLIC_APP_NAME=Scentinel
NEXT_PUBLIC_APP_VERSION=1.0.0

# Feature Flags
NEXT_PUBLIC_ENABLE_ANALYTICS=false
NEXT_PUBLIC_ENABLE_DARK_MODE=true
```

```
# Performance Configuration
NEXT_PUBLIC_RECOMMENDATION_REFRESH_INTERVAL=30000
NEXT_PUBLIC_API_TIMEOUT=10000
EOF
```

Docker Compose Configuration

Development Configuration (docker-compose.yml):

```
version: '3.8'

services:
  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    environment:
      - NEXT_PUBLIC_API_URL=http://localhost:5000
    depends_on:
      - backend
    volumes:
      - ./frontend:/app
      - /app/node_modules

  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    ports:
      - "5000:5000"
    environment:
      - FLASK_ENV=development
      - MONGO_URI=mongodb://scentinel-mongodb:27017/scentinel
    depends_on:
      - scentinel-mongodb
    volumes:
      - ./backend:/app
      - ./data/models:/app/app/data/models

  scentinel-mongodb:
    image: mongo:6.0
    ports:
      - "27017:27017"
    volumes:
      - mongodb_data:/data/db
      - ./data/mongo-init:/docker-entrypoint-initdb.d

volumes:
  mongodb_data:

networks:
  default:
    name: scentinel-network
```

Database Configuration

MongoDB Initialization Script:

```
// Create data/mongo-init/init.js
db = db.getSiblingDB('scentinel');

// Create collections with validation
db.createCollection('users', {
  validator: {
    $jsonSchema: {
      bsonType: 'object',
      required: ['email', 'password'],
      properties: {
        email: { bsonType: 'string', pattern: '^.+@.+$' },
        password: { bsonType: 'string', minLength: 6 }
      }
    }
  }
});

// Create indexes for performance
db.users.createIndex({ email: 1 }, { unique: true });
db.perfumes.createIndex({ Brand: 1 });
db.perfumes.createIndex({ Gender: 1 });
db.rankings.createIndex({ user_id: 1, perfume_id: 1 }, { unique: true });
db.recommendations.createIndex({ user_id: 1, created_at: -1 });

print('Database initialized successfully');
```

10. VERIFICATION AND TESTING

System Health Checks

1. Docker Services Verification:

```
# Check all services are running
docker-compose ps

# Expected output:
# scentinel-frontend    running    0.0.0.0:3000->3000/tcp
# scentinel-backend     running    0.0.0.0:5000->5000/tcp
# scentinel-mongodb     running    0.0.0.0:27017->27017/tcp
# Check service logs
docker-compose logs frontend
docker-compose logs backend
docker-compose logs scentinel-mongodb
```

2. API Endpoint Testing:

```
# Test backend health endpoint
curl <http://localhost:5000/health>
# Expected: {"status": "healthy", "timestamp": "..."}

# Test database connection
curl <http://localhost:5000/api/health/db>
# Expected: {"database": "connected", "collections": 4}

# Test cold-start recommendations
curl <http://localhost:5000/api/recommend/cold-start?limit=5>
# Expected: JSON array with 5 perfume recommendations
```

3. Frontend Accessibility:

```
# Test frontend loading
curl -I <http://localhost:3000>
# Expected: HTTP/1.1 200 OK

# Check if frontend can reach backend
# Visit <http://localhost:3000> in browser
# Verify login/register pages load correctly
```

4. Database Verification:

```
# Connect to MongoDB container
docker-compose exec scentinel-mongodb mongosh scentinel

# Verify collections exist
show collections
# Expected: users, perfumes, rankings, recommendations

# Check sample data
db.perfumes.countDocuments()
# Expected: > 0 (number of perfumes in dataset)
```

Automated Testing Suite

Backend API Tests:

