EmotionMelody Project - Technical Report

Table of Contents

Project Overview

EmotionMelody (branded as "Moodify") is an innovative web application that combines artificial intelligence, computer vision, and music streaming APIs to create a personalized music recommendation system based on real-time emotion detection. The application uses facial expression analysis to determine a user's emotional state and suggests appropriate music tracks from Spotify, enhanced with weather context for more accurate recommendations.

Project Structure

```
EmotionMelody/
├── client/                    # Frontend React application
│   ├── src/
│   │   ├── components/        # React components
│   │   ├── lib/              # Utility libraries
│   │   ├── pages/            # Page components
│   │   └── hooks/            # Custom React hooks
├── server/                    # Backend Express.js server
├── shared/                    # Shared schemas and types
└── Configuration files
```

Architecture & Technology Stack

Frontend Technologies

- **React 18.3.1** - Modern UI library with hooks
- **TypeScript 5.6.3** - Type-safe JavaScript
- **Vite 5.4.19** - Fast build tool and dev server
- **TailwindCSS 3.4.14** - Utility-first CSS framework
- **Radix UI** - Accessible component primitives
- **TanStack Query** - Data fetching and state management
- **Wouter** - Lightweight routing
- **Face-API.js 0.22.2** - Client-side face detection and emotion recognition

Backend Technologies

- **Node.js with Express 4.21.2** - Web server framework
- **TypeScript** - Server-side type safety
- **Drizzle ORM 0.39.1** - TypeScript ORM
- **PostgreSQL** - Database (via Neon)
- **Zod 3.23.8** - Runtime type validation

External APIs

- **Spotify Web API** - Music streaming and search
- **OpenWeather API** - Weather data integration

Development Tools

- **ESBuild** - Fast JavaScript bundler
- **PostCSS & Autoprefixer** - CSS processing
- **Drizzle Kit** - Database migrations

File-by-File Analysis

Configuration Files

`package.json`

The project manifest defines a full-stack Node.js application with comprehensive dependencies for both frontend and backend development. Key scripts include:

- `dev`: Runs development server using tsx
- `build`: Creates production build with Vite and ESBuild
- `start`: Production server execution

- `db:push`: Database schema synchronization

`tsconfig.json`

TypeScript configuration emphasizing modern ES modules with strict type checking. Notable configurations:

- **Module Resolution**: Bundler-based for Vite compatibility
- **Path Mapping**: `@/*` for client source, `@shared/*` for shared schemas
- **Strict Mode**: Enabled for type safety
- **JSX**: Preserve mode for React processing

`vite.config.ts`

Vite configuration optimized for React development with:

- **Plugin Integration**: React, theme handling, error overlay
- **Path Resolution**: Alias mapping for clean imports
- **Build Configuration**: Output to dist/public for deployment
- **Development Features**: Hot Module Replacement, error handling

Backend Architecture

`server/index.ts`

The main server entry point implementing:

- **Express Application Setup**: JSON parsing, URL encoding
- **Request Logging Middleware**: API call monitoring with timing
- **Error Handling**: Centralized error processing
- **Environment-Based Serving**: Development (Vite) vs Production (static)
- **Port Configuration**: Fixed port 5000 for deployment

Key Functions:

```
// Request logging with performance metrics
app.use((req, res, next) => {
  const start = Date.now();
  // ... logging logic
});

// Centralized error handling
app.use((err: any, _req: Request, res: Response, _next: NextFunction) => {
  const status = err.status || err.statusCode || 500;
  const message = err.message || "Internal Server Error";
  res.status(status).json({ message });
});
```

`server/routes.ts`

API route definitions implementing RESTful endpoints:

Spotify Integration:

- `/api/spotify/search` - General Spotify track search
- `/api/spotify/recommendations/emotion/:emotion` - Emotion-based recommendations

Weather Integration:

- `/api/weather` - Location-based weather data

Technical Implementation Details:

```
// Spotify authentication using client credentials flow
const tokenResponse = await fetch('https://accounts.spotify.com/api/token', {
  method: 'POST',
  headers: {
    'Authorization': `Basic ${Buffer.from(
      `${process.env.SPOTIFY_CLIENT_ID}:${process.env.SPOTIFY_CLIENT_SECRET}`
    ).toString('base64')}`
  },
  body: new URLSearchParams({ grant_type: 'client_credentials' })
});
```

