# Sentiment Analysis Engine - Complete Project Documentation

## Table of Contents

## Project Overview

The Sentiment Analysis Engine is a comprehensive web application designed to analyze driver feedback and sentiment for transportation services. The system provides real-time sentiment analysis, user management, and detailed reporting capabilities.

### Key Features

- **Real-time Sentiment Analysis**: Process and analyze driver feedback using AI
- **User Authentication**: Role-based access control (Admin, Manager, Employee)
- **Dashboard**: Interactive charts and statistics
- **Feedback Management**: Submit, review, and track feedback
- **Driver Management**: CRUD operations for driver profiles
- **Alert System**: Real-time notifications for critical feedback

### Business Value

- Improve driver performance through feedback analysis
- Identify trends and patterns in customer satisfaction
- Enable data-driven decision making
- Automate sentiment classification and alerts

## System Architecture

### High-Level Architecture
```
┌─────────────────┐      ┌─────────────────┐      ┌───────
──────────┐
│  React Frontend   │      │ Spring Boot API    │      │ PostgreSQL DB    │
│  (Port 5173)        │◄───►│  (Port 8080)         │◄───►│  (Port 5432)         │
└─────────────────┘      └─────────────────┘      └───────
──────────┘
              │                            │                           │
   Vite Dev Server              JWT Authentication          Docker Container
   Material-UI                   Spring Security               Persistent Storage
   Axios HTTP Client            BCrypt Encryption           User Data & Feedback
```

### Component Architecture
```
Frontend (React)
 ├── Pages/
│    ├── LoginPage.jsx
│    ├── DashboardPage.jsx
│    ├── AdminPage.jsx
│    ├── DriversPage.jsx
│    └── FeedbackPage.jsx
```

```
 ├── Components/
│    ├── AuthContext.jsx
│    ├── ProtectedRoute.jsx
│    └── Layout/
 ├── Services/
│    ├── authService.js
│    ├── api.js
│    ├── userService.js
│    └── feedbackService.js
└── Hooks/
    ├── useFetchDrivers.js
    └── useFetchFeedback.js

Backend (Spring Boot)
 ├── Controllers/
│    ├── AuthController.java
│    ├── UserController.java
│    └── FeedbackController.java
 ├── Services/
│    ├── AuthenticationService.java
│    ├── UserService.java
│    └── FeedbackService.java
 ├── Repositories/
│    ├── UserRepository.java
│    └── FeedbackRepository.java
└── Security/
    ├── JwtTokenProvider.java
    └── SecurityConfig.java
```

## Technology Stack

### Frontend Technologies
- **React 18.2.0**: Modern JavaScript framework for building user interfaces
- **Vite**: Fast build tool and development server
- **Material-UI (MUI)**: React component library for consistent design
- **Axios**: HTTP client for API communication
- **React Router**: Client-side routing
- **React Hot Toast**: User notifications

### Backend Technologies
- **Spring Boot 3.1.0**: Java framework for enterprise applications
- **Spring Security**: Authentication and authorization
- **Spring Data JPA**: Object-relational mapping
- **JWT (JSON Web Tokens)**: Stateless authentication
- **BCrypt**: Password hashing
- **Maven**: Dependency management and build tool

### Database & Infrastructure
- **PostgreSQL 15**: Relational database management system
- **Docker**: Containerization for database
- **Docker Compose**: Multi-container orchestration

### Development Tools
- **Git**: Version control
- **VS Code**: Integrated development environment
- **Postman**: API testing
- **pgAdmin**: Database administration

## Database Schema

### Users Table
```sql
CREATE TABLE users (
    id BIGSERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    role VARCHAR(20) NOT NULL CHECK (role IN ('ADMIN', 'MANAGER', 'EMPLOYEE')),
    phone_number VARCHAR(15),
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_login_at TIMESTAMP
);
```

