FeelWise - Complete System Documentation Guide

📋 Table of Contents

🎯 System Overview

FeelWise is an **Emotional Intelligence Platform** that analyzes user emotions through multiple input methods:

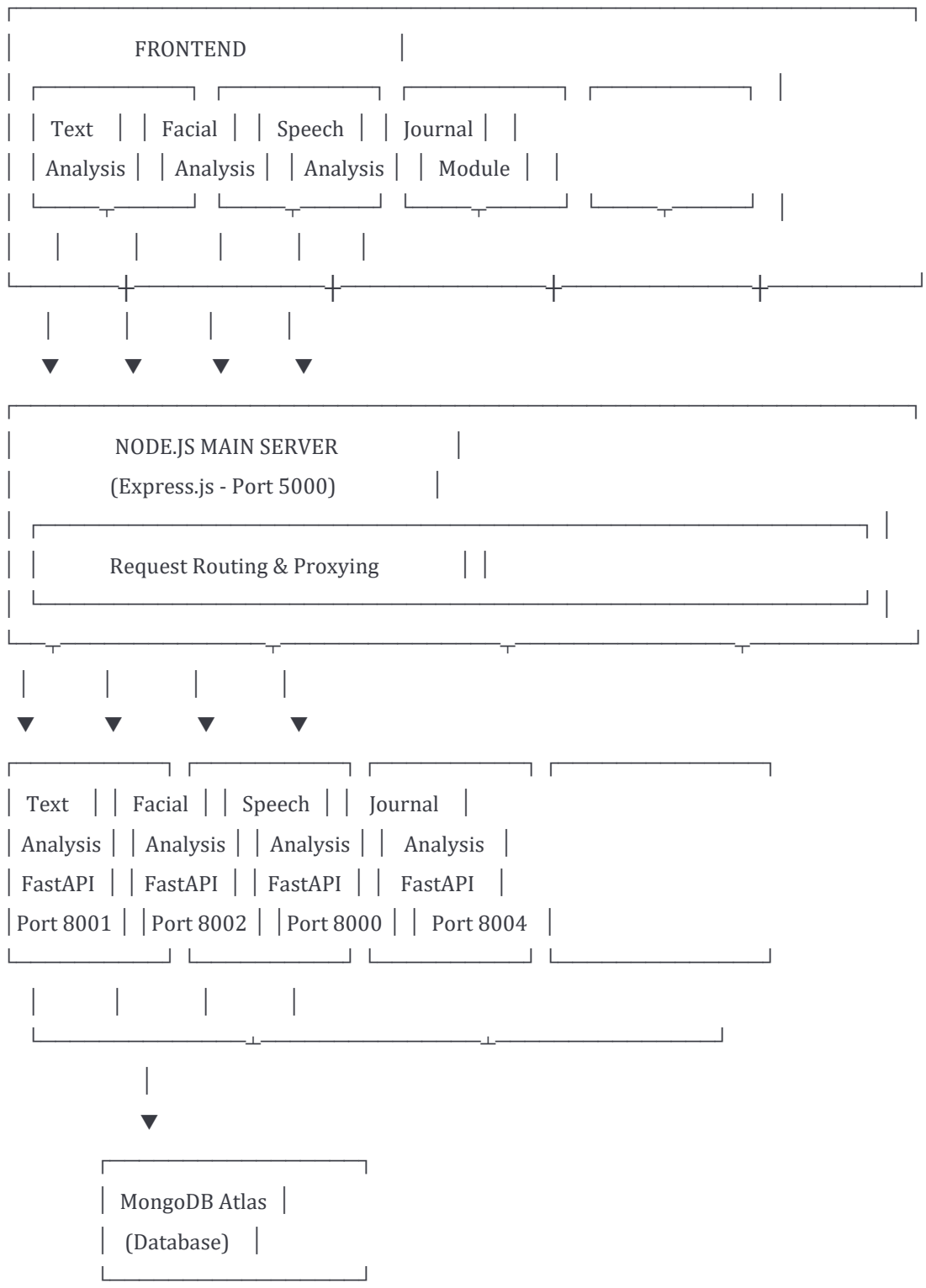
- **Text Analysis** - Analyzes written text for emotional content
- **Facial Expression Analysis** - Detects emotions from facial images
- **Voice/Speech Analysis** - Analyzes vocal tone and speech patterns
- **Journal Module** - AI-powered journaling with mood tracking