Emotion-to-Music Mapping:
The system implements sophisticated emotion-to-search-query mapping:

- Happy → "bollywood hindi happy OR upbeat songs arijit"
- Sad → "bollywood hindi sad OR melancholic songs"
- Angry → "bollywood hindi powerful OR intense songs"

`server/storage.ts`

Database abstraction layer with in-memory implementation:

- **Interface-Based Design**: `IStorage` interface for future database integration
- **Memory Storage**: Development-friendly user storage
- **CRUD Operations**: User creation, retrieval by ID and username

`server/vite.ts`

Development server configuration handling:

- **Vite Integration**: Middleware setup for development
- **Hot Module Replacement**: Real-time code updates
- **Static Serving**: Production file serving
- **Template Processing**: Dynamic HTML template injection

Frontend Architecture

`client/src/main.tsx`

React application entry point with minimal, clean initialization:

```
import { createRoot } from "react-dom/client";
import App from "./App";
import "./index.css";

createRoot(document.getElementById("root")!).render(<App />);
```

`client/src/App.tsx`

Root application component implementing:

- **Query Client Integration**: TanStack Query for API state management
- **Routing Setup**: Wouter-based routing
- **Toast System**: User notification system
- **Component Architecture**: Clean separation of concerns

`client/src/pages/Home.tsx`

Main application page orchestrating core functionality:

State Management:

```
const [currentEmotion, setCurrentEmotion] = useState<Emotion | null>(null);
const [weatherLocation, setWeatherLocation] = useState<string | null>(null);
```

Geolocation Integration:

```
useEffect(() => {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(
      (position) => {
        const { latitude, longitude } = position.coords;
        setWeatherLocation(`lat=${latitude}&lon=${longitude}`);
      },
      (error) => {
        setWeatherLocation("city=New York"); // Fallback
      }
    );
  }
}, [toast]);
```

Core Components

`WebcamSection.tsx`

The heart of the emotion detection system:

Features:

- **Camera Permission Management**: Progressive permission requests
- **Face-API Integration**: Real-time facial expression analysis
- **Visual Feedback**: Face detection overlay with emotion labels
- **Error Handling**: Graceful degradation for camera/model issues

Technical Implementation:

```
const detectEmotion = async () => {
  const detection = await detectFace(video);
  if (detection) {
    const { emotion, box } = detection;
    setDetectedEmotion(emotion);
    setFacePosition(scaledBoxCoordinates);
    onEmotionDetected(emotion);
  }
};
```

Real-time Processing:

- 1-second intervals for emotion detection
- Coordinate scaling for face overlay
- Model loading state management

`SongRecommendations.tsx`

Music recommendation and playback component:

Features:

- **Dynamic Loading**: Emotion-based song fetching
- **Audio Preview**: In-browser 30-second previews
- **Spotify Integration**: Fallback to Spotify web player
- **Auto-play Logic**: Intelligent first-song selection

Audio Management:

```
const handlePlayClick = (song: Song) => {
  if (!song.previewUrl) {
    window.open(`https://open.spotify.com/track/${song.id}`, '_blank');
    return;
  }

  const audio = new Audio(song.previewUrl);
  audio.play().catch(error => {
    // Fallback to Spotify on error
    window.open(`https://open.spotify.com/track/${song.id}`, '_blank');
  });
};
```

`MoodInformation.tsx`

Educational and informational component providing:

- **Emotion Descriptions**: Context for detected emotions
- **Genre Suggestions**: Music style recommendations
- **Process Explanation**: How the system works
- **Privacy Information**: Data handling transparency

`WeatherWidget.tsx`

Weather integration component featuring:

- **Icon Mapping**: Weather condition visualization
- **Temperature Display**: Celsius temperature with location
- **Responsive Design**: Compact weather card layout

Utility Libraries

`lib/faceDetection.ts`

Face-API.js wrapper providing:

Model Loading:

```
export async function loadModels(): Promise<void> {
  const MODEL_URL = 'https://justadudewhohacks.github.io/face-api.js/models';

  await Promise.all([
    faceapi.nets.tinyFaceDetector.loadFromUri(MODEL_URL),
    faceapi.nets.faceLandmark68Net.loadFromUri(MODEL_URL),
    faceapi.nets.faceRecognitionNet.loadFromUri(MODEL_URL),
    faceapi.nets.faceExpressionNet.loadFromUri(MODEL_URL)
  ]);
}
```