### Feedback Table
```sql
CREATE TABLE feedback (
    id BIGSERIAL PRIMARY KEY,
    entity_type VARCHAR(20) NOT NULL,
    entity_id BIGINT NOT NULL,
    feedback_text TEXT NOT NULL,
    rating INTEGER CHECK (rating BETWEEN 1 AND 5),
    sentiment_label VARCHAR(20),
    sentiment_score DECIMAL(3,2),
    source VARCHAR(50),
    status VARCHAR(20) DEFAULT 'PENDING',
    submitted_by BIGINT REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

    processed_at TIMESTAMP
);
~~~

Sample Data
```sql
-- Admin Users
INSERT INTO users (name, email, password, role) VALUES
('Admin User', 'admin@moveinsync.com', '$2a$10$hash...', 'ADMIN'),
('Manager User', 'manager@moveinsync.com', '$2a$10$hash...', 'MANAGER');

-- Employee Users (Drivers)
INSERT INTO users (name, email, password, role, phone_number) VALUES
('John Smith', 'john.employee@moveinsync.com', '$2a$10$hash...', 'EMPLOYEE',
'9876543210'),
('Mary Johnson', 'mary.employee@moveinsync.com', '$2a$10$hash...', 'EMPLOYEE',
'9876543211');
~~~

---

## Authentication System

### JWT Token Flow
```
1. User Login Request
   ↓
2. Backend Validates Credentials
   ↓
3. Generate JWT Token
   ↓
4. Return Token + User Info
   ↓
```

```
5. Frontend Stores Token
  ↓
6. Include Token in API Requests
  ↓
7. Backend Validates Token
  ↓
8. Process Authorized Request
```

### Authentication Service (Frontend)

```javascript
// authService.js
import { api } from './api';

export const authService = {
    // Login user and store JWT token
    async login(email, password) {
        const response = await api.post('/auth/login', { email, password });
        const { token, user } = response.data;

        // Store authentication data
        localStorage.setItem('jwt_token', token);
        localStorage.setItem('user_info', JSON.stringify(user));

        // Set default authorization header
        api.defaults.headers.common['Authorization'] = `Bearer ${token}`;

        return { user, token };
    },

    // Initialize auth state from localStorage
    initialize() {
        const token = localStorage.getItem('jwt_token');
        const userInfo = localStorage.getItem('user_info');
        if (token && userInfo) {
            api.defaults.headers.common['Authorization'] = `Bearer ${token}`;
        }
    },

    // Get current user from localStorage
    getCurrentUser() {
        const userInfo = localStorage.getItem('user_info');
        return userInfo ? JSON.parse(userInfo) : null;
    },

    // Logout and clear stored data
    logout() {
        localStorage.removeItem('jwt_token');
        localStorage.removeItem('user_info');
        delete api.defaults.headers.common['Authorization'];
    }
};
```

### Authentication Controller (Backend)

```java
// AuthController.java
@RestController
@RequestMapping("/api/auth")
public class AuthController {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private JwtTokenProvider tokenProvider;

    @Autowired
    private UserService userService;
    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody LoginRequest request) {
        try {
            // Authenticate user credentials
            Authentication auth = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(
                    request.getEmail(),
                    request.getPassword()
                )
            );

            // Get user details
            User user = userService.findByEmail(request.getEmail());

            // Generate JWT token
            String token = tokenProvider.generateToken(auth);

            // Update last login time
            user.setLastLoginAt(LocalDateTime.now());
            userService.save(user);

            // Return response
            LoginResponse response = new LoginResponse(token, user);
            return ResponseEntity.ok(response);

        } catch (BadCredentialsException e) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                .body(new ErrorResponse("Invalid credentials"));
        }
    }
}
```

### Protected Route Component

```jsx
// ProtectedRoute.jsx
```

```
import { Navigate } from 'react-router-dom';
import { useAuth } from '../contexts/AuthContext';

export const ProtectedRoute = ({ children, adminOnly = false }) => {
    const { isAuthenticated, isAdmin, loading } = useAuth();

    if (loading) {
        return <CircularProgress />;
    }

    if (!isAuthenticated) {
        return <Navigate to="/login" replace />;
    }

    if (adminOnly && !isAdmin()) {
        return <Navigate to="/feedback" replace />;
    }

    return children;
};
```