Key Features

☑ Multi-modal emotion detection (text, face, voice)
☑ Real-time sentiment analysis
☑ Personalized recommendations and daily challenges
☑ Mood tracking with historical data visualization
☑ User authentication and progress tracking
☑ AI-powered journaling with insights

🏗️ Architecture

System Architecture Diagram



💻 Technology Stack

Frontend Technologies

Technology	Purpose	Version
HTML5	Structure & Markup	-
CSS3	Styling & Animations	-
JavaScript (ES6+)	Client-side logic	-
Chart.js	Data visualization	Latest
Font Awesome	Icons	6.4.0
Google Fonts (Nunito)	Typography	-

Why These Choices?

- **Vanilla JavaScript**: Fast, no framework overhead, direct DOM manipulation
- **Chart.js**: Lightweight, excellent for mood trend visualizations
- **Responsive Design**: Works on all devices without additional frameworks

---

Backend Technologies

1. Node.js + Express.js (Main Server)

javascript

// Purpose: API Gateway, Request Routing, Authentication

Port: 5000

File: main-server.js

Key Libraries:

- `express` (4.18.2) - Web framework
- `cors` (2.8.5) - Cross-Origin Resource Sharing
- `mongoose` (8.18.0) - MongoDB ODM
- `bcryptjs` (3.0.2) - Password hashing
- `jsonwebtoken` (9.0.2) - JWT authentication
- `multer` (2.0.2) - File uploads
- `node-fetch` (2.6.7) - HTTP requests
- `dotenv` (17.2.1) - Environment variables

Why Express.js?

- Lightweight and fast

- Excellent middleware support
- Easy integration with MongoDB
- Strong community support

---

2. FastAPI (Python Microservices)

Why FastAPI?

- **High Performance**: As fast as Node.js and Go
- **Automatic API Documentation**: Built-in Swagger UI
- **Type Safety**: Python type hints for better code quality
- **Async Support**: Handle concurrent requests efficiently
- **ML Integration**: Easy integration with AI/ML libraries

AI/ML Models & Libraries

1. Text Analysis Service (Port 8001)

python

File: text-analysis-api.py

Framework: FastAPI

Libraries Used:

- `fastapi` - Web framework
- `pydantic` - Data validation
- `uvicorn` - ASGI server

Emotion Detection Method:

- **Keyword-based analysis** using predefined emotion dictionaries
- Maps emotions: joy, sadness, anger, fear, surprise, love

File: `analysis_utils.py`

python

```
EMOTION_KEYWORDS = {
    "joy": ["happy", "joy", "excited", "great", "wonderful"],
    "sadness": ["sad", "down", "unhappy", "depressed"],
    "anger": ["angry", "mad", "furious", "irritated"],
```

```
    # ... more emotions
}
```

---

2. Facial Analysis Service (Port 8002)

python

File: face-analysis-api.py

Framework: FastAPI + DeepFace

AI Model: DeepFace

- **Library**: `deepface` (Facebook's facial recognition)
- **Backend**: PyTorch (`torch`)
- **Detection**: OpenCV (`opencv`)

How It Works:

1. Receives base64 encoded image
2. Decodes image to numpy array
3. Uses DeepFace to analyze facial emotions
4. Returns dominant emotion with confidence scores

Detected Emotions:

- Happy, Sad, Angry, Surprise, Fear, Disgust, Neutral

Code Snippet:

python

```
result = DeepFace.analyze(
    img,
    actions=['emotion'],
    enforce_detection=False,
    detector_backend="opencv"
)
emotion = result[0]['dominant_emotion']
```

---

3. Speech Analysis Service (Port 8000)

python

File: speech_analysis_fastapi.py

Framework: FastAPI + Transformers (Hugging Face)

AI Model: Wav2Vec2

- **Model Name**: `ehcalabres/wav2vec2-lg-xlsr-en-speech-emotion-recognition`
- **Type**: Pre-trained speech emotion recognition
- **Framework**: Hugging Face Transformers + PyTorch

Libraries:

- `transformers` - Hugging Face models
- `torch` - PyTorch deep learning
- `numpy` - Numerical operations
- `ffmpeg` - Audio conversion

How It Works:

1. Records audio in WebM format
2. Converts to WAV using FFmpeg (16kHz, mono)
3. Extracts features using Wav2Vec2
4. Classifies emotion from audio features
5. Returns emotion with confidence scores

Detected Emotions:

- Angry, Disgust, Fear, Happy, Neutral, Sad, Surprise

Key Code:

python

```
# Audio preprocessing
inputs = feature_extractor(audio, sampling_rate=16000,
                           return_tensors="pt")

# Model inference
with torch.no_grad():
    logits = model(**inputs).logits
    probs = torch.nn.functional.softmax(logits, dim=-1)
```

---

4. Journal Analysis Service (Port 8004)

python

File: journal_api.py

Framework: FastAPI + VADER Sentiment + Custom NLP

AI/NLP Libraries:

- `vaderSentiment` - Sentiment analysis (positive/negative/neutral)
- Custom keyword extraction
- Text summarization algorithms

Features:

1. **Sentiment Analysis**: VADER (Valence Aware Dictionary and sEntiment Reasoner)
2. **Mood Detection**: Reuses emotion analysis from `analysis_utils.py`
3. **Keyword Extraction**: Custom TF-IDF-like approach
4. **Text Summarization**: First sentence extraction + word limiting
5. **Personalized Suggestions**: Rule-based recommendation engine

How Journal AI Works:

python

```
# Sentiment scoring
sentiment = sentiment_analyzer.polarity_scores(text)
sentiment_score = sentiment["compound"]  # -1 to +1

# Emotion detection
emotion_result = analyze_text(text)
dominant_emotion = emotion_result["emotion"]

# Mood score calculation (0-1 scale)
mood_score = (emotion_score * 0.6) + (sentiment_normalized * 0.4)
```

Database & Storage

MongoDB Atlas

Why MongoDB?

- **Flexible Schema**: Perfect for varying user data structures
- **Cloud-Based**: MongoDB Atlas for automatic backups and scaling
- **JSON-like**: Easy integration with JavaScript and Python

- **Scalable**: Can handle millions of records

Connection:

javascript
```
MONGODB_URI = "mongodb+srv://username:password@cluster.mongodb.net/feelwise_db"
```

Collections:

1. users Collection

javascript
```
{
 _id: ObjectId,
 username: String,
 email: String,
 password: String (bcrypt hashed),
 image: String (profile image path),
 mood: String,
 progress: {
  selfAwareness: Number,
  selfRegulation: Number,
  empathy: Number,
  socialSkills: Number,
  motivation: Number
 },
 badges: [String],
 moodHistory: Array,
 completedChallenges: Array,
 createdAt: Date,
 updatedAt: Date
}
```

2. journals Collection

javascript
```
{
 _id: ObjectId,
 user_id: String,
```

```
  text: String,
  mood: String,
  prompt: String,
  datetime: ISO String,
  ai_summary: String,
  dominant_mood: String,
  mood_scores: {
    positive: Number,
    negative: Number,
    neutral: Number
  },
  keywords: [String],
  suggestion: String,
  sentiment_score: Number,
  emotion_distribution: Object,
  created_at: Date
}
```

3. text_analyses Collection (if implemented)

javascript

```
{
  _id: ObjectId,
  user_id: String,
  text: String,
  emotions: {
    positive: Number,
    negative: Number,
    neutral: Number
  },
  dominantEmotion: String,
  emotionDetails: Object,
  timestamp: ISO String
}
```

4. facial_analyses Collection (if implemented)

javascript

```
{
 _id: ObjectId,
 user_id: String,
 emotion: String,
 confidence: Number,
 emotionDistribution: Object,
 timestamp: ISO String
}
```

Local Storage (Browser)

Used for:

- Temporary data caching
- Guest user data
- Session management
- Recent analysis results

Storage Keys:

- `token` - JWT authentication token
- `user` - User profile data
- `emotionHistory_{userId}` - Text analysis history
- `facialAnalysisHistory_{userId}` - Facial analysis history
- `completedChallenges_{userId}` - Challenge completion data

🗀 Frontend Components

Page Structure

Page	File	Purpose
Home	`index.html`	Landing page with features
Login	`login.html`	User authentication
Signup	`signup.html`	New user registration
Profile	`profile.html`	User dashboard
Modules	`module.html`	Analysis options hub
Text Analysis	`text-analysis.html`	Text emotion detection
Facial Analysis	`facial-analysis.html`	Face emotion detection