Expression Analysis:

```
const result = await faceapi.detectSingleFace(videoElement, options)
  .withFaceLandmarks()
  .withFaceExpressions();

const dominantExpression = Object.keys(expressions).reduce((a, b) =>
  expressions[a] > expressions[b] ? a : b
);
```

`lib/queryClient.ts`

API client configuration with:

- **Error Handling**: HTTP status code management
- **Authentication**: Cookie-based session handling
- **Query Configuration**: Optimized caching and retry policies

`lib/api.ts`

Typed API functions for external service integration:

- **Spotify API**: Song recommendations and search
- **Weather API**: Location-based weather data
- **Type Safety**: Full TypeScript integration

Shared Schema (`shared/schema.ts`)

Database Schema:

```
export const users = pgTable("users", {
  id: serial("id").primaryKey(),
  username: text("username").notNull().unique(),
  password: text("password").notNull(),
});
```

Type Definitions:

```
export const emotionSchema = z.object({
  emotion: z.enum(["happy", "sad", "angry", "neutral", "surprised", "fearful",
"disgusted"]),
  probability: z.number(),
});

export const songSchema = z.object({
  id: z.string(),
  name: z.string(),
  artist: z.string(),
  album: z.string().optional(),
  imageUrl: z.string().optional(),
  previewUrl: z.string().optional(),
  emotion: z.enum(["happy", "sad", "angry", "neutral", "surprised", "fearful",
"disgusted"]),
});
```

Core Concepts & Technologies

1. Computer Vision & AI

- **Face Detection**: TinyFaceDetector for lightweight face recognition
- **Emotion Recognition**: FaceExpressionNet for real-time emotion analysis
- **Client-Side Processing**: Privacy-preserving local computation

2. API Integration

- **OAuth 2.0**: Spotify client credentials flow
- **RESTful Design**: Clean API endpoint structure
- **Error Handling**: Comprehensive error management with fallbacks

3. Real-Time Processing

- **WebRTC**: Camera stream access via getUserMedia
- **Interval Processing**: 1-second emotion detection cycles
- **State Synchronization**: React state management for real-time updates

4. Modern Web Development

- **TypeScript**: End-to-end type safety
- **Component Architecture**: Modular React component design
- **Responsive Design**: Mobile-first TailwindCSS approach
- **Progressive Enhancement**: Graceful degradation for missing features

5. Data Flow Architecture

```
Camera Stream → Face Detection → Emotion Analysis → API Request → Music Recommendations →
Audio Playback
```

System Workflow

1. Application Initialization

1. **Model Loading**: Face-API models download and initialization
2. **Permission Requests**: Camera and geolocation access
3. **Weather Data**: Location-based weather information fetch
4. **UI Rendering**: Component hierarchy establishment

2. Emotion Detection Pipeline

1. **Video Stream**: Webcam feed capture
2. **Face Detection**: Real-time facial recognition
3. **Expression Analysis**: Emotion classification with confidence scores
4. **Visual Feedback**: Face overlay with emotion labels
5. **State Update**: Emotion propagation to recommendation system

3. Music Recommendation Process

1. **Emotion Mapping**: Convert emotion to search query
2. **Spotify Authentication**: Client credentials token acquisition
3. **Search Execution**: Emotion-specific music search
4. **Result Filtering**: Preview URL prioritization
5. **UI Population**: Song card generation with playback controls

4. Audio Playback Management

1. **Preview Handling**: 30-second audio clip playback
2. **Fallback Logic**: Spotify web player redirection
3. **State Management**: Play/pause status tracking
4. **Error Recovery**: Graceful audio failure handling

Key Features

1. Real-Time Emotion Detection

- **Accuracy**: Multiple facial landmarks for precise detection
- **Performance**: Optimized 1-second detection intervals
- **Privacy**: Client-side processing, no data transmission
- **Visual Feedback**: Live face detection overlay

2. Intelligent Music Recommendations

- **Contextual Mapping**: Emotion-specific music genre selection
- **Cultural Focus**: Hindi/Bollywood music prioritization
- **Quality Filtering**: Preview-available track prioritization
- **Instant Playback**: Automatic first-song selection