## Frontend Implementation

### Main Application Structure

```jsx
```

~~~
// App.jsx
{/* Public Routes */}
            <Route path="/login" element={<LoginPage />} />
 {/* Protected Admin Routes */}
            <Route path="/admin" element={
              <ProtectedRoute adminOnly>
                <MainLayout>
                  <AdminPage />
                </MainLayout>
              </ProtectedRoute>
            } />
{/* Protected Employee Routes */}
            <Route path="/feedback" element={
              <ProtectedRoute>
                <MainLayout>
                  <FeedbackPage />
                </MainLayout>
              </ProtectedRoute>
            } />
```
~~~

### Dashboard Implementation

### API Service Layer

## Backend API

### User Management Endpoints

~~~
```java
// UserController.java
~~~

```
 @GetMapping
    public ResponseEntity<Page<User>> getAllUsers(
            @RequestParam(defaultValue = "0") int page,
            @RequestParam(defaultValue = "20") int size,
            @RequestParam(defaultValue = "name,asc") String sort
    ) {
        Pageable pageable = PageRequest.of(page, size, Sort.by(sort));
        Page<User> users = userService.findAll(pageable);
        return ResponseEntity.ok(users);
    }

    @PostMapping
    public ResponseEntity<User> createUser(@Valid @RequestBody CreateUserRequest
request) {
        User user = userService.createUser(request);
        return ResponseEntity.status(HttpStatus.CREATED).body(user);
    }

    @GetMapping("/{id}")
    public ResponseEntity<User> getUserById(@PathVariable Long id) {
        User user = userService.findById(id);
        return ResponseEntity.ok(user);
    }

    @PutMapping("/{id}")
    public ResponseEntity<User> updateUser(
            @PathVariable Long id,
            @Valid @RequestBody UpdateUserRequest request
    ) {
        User updatedUser = userService.updateUser(id, request);
        return ResponseEntity.ok(updatedUser);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteUser(@PathVariable Long id) {
        userService.deleteUser(id);
        return ResponseEntity.noContent().build();
    }
}
```

### Feedback Management Endpoints

```java
// FeedbackController.java
public class FeedbackController {

    @PostMapping
    @PreAuthorize("hasAnyRole('ADMIN', 'EMPLOYEE')")
    public ResponseEntity<Feedback> submitFeedback(
            @Valid @RequestBody FeedbackRequest request,
            Authentication auth
    ) {
        User currentUser = userService.findByEmail(auth.getName());
        Feedback feedback = feedbackService.submitFeedback(request, currentUser);
        return ResponseEntity.status(HttpStatus.CREATED).body(feedback);
    }

    @GetMapping
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<Page<Feedback>> getAllFeedback(
            @RequestParam(defaultValue = "0") int page,
            @RequestParam(defaultValue = "20") int size
    ) {
        Pageable pageable = PageRequest.of(page, size);
        Page<Feedback> feedback = feedbackService.findAll(pageable);
        return ResponseEntity.ok(feedback);
    }

    @GetMapping("/driver/{driverId}")
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<Page<Feedback>> getDriverFeedback(
            @PathVariable Long driverId,
            @RequestParam(defaultValue = "0") int page,
            @RequestParam(defaultValue = "20") int size
    ) {
        Pageable pageable = PageRequest.of(page, size);
        Page<Feedback> feedback = feedbackService.findByDriverId(driverId, pageable);
        return ResponseEntity.ok(feedback);
    }
}
```

### Security Configuration

```java
// SecurityConfig.java
public class SecurityConfig {

    @Autowired
    private JwtAuthenticationEntryPoint authenticationEntryPoint;

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager authenticationManager(
            AuthenticationConfiguration config
    ) throws Exception {
        return config.getAuthenticationManager();
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .cors().and()
            .csrf().disable()
            .sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**").permitAll()
                .requestMatchers("/api/admin/**").hasRole("ADMIN")
                .anyRequest().authenticated()
            )
            .exceptionHandling()
                .authenticationEntryPoint(authenticationEntryPoint)
            .and()
            .addFilterBefore(
                jwtAuthenticationFilter(),
                UsernamePasswordAuthenticationFilter.class
            );

        return http.build();
    }
}
```