Page	File	Purpose
Speech Analysis	`speech-analysis.html`	Voice emotion detection
Journal	`journal.html`	AI-powered journaling
Assessment	`full_assessment.html`	Comprehensive emotional report
Progress	`progress.html`	User progress tracking
Challenges	`dailychallenges.html`	Daily wellness challenges

Key JavaScript Files

1. text-analysis.js

- Sends text to backend for analysis
- Displays emotion distribution charts
- Provides personalized recommendations
- Tracks analysis history
- Syncs with MongoDB

Key Functions:

javascript

```
analyzeText(text)        // Analyzes user input
displayAnalysisResults()  // Shows results on UI
updateProgressChart()     // Updates Chart.js visualization
saveAnalysisToBackend()   // Saves to MongoDB
generateAssessmentReport() // Creates detailed report
```

2. facial-analysis.js

- Captures webcam video
- Processes image uploads
- Sends base64 images to backend
- Displays detected emotions
- Provides facial expression insights

Key Functions:

javascript

```
captureFromWebcam()      // Captures photo from camera
sendImageForAnalysis()   // Sends to FastAPI service
displayFacialResults()   // Shows emotion results
updateRecommendations()  // Context-aware suggestions
```

3. speech-analysis.js

- Records audio using MediaRecorder API
- Implements speech-to-text (Web Speech API)
- Sends audio to backend (base64)
- Handles FFmpeg audio conversion
- Displays voice analysis results

Key Functions:

```javascript
startRecording()      // Starts mic recording
stopRecording()       // Stops and processes audio
analyzeSpeech()       // Sends to FastAPI
blobToBase64()        // Converts audio to base64
```

4. journal.js

- AI-powered journaling interface
- Mood tag selection
- Speech-to-text journaling
- Real-time text analysis
- Historical mood trends
- Word cloud generation
- Badge/achievement system

Key Functions:

```javascript
analyzeEntry()        // Analyzes journal text
saveEntry()           // Saves to MongoDB
loadTimeline()        // Loads journal history
loadMoodTrend()       // Generates mood chart
loadWordCloud()       // Creates word cloud
updateBadges()        // Awards achievements
```

🔧 Backend Services

1. Main Server (Node.js/Express)

File: `main-server.js` **Port:** 5000

Responsibilities:

- **API Gateway**: Routes requests to appropriate microservices
- **Authentication**: JWT-based user auth
- **Database Operations**: MongoDB CRUD operations
- **File Handling**: Profile image uploads
- **CORS Management**: Cross-origin request handling

Key Endpoints:

```
javascript
// Authentication
POST /api/auth/register   // User registration
POST /api/auth/login      // User login
GET  /api/auth/profile    // Get user profile
PUT  /api/auth/profile    // Update profile

// Analysis Proxying
POST /analyze          // → Text Analysis (8001)
POST /analyze-face     // → Facial Analysis (8002)
POST /analyze-speech   // → Speech Analysis (8000)

// Journal Module
GET  /journal/prompts     // Get random prompt
POST /journal/analyze     // Analyze journal text
POST /journal/entry       // Save journal entry
GET  /journal/entries     // Get entries
GET  /journal/insights    // Get mood trends
DELETE /journal/entry/:id // Delete entry

// Progress Tracking
GET  /api/progress        // Get user progress
POST /api/progress        // Update progress
```

Middleware Chain:

```
javascript
1. CORS headers
2. JSON body parsing (200MB limit for images/audio)
```

3. Request ID generation
4. JWT authentication (where required)
5. Route handlers
6. Error handling

---

2. Text Analysis API (FastAPI)

File: `text-analysis-api.py` **Port:** 8001

Algorithm:

1. Receives text input
2. Tokenizes text (lowercase, split words)
3. Matches words against emotion keyword dictionaries
4. Calculates emotion scores
5. Determines emotion distribution percentages
6. Returns dominant emotion + breakdown

API Response:

json


```
{
 "text": "I am so happy today!",
 "emotion": "joy",
 "emotion_distribution": {
  "joy": 50.0,
  "sadness": 10.0,
  "anger": 5.0,
  "fear": 5.0,
  "surprise": 15.0,
  "love": 15.0
 }
}
```


---

3. Facial Analysis API (FastAPI)

File: `face-analysis-api.py` **Port:** 8002

Process Flow:

1. Receive base64 image
2. Decode to numpy array
3. Load image with OpenCV
4. Use DeepFace for emotion detection
5. Extract dominant emotion
6. Return results with recommendations

DeepFace Model:

- **Pre-trained on FER dataset**
- **Accuracy**: ~65-70% on diverse faces
- **7 emotion categories**

API Response:

json

```
{
  "emotion": "happy",
  "recommendation": "Share your happiness with someone today!",
  "challenge": "Compliment three people today.",
  "tip": "Happiness is amplified when shared.",
  "trend": {
    "labels": ["Mon", "Tue", "Wed", "Thu", "Fri"],
    "values": [3, 4, 5, 4, 5]
  }
}
```

---

4. Speech Analysis API (FastAPI)

File: `speech_analysis_fastapi.py`

Port: 8000

Audio Processing Pipeline:

```
WebM Audio (Frontend)
  ↓
FFmpeg Conversion → WAV (16kHz mono)
  ↓
```

Load with wave library → Numpy array

↓

Normalize audio [-1, 1]

↓

Wav2Vec2 Feature Extraction

↓

Model Inference (PyTorch)

↓

Softmax → Emotion Probabilities

↓

Dominant Emotion + Top 3

Model Details:

- **Parameters**: ~300M parameters
- **Training Data**: English speech emotion datasets
- **Inference Time**: ~2-5 seconds per audio clip
- **Accuracy**: ~60-70% on clean audio

API Response:

json


```
{
 "status": "success",
 "emotion": "happiness",
 "raw_label": "happy",
 "probabilities": {
  "angry": 0.05,
  "happy": 0.72,
  "sad": 0.10,
  "neutral": 0.13
 },
 "top3": [
  ["happy", 0.72],
  ["neutral", 0.13],
  ["sad", 0.10]
 ],
 "recommendation": "Share your positive energy!",
 "daily_challenge": "Compliment three people.",
```

```
  "daily_tip": "Happiness is contagious."
}
```

---

5. Journal Analysis API (FastAPI)

File: `journal_api.py` **Port:** 8004

Analysis Components:

1. **VADER Sentiment Analysis**

python

```
sentiment = analyzer.polarity_scores(text)
# Returns: {"neg": 0.1, "neu": 0.5, "pos": 0.4, "compound": 0.6}
```

2. **Emotion Detection** (reuses `analysis_utils.py`)
3. **Keyword Extraction**

python

```
# Removes stop words
# Counts word frequency
# Returns top N keywords
```

4. **Text Summarization**

python

```
# Extracts first sentence
# Limits to max_words (default 15)
```

5. **Mood Score Calculation**

python

```
mood_score = (emotion_score * 0.6) + (sentiment_normalized * 0.4)
```

API Response:

json

```
{
  "ai_summary": "Feeling grateful and energized today.",
  "dominant_mood": "joy",
  "mood_scores": {
```

```
  "joy": 0.85,
  "positive": 0.8,
  "negative": 0.1,
  "neutral": 0.1
 },
 "keywords": ["grateful", "energized", "morning", "workout"],
 "suggestion": "Use this positive momentum to tackle challenges.",
 "sentiment_score": 0.75,
 "emotion_distribution": {...}
}
```

🔐 Authentication System

JWT-Based Authentication

Flow:

1. User registers/logs in
2. Server generates JWT token
3. Token stored in localStorage
4. Token sent in Authorization header for protected routes
5. Server verifies token on each request

JWT Payload:

javascript

```
{
 "id": "user_mongodb_id",
 "email": "user@example.com",
 "iat": 1234567890, // Issued at
 "exp": 1234571490  // Expiration (1 hour)
}
```

Password Security:

- **Hashing Algorithm**: bcrypt (10 rounds)
- **Never stored in plain text**
- **Salted automatically by bcrypt**

Protected Routes:

- All `/api/auth/profile` endpoints
- All `/api/progress` endpoints
- Journal saving endpoints (optional - works with guests too)

📊 Data Flow Examples

Text Analysis Flow

mermaid

```
sequenceDiagram
    User->>Frontend: Enters text
    Frontend->>Main Server: POST /analyze
    Main Server->>Text API: POST /analyze (8001)
    Text API->>Text API: Analyze emotions
    Text API->>Main Server: Return results
    Main Server->>Frontend: Return results
    Frontend->>Frontend: Display + Save to localStorage
    Frontend->>Main Server: POST /api/text-analysis/save
    Main Server->>MongoDB: Insert document
    MongoDB->>Main Server: Success
    Main Server->>Frontend: Confirmation
```