3. Weather Integration

- **Location Awareness**: GPS-based weather detection
- **Contextual Enhancement**: Weather-influenced recommendations
- **Visual Integration**: Clean weather widget display
- **Privacy Fallback**: Default location on permission denial

4. Modern User Experience

- **Responsive Design**: Cross-device compatibility
- **Progressive Enhancement**: Feature detection and fallbacks
- **Error Management**: User-friendly error messages
- **Performance Optimization**: Lazy loading and efficient rendering

---

Strengths

Technical Excellence

1. **Type Safety**: Comprehensive TypeScript implementation
2. **Modern Architecture**: Latest React patterns and best practices
3. **Performance**: Optimized bundle size and loading strategies
4. **Scalability**: Modular component architecture

User Experience

1. **Intuitive Interface**: Clear visual hierarchy and navigation

2. **Responsive Design**: Mobile-first approach with desktop optimization
3. **Accessibility**: Semantic HTML and keyboard navigation support
4. **Error Handling**: Graceful degradation with helpful messaging

Integration Quality

1. **API Robustness**: Comprehensive error handling and retries
2. **Real-time Processing**: Smooth emotion detection and music updates
3. **Privacy-First**: Local processing with minimal data transmission
4. **Cross-Platform**: Web-based deployment for universal access

Development Practices

1. **Code Organization**: Clear separation of concerns
2. **Reusable Components**: DRY principles with modular design
3. **Configuration Management**: Environment-based deployments
4. **Documentation**: Self-documenting code with TypeScript

Areas for Improvement

Performance Optimization

1. **Model Loading**: Implement progressive model loading for faster startup
2. **Bundle Splitting**: Code splitting for reduced initial load time
3. **Caching Strategy**: Implement service worker for offline functionality
4. **Memory Management**: Optimize audio object lifecycle

Feature Enhancements

1. **User Accounts**: Persistent preferences and listening history
2. **Playlist Creation**: Save mood-based playlists to Spotify
3. **Social Features**: Share mood snapshots and music discoveries
4. **Analytics**: Mood tracking and music preference insights

Technical Improvements

1. **Database Integration**: Replace in-memory storage with persistent database
2. **Authentication**: Implement user authentication system
3. **Rate Limiting**: API call optimization and caching
4. **Monitoring**: Application performance and error tracking

User Experience Refinements

1. **Onboarding**: Interactive tutorial for new users
2. **Customization**: User preference settings for music genres
3. **Accessibility**: Enhanced screen reader support
4. **Internationalization**: Multi-language support

Conclusion

EmotionMelody represents a sophisticated intersection of artificial intelligence, web development, and user experience design. The project successfully demonstrates:

Technical Mastery

- **Full-Stack Development**: Seamless integration of frontend and backend technologies
- **AI Integration**: Practical application of computer vision in web browsers
- **API Orchestration**: Complex third-party service coordination
- **Modern Web Standards**: Progressive web app capabilities

Innovation

- **Novel Concept**: Unique combination of emotion detection and music recommendation
- **Privacy-Conscious**: Client-side processing prioritizing user privacy
- **Cultural Awareness**: Emphasis on regional music preferences
- **Contextual Intelligence**: Weather integration for enhanced recommendations

Production Readiness

- **Error Handling**: Comprehensive fallback strategies
- **Performance**: Optimized for real-world usage patterns
- **Scalability**: Architecture supporting future enhancements
- **Maintainability**: Clean code structure for long-term development

The EmotionMelody project showcases advanced web development skills, creative problem-solving, and a deep understanding of user needs. It represents a portfolio piece that demonstrates both technical proficiency and innovative thinking in the modern web development landscape.

Interview Highlights

When presenting this project, emphasize:

1. **Technical Complexity**: Real-time AI processing in web browsers

2. **User Experience Focus**: Privacy-first design with intuitive interactions
3. **Problem-Solving**: Creative solutions for API limitations and error handling
4. **Modern Development**: Latest technologies and best practices implementation
5. **Scalable Architecture**: Foundation for future feature development

This project effectively demonstrates the ability to build complex, user-focused applications that integrate cutting-edge technologies while maintaining high standards of code quality and user experience.