```
# Create tests/test_api.py
import requests
import json
```

```
BASE_URL = "<http://localhost:5000>"

def test_health_endpoint():
    response = requests.get(f"{BASE_URL}/health")
    assert response.status_code == 200
    assert "status" in response.json()

def test_cold_start_recommendations():
    response = requests.get(f"{BASE_URL}/api/recommend/cold-start")
    assert response.status_code == 200
    data = response.json()
    assert isinstance(data, list)
    assert len(data) > 0

def test_user_registration():
    user_data = {
        "email": "test@example.com",
        "password": "testpassword123"
    }
    response = requests.post(f"{BASE_URL}/api/auth/register", json=user_data)
    assert response.status_code in [201, 409]  # 409 if user already exists

if __name__ == "__main__":
    test_health_endpoint()
    test_cold_start_recommendations()
    test_user_registration()
    print("All tests passed!")
```

Frontend Component Tests:

```
# Run frontend tests (if available)
cd frontend
npm test

# Build verification
npm run build
# Should complete without errors
```

Performance Benchmarking

Response Time Testing:

```
# Install Apache Bench (ab) for load testing
# Ubuntu: sudo apt install apache2-utils
# macOS: brew install httpie
# Windows: Download from Apache website

# Test recommendation endpoint performance
ab -n 100 -c 10 <http://localhost:5000/api/recommend/cold-start>
```

```
# Expected results:
# - Mean response time: < 500ms
# - 95th percentile: < 1000ms
# - No failed requests
```

Resource Usage Monitoring:

```
# Monitor Docker container resources
docker stats

# Expected resource usage:
# Frontend: < 100MB RAM, < 5% CPU
# Backend: < 500MB RAM, < 20% CPU
# MongoDB: < 200MB RAM, < 10% CPU
```

---

11. TROUBLESHOOTING

Common Installation Issues

Issue 1: Docker Not Starting

Symptoms: "Docker daemon not running" error

Solutions:

```
# Windows
# Start Docker Desktop application
# Check Windows Subsystem for Linux (WSL2) is enabled

# macOS
# Start Docker Desktop from Applications folder
sudo docker start

# Linux
sudo systemctl start docker
sudo systemctl enable docker

# Check Docker status
docker info
```

Issue 2: Port Conflicts

Symptoms: "Port already in use" error

Solutions:

```
# Find processes using ports
# Windows
netstat -ano | findstr :3000
netstat -ano | findstr :5000
netstat -ano | findstr :27017

# macOS/Linux
lsof -i :3000
lsof -i :5000
lsof -i :27017

# Kill conflicting processes or change ports in docker-compose.yml
```

Issue 3: Container Build Failures

Symptoms: Docker build errors

Solutions:

```
# Clear Docker cache
docker system prune -a

# Rebuild with no cache
docker-compose build --no-cache

# Check Dockerfile syntax
docker build -t test-build ./backend
docker build -t test-build ./frontend
```

Issue 4: Database Connection Issues

Symptoms: Backend cannot connect to MongoDB

Solutions:

```
# Check MongoDB container status
docker-compose logs scentinel-mongodb

# Verify network connectivity
docker network ls
docker network inspect scentinel-network

# Test manual connection
docker-compose exec backend python -c "from app import mongo; print(mongo.db.list_collection_na
mes())"
```

Issue 5: Frontend Build Errors

Symptoms: Next.js build failures

Solutions:

```
# Clear Next.js cache
cd frontend
rm -rf .next node_modules package-lock.json
```

```
# Reinstall dependencies
npm install

# Check Node.js version
node --version
# Should be 16.0.0 or higher
```

Performance Issues

Issue 1: Slow Recommendation Generation

Symptoms: API responses > 1 second

Diagnostics:

```
# Check backend logs
docker-compose logs backend | grep "recommendation"

# Monitor CPU and memory usage
docker stats scentinel-backend

# Profile API endpoint
curl -w "@curl-format.txt" <http://localhost:5000/api/recommend/cold-start>
```

Solutions:

- Increase Docker memory allocation
- Enable model caching
- Optimize database queries

Issue 2: High Memory Usage

Symptoms: System slowdown, container crashes

Solutions:

```
# Increase Docker memory limits
# Docker Desktop → Settings → Resources → Memory

# Monitor container memory usage
docker stats --format "table {{.Container}}\\t{{.CPUPerc}}\\t{{.MemUsage}}"

# Optimize backend configuration
# Reduce model cache size
# Enable garbage collection
```

Network and Connectivity Issues

Issue 1: CORS Errors

Symptoms: Frontend cannot reach backend API

Solutions:

```
# Check backend CORS configuration
# In backend/app/__init__.py
from flask_cors import CORS
CORS(app, origins=['<http://localhost:3000>'])
```

Issue 2: JWT Token Issues

Symptoms: Authentication failures

Solutions:

```
# Check JWT configuration
# Verify SECRET_KEY is set in backend/.env
# Check token expiration settings

# Test token generation
curl -X POST <http://localhost:5000/api/auth/login> \\
  -H "Content-Type: application/json" \\
  -d '{"email":"test@example.com","password":"testpass"}'
```

Database Issues

Issue 1: Missing Collections

Symptoms: "Collection not found" errors

Solutions:

```
# Reinitialize database
docker-compose exec backend python init_db.py

# Check collection creation
docker-compose exec scentinel-mongodb mongosh scentinel
show collections
```

Issue 2: Index Creation Failures

Symptoms: Slow database queries

Solutions:

```
// Connect to MongoDB and create indexes manually
use scentinel;
db.users.createIndex({email: 1}, {unique: true});
db.perfumes.createIndex({Brand: 1});
db.rankings.createIndex({user_id: 1, perfume_id: 1});
```

12. PERFORMANCE OPTIMIZATION

Docker Optimization

1. Container Resource Allocation:

```
# docker-compose.override.yml
version: '3.8'

services:
  frontend:
    deploy:
      resources:
        limits:
          cpus: '1.0'
          memory: 512M
        reservations:
          cpus: '0.5'
          memory: 256M

  backend:
    deploy:
      resources:
        limits:
          cpus: '2.0'
          memory: 2G
        reservations:
          cpus: '1.0'
          memory: 1G

  scentinel-mongodb:
    deploy:
      resources:
        limits:
          cpus: '1.0'
          memory: 1G
        reservations:
          cpus: '0.5'
          memory: 512M
```

2. Docker Image Optimization:

```
# Multi-stage build for smaller images
# backend/Dockerfile
FROM python:3.10-slim as builder
WORKDIR /app
COPY requirements.txt .
RUN pip install --user -r requirements.txt

FROM python:3.10-slim
WORKDIR /app
COPY --from=builder /root/.local /root/.local
COPY . .
```

```
ENV PATH=/root/.local/bin:$PATH
CMD ["python", "run.py"]
```

Application Performance Tuning

1. Backend Optimization:

```
# app/config.py
class Config:
    # Enable response caching
    CACHE_TYPE = "simple"
    CACHE_DEFAULT_TIMEOUT = 3600

    # Database connection pooling
    MONGO_POOL_SIZE = 10
    MONGO_MAX_POOL_SIZE = 50

    # ML model optimization
    MODEL_BATCH_SIZE = 32
    EMBEDDING_CACHE_SIZE = 1000

    # API rate limiting
    RATELIMIT_DEFAULT = "1000 per hour"
```

2. Frontend Optimization:

```
// next.config.js
const nextConfig = {
  // Enable compression
  compress: true,

  // Optimize images
  images: {
    domains: ['localhost'],
    formats: ['image/webp', 'image/avif'],
  },

  // Enable experimental features
  experimental: {
    optimizeCss: true,
    optimizeImages: true,
  },

  // Production optimizations
  swcMinify: true,
  poweredByHeader: false,
}

module.exports = nextConfig
```

3. Database Optimization:

```
// MongoDB optimization script
// Run in MongoDB shell
use scentinel;

// Create compound indexes for common queries
db.rankings.createIndex({user_id: 1, created_at: -1});
db.recommendations.createIndex({user_id: 1, score: -1});
db.perfumes.createIndex({Brand: 1, Gender: 1});

// Enable profiler for slow queries
db.setProfilingLevel(2, {slowms: 100});

// Analyze collection statistics
db.perfumes.stats();
db.rankings.stats();
```

Monitoring and Metrics

1. Application Monitoring:

```
# backend/app/monitoring.py
import time
import logging
from functools import wraps

def monitor_performance(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()

        logging.info(f"{func.__name__} took {end_time - start_time:.3f} seconds")
        return result
    return wrapper

# Apply to recommendation functions
@monitor_performance
def generate_recommendations(user_id):
    # Function implementation
    pass
```

2. Health Check Endpoints:

```
# backend/app/routes/health.py
@app.route('/health')
def health_check():
    return {
        'status': 'healthy',
        'timestamp': datetime.utcnow().isoformat(),
        'version': '1.0.0'
    }

@app.route('/health/detailed')
def detailed_health():
    return {
        'database': check_database_connection(),
        'models': check_ml_models(),
        'memory_usage': get_memory_usage(),
        'response_time': measure_api_response_time()
    }
```

13. MAINTENANCE AND UPDATES

Regular Maintenance Tasks

1. Daily Maintenance:

```
#!/bin/bash
# daily_maintenance.sh

# Check container health
docker-compose ps

# Clean up unused Docker resources
docker system prune -f

# Backup database
docker-compose exec scentinel-mongodb mongodump --out /backup/$(date +%Y%m%d)

# Check disk usage
df -h

# Monitor logs
docker-compose logs --tail=100 backend | grep ERROR
```

2. Weekly Maintenance:

```
#!/bin/bash
# weekly_maintenance.sh

# Update Docker images
docker-compose pull

# Restart services for fresh state
docker-compose restart

# Analyze database performance
```

```
docker-compose exec scentinel-mongodb mongosh --eval "db.runCommand({dbStats: 1})"

# Clean old log files
find ./logs -name "*.log" -mtime +7 -delete

# Generate performance report
python scripts/performance_report.py
```

3. Monthly Maintenance:

```
#!/bin/bash
# monthly_maintenance.sh

# Full system backup
tar -czf backup/scentinel_$(date +%Y%m%d).tar.gz data/ backend/app/data/

# Update dependencies
cd frontend && npm audit fix
cd ../backend && pip-review --local --auto

# Retrain models with latest data
python backend/scripts/retrain_models.py

# Security scan
docker scout cves scentinel-backend
docker scout cves scentinel-frontend
```

Update Procedures

1. Application Updates:

```
# Create backup before updates
docker-compose exec scentinel-mongodb mongodump --out /backup/pre_update

# Pull latest code
git pull origin main

# Update dependencies
cd frontend && npm install
cd ../backend && pip install -r requirements.txt

# Rebuild containers
docker-compose build --no-cache

# Deploy with zero downtime
docker-compose up -d --force-recreate
```

2. Security Updates:

```
# Update base images
docker pull node:18-alpine
docker pull python:3.10-slim
docker pull mongo:6.0

# Rebuild with updated base images
docker-compose build --pull --no-cache

# Scan for vulnerabilities
docker scan scentinel-backend
docker scan scentinel-frontend
```

Backup and Recovery

1. Database Backup:

```
#!/bin/bash
# backup_database.sh

BACKUP_DIR="/backup/mongodb"
DATE=$(date +%Y%m%d_%H%M%S)

# Create backup directory
mkdir -p $BACKUP_DIR

# Backup database
docker-compose exec scentinel-mongodb mongodump \\
  --db scentinel \\
  --out $BACKUP_DIR/dump_$DATE

# Compress backup
tar -czf $BACKUP_DIR/scentinel_$DATE.tar.gz $BACKUP_DIR/dump_$DATE

# Clean up
rm -rf $BACKUP_DIR/dump_$DATE

# Keep only last 30 days of backups
find $BACKUP_DIR -name "scentinel_*.tar.gz" -mtime +30 -delete

echo "Backup completed: scentinel_$DATE.tar.gz"
```

2. Database Recovery:

```
#!/bin/bash
# restore_database.sh

BACKUP_FILE=$1

if [ -z "$BACKUP_FILE" ]; then
    echo "Usage: $0 <backup_file.tar.gz>"
    exit 1
fi

# Extract backup
tar -xzf $BACKUP_FILE -C /tmp/

# Stop application
docker-compose stop backend frontend

# Restore database
docker-compose exec scentinel-mongodb mongorestore \\
  --db scentinel \\
  --drop \\
  /tmp/dump_*/scentinel/

# Start application
docker-compose start backend frontend

echo "Database restored from $BACKUP_FILE"
```

Log Management

1. Log Rotation Configuration:

```
# /etc/logrotate.d/scentinel
/var/log/scentinel/*.log {
    daily
    missingok
    rotate 30
    compress
    delaycompress
    notifempty
    create 0644 root root
    postrotate
        docker-compose restart backend frontend
    endscript
}
```

2. Centralized Logging:

```
# docker-compose.logging.yml
version: '3.8'

services:
  backend:
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"
```

```
frontend:
  logging:
    driver: "json-file"
    options:
      max-size: "10m"
      max-file: "3"
```

---

14. AUTOMATION SCRIPTS

Quick Setup Scripts

The project includes automated setup scripts to streamline the installation process on different operating systems.

For Linux/macOS: `setup.sh`

```
#!/bin/bash
# Complete automated setup for Unix-based systems

./setup.sh
```

What the script does:

1. **Environment Validation**

   ```
   # Checks for Docker and Docker Compose installation
   if ! command -v docker &> /dev/null; then
       echo "❌ Docker is not installed. Please install Docker first."
       exit 1
   fi
   ```

2. **Configuration Setup**

   ```
   # Creates backend .env file from template
   if [ ! -f "backend/.env" ]; then
       cp backend/env.example backend/.env
   fi
   ```

3. **Directory Structure Creation**

   ```
   # Creates necessary directories
   mkdir -p data/db
   mkdir -p logs
   ```

4. **Permissions Management**

   ```
   # Sets executable permissions on scripts
   chmod +x pretraining/pretrain.sh
   chmod +x pretraining/pretrain.bat
   chmod +x backend/entrypoint.sh
   ```

5. **User Guidance**
    - Provides clear next steps
    - Shows URLs for accessing services
    - Mentions optional ML model pre-training

For Windows: `setup-windows.bat`

```
@echo off
# Complete automated setup for Windows systems

setup-windows.bat
```

What the script does:

1. **Docker Service Validation**

    ```
    docker info >nul 2>&1
    if %errorlevel% neq 0 (
        echo ❌ Docker is not running. Please start Docker Desktop first.
        exit /b 1
    )
    ```

2. **System Cleanup**

    ```
    # Calls cleanup script to remove problematic files
    call cleanup-windows.bat

    # Stops existing containers and removes images
    docker compose down -v
    docker rmi scentinel-backend scentinel-frontend 2>nul
    ```

3. **Environment Preparation**

    ```
    # Creates directories and configuration files
    if not exist "data\\db" mkdir data\\db
    if not exist "logs" mkdir logs

    # Creates .env file from template
    if not exist "backend\\.env" (
        copy backend\\env.example backend\\.env
    )
    ```

4. **Container Deployment**

    ```
    # Builds and starts all services
    docker compose up --build
    ```

Script Usage Instructions

Linux/macOS Setup:

```
# Make script executable (if needed)
chmod +x setup.sh

# Run setup
./setup.sh

# Follow the prompted next steps:
# 1. Edit backend/.env if needed
# 2. Start services: docker-compose up -d
# 3. Access at <http://localhost:3000>
```

Windows Setup:

```
# Simply run the batch file
setup-windows.bat

# The script will:
# 1. Validate Docker
# 2. Clean previous installations
# 3. Build and start all services automatically
# 4. Display access URLs
```

Additional Utility Scripts

ML Model Pre-training (Optional):

Linux/macOS:

```
./pretraining/pretrain.sh
```

Windows:

```
./pretraining/pretrain.bat
```

Backend Entry Point:

```
# Used internally by Docker containers
./backend/entrypoint.sh
```

Script Features

- **Error Handling**: All scripts include comprehensive error checking
- **User Feedback**: Clear progress indicators and status messages
- **Cross-Platform**: Separate implementations for Unix and Windows
- **Idempotent**: Safe to run multiple times
- **Cleanup**: Windows script includes automatic cleanup functionality
- **Validation**: Pre-flight checks for required dependencies

Troubleshooting Script Issues

If scripts fail to run:

1. **Permission Issues (Linux/macOS):**

   ```
   chmod +x setup.sh
   chmod +x pretraining/pretrain.sh
   ```

2. **Docker Not Running:**
   - Start Docker Desktop
   - Verify with: `docker --version`
3. **Port Conflicts:**
   - Check if ports 3000, 5000/5001 are in use
   - Stop conflicting services
4. **File Permission Issues (Windows):**
   - Run Command Prompt as Administrator
   - Ensure Docker Desktop is running