Facial Analysis Flow

mermaid

```
sequenceDiagram
    User->>Frontend: Captures/uploads image
    Frontend->>Frontend: Convert to base64
    Frontend->>Main Server: POST /analyze-face
    Main Server->>Facial API: POST /analyze_face (8002)
    Facial API->>DeepFace: Analyze image
    DeepFace->>Facial API: Emotion results
    Facial API->>Main Server: Return results
    Main Server->>Frontend: Return results
    Frontend->>Frontend: Display + recommendations
```

Journal Entry Flow

mermaid

```
sequenceDiagram
    User->>Frontend: Writes journal entry
    Frontend->>Main Server: POST /journal/analyze
    Main Server->>Journal API: POST /journal/analyze
    Journal API->>VADER: Sentiment analysis
    Journal API->>Custom NLP: Emotion + keywords
    Journal API->>Main Server: Analysis results
    Main Server->>Frontend: Display analysis
    User->>Frontend: Clicks "Save Entry"
    Frontend->>Main Server: POST /journal/entry
    Main Server->>Journal API: POST /journal/entry
    Journal API->>MongoDB: Insert entry
    MongoDB->>Journal API: Success + stats
    Journal API->>Main Server: Return entry + streak
    Main Server->>Frontend: Display confirmation
```

🚀 Setup & Installation

Prerequisites

- **Node.js** (v16+)
- **Python** (v3.8+)
- **MongoDB Atlas Account** (free tier)
- **FFmpeg** (for speech analysis)

Environment Variables

Create `.env` file:

bash

```
# MongoDB
MONGODB_URI=mongodb+srv://username:password@cluster.mongodb.net/feelwise_db

# JWT Secret
JWT_SECRET=your_secret_key_here_min_32_characters

# Server Port
```

```
PORT=5000
```

Installation Steps

1. Clone Repository

bash

```
git clone <repository-url>
cd feelwise
```

2. Backend Setup (Node.js)

bash

```
cd FastAPI_Backend
npm install
```

3. Python Dependencies

bash

```
# Text Analysis
pip install fastapi uvicorn pydantic

# Facial Analysis
pip install deepface opencv-python torch

# Speech Analysis
pip install transformers torch numpy

# Journal Analysis
pip install vaderSentiment pymongo python-dotenv
```

Or use requirements.txt:

bash

```
pip install -r requirements.txt
```

4. FFmpeg Installation

Windows:

bash

```
# Download from https://ffmpeg.org/download.html
# Add to PATH
```

Mac:

```
bash
brew install ffmpeg
```

Linux:

```
bash
sudo apt-get install ffmpeg
```

5. MongoDB Setup

1. Create MongoDB Atlas account
2. Create new cluster (free M0)
3. Create database user
4. Whitelist your IP (or 0.0.0.0/0 for testing)
5. Get connection string
6. Add to `.env` file

Running the Application

Terminal 1 - Main Server:

```
bash
cd FastAPI_Backend
node main-server.js
```

Terminal 2 - Text Analysis:

```
bash
cd FastAPI_Backend
python text-analysis-api.py
```

Terminal 3 - Facial Analysis:

```
bash
cd FastAPI_Backend
python face-analysis-api.py
```

Terminal 4 - Speech Analysis:

bash

```
cd FastAPI_Backend
python speech_analysis_fastapi.py
```

Terminal 5 - Journal API:

bash

```
cd FastAPI_Backend
python journal_api.py
```

Terminal 6 - Frontend:

bash

```
cd Frontend
# Use Live Server extension in VS Code
# OR
python -m http.server 5500
```

```

**Access Application:**
```

```
http://localhost:5500
```

□ Testing

Manual Testing Checklist

Text Analysis:

- ☐ Enter positive text → Check joy/love detection
- ☐ Enter negative text → Check sadness/anger detection
- ☐ Test empty input → Should show error
- ☐ Test very long text → Should process correctly
- ☐ Check chart updates with new data

Facial Analysis:

- ☐ Capture from webcam → Should detect face
- ☐ Upload image → Should analyze correctly

- [ ] Test with no face → Should handle gracefully
- [ ] Test multiple faces → Should detect primary face
- [ ] Check emotion confidence scores

Speech Analysis:

- [ ] Record happy voice → Should detect positive
- [ ] Record sad voice → Should detect negative
- [ ] Test short recording → Should warn user
- [ ] Test long recording → Should process completely
- [ ] Check transcript accuracy

Journal:

- [ ] Save entry → Should appear in timeline
- [ ] Load entries → Should display correctly
- [ ] Delete entry → Should remove from list
- [ ] Check mood trends → Chart should update
- [ ] Test word cloud → Keywords should appear

Authentication:

- [ ] Register new user → Should create account
- [ ] Login → Should redirect to dashboard
- [ ] Logout → Should clear session
- [ ] Protected routes → Should require login
- [ ] Update profile → Should save changes

📈 Performance Optimization

Frontend Optimizations

- **Lazy loading** images
- **Debounced** search inputs
- **Cached** analysis results in localStorage
- **Chart.js** destroy/recreate to prevent memory leaks
- **Event delegation** for dynamic elements

Backend Optimizations

- **Connection pooling** for MongoDB
- **Request ID** tracking for debugging
- **Compression** middleware
- **Rate limiting** (can be added)
- **Caching** frequent queries (can be added)

Model Optimizations

- **Model caching** - Load models once at startup
- **Batch processing** - Process multiple requests together (can be added)
- **GPU acceleration** - Use CUDA if available
- **Model quantization** - Reduce model size (can be added)

🐛 Common Issues & Solutions

1. MongoDB Connection Fails

Error: `MongoServerSelectionTimeoutError`

Solution:

- Check internet connection
- Verify MongoDB URI in `.env`
- Whitelist IP address in MongoDB Atlas
- Check database username/password

2. FFmpeg Not Found

Error: `FFmpeg not installed or not in PATH`

Solution:

- Install FFmpeg
- Add to system PATH
- Restart terminal
- Verify: `ffmpeg -version`

3. Model Loading Errors

Error: `Failed to load model`

Solution:

- Check Python version (3.8+)
- Install PyTorch: `pip install torch`
- Install transformers: `pip install transformers`
- Download model manually if needed

4. CORS Errors

Error: `Access to fetch blocked by CORS policy`

Solution:

- Ensure main server has CORS enabled
- Check allowed origins in `main-server.js`
- Use same origin for frontend and backend during development

5. Speech Recognition Not Working

Error: `Speech recognition not supported`

Solution:

- Use Chrome/Edge (best support)
- Enable microphone permissions
- Use HTTPS (required for production)
- Check browser console for errors

Libraries & Dependencies Summary

Frontend

json

```
{
  "Chart.js": "Data visualization",
  "Font Awesome": "Icons",
  "Google Fonts": "Typography",
  "Web Speech API": "Speech recognition",
  "MediaRecorder API": "Audio recording",
  "Canvas API": "Image capture"
}
```

Backend (Node.js)

json


```
{
  "express": "Web framework",
  "mongoose": "MongoDB ODM",
  "bcryptjs": "Password hashing",
  "jsonwebtoken": "JWT auth",
  "cors": "CORS handling",
  "multer": "File uploads",
  "node-fetch": "HTTP requests",
  "dotenv": "Environment variables"
}
```


AI/ML (Python)

json


```
{
  "fastapi": "API framework",
  "deepface": "Facial emotion detection",
  "transformers": "Hugging Face models",
  "torch": "PyTorch deep learning",
  "vaderSentiment": "Sentiment analysis",
  "opencv-python": "Image processing",
  "numpy": "Numerical computing",
  "pymongo": "MongoDB driver"
}
```

🎓 Learning Resources

For Understanding the System:

1. **FastAPI Tutorial**
   - https://fastapi.tiangolo.com/tutorial/
2. **Express.js Guide**
   - https://expressjs.com/en/guide/routing.html
3. **MongoDB University**
   - https://university.mongodb.com/

4. **Hugging Face Transformers**
    - https://huggingface.co/docs/transformers/
5. **DeepFace Documentation**
    - https://github.com/serengil/deepface
6. **Chart.js Docs**
    - https://www.chartjs.org/docs/
7. **JWT Authentication**
    - https://jwt.io/introduction