## User Management

### User Roles and Permissions

| Role | Permissions
|

ADMIN	Full system access, user management, all reports
MANAGER	View reports, manage drivers, limited admin functions
EMPLOYEE	Submit feedback, view own feedback, basic dashboard

### User Creation Process

```javascript
// userService.js
export const createUser = async (userData) => {
    return await api.post('/users', {
        name: userData.name,
        email: userData.email,
        role: userData.role,
        phoneNumber: userData.phoneNumber,
        password: generateDefaultPassword(userData.name)
    });
};

const generateDefaultPassword = (name) => {
    const namePart = name.replace(/\s+/g, '').toLowerCase();
    const numberPart = Math.floor(1000 + Math.random() * 9000);
    return `${namePart}${numberPart}`;
};
```

### Password Security

- **BCrypt Hashing**: All passwords encrypted with BCrypt (cost factor 10)
- **Default Passwords**: Auto-generated for new users
- **Password Reset**: Admin can reset user passwords
- **Security**: No plain text storage, secure hash comparison

## Installation & Setup

### Prerequisites

- **Node.js 18+**: Frontend development and build
- **Java 17+**: Backend development
- **Docker**: Database containerization
- **Git**: Version control

### Database Setup

### Backend Setup

### Frontend Setup

### Environment Configuration

#### Backend (application.properties)

```properties
# Database Configuration
spring.datasource.url=jdbc:postgresql://localhost:5432/sentiment_db
spring.datasource.username=sentiment_user
spring.datasource.password=sentiment_password

# JPA Configuration
spring.jpa.hibernate.ddl-auto=validate
spring.jpa.show-sql=false
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

# JWT Configuration
jwt.secret=mySecretKey
jwt.expiration=86400000

# Server Configuration
server.port=8080
server.servlet.context-path=/api

# CORS Configuration
cors.allowed-origins=http://localhost:5173
```

#### Frontend (vite.config.js)

```javascript
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';

export default defineConfig({
  plugins: [react()],
  server: {
    port: 5173,
    proxy: {
      '/api': {
        target: 'http://localhost:8080',
        changeOrigin: true,
        secure: false
      }
    }
  },
  build: {
    outDir: 'dist',
    sourcemap: true
  }
});
```

## Testing & Deployment

### Testing Strategy
1. **Unit Tests**: Individual component testing
2. **Integration Tests**: API endpoint testing

3. **End-to-End Tests**: Full user flow testing

4. **Security Tests**: Authentication and authorization

### Sample Test Cases

```java
// UserControllerTest.java
    @Test
    void testCreateUser() {
        CreateUserRequest request = new CreateUserRequest(
            "Test User",
            "test@test.com",
            "password123",
            "EMPLOYEE"
        );
```

### Production Deployment

```bash
# Build production bundles
npm run build
mvn clean package -DskipTests
# Docker deployment
docker-compose up -d
# Environment variables
export SPRING_PROFILES_ACTIVE=prod
export JWT_SECRET=production-secret-key
export DATABASE_URL=postgresql://prod-host:5432/sentiment_db
```

## Code Examples

### Complete Login Flow

```jsx
// LoginPage.jsx - Complete login implementation
```

```
import React, { useState } from 'react';
import { useAuth } from '../contexts/AuthContext';
import { useNavigate } from 'react-router-dom';
import {
  Container,
  Paper,
  TextField,
  Button,
  Typography,
  Alert
} from '@mui/material';

export const LoginPage = () => {
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [error, setError] = useState('');
    const [loading, setLoading] = useState(false);

    const { login } = useAuth();
    const navigate = useNavigate();

    const handleSubmit = async (e) => {
        e.preventDefault();
        setLoading(true);
        setError('');

        try {
            const user = await login(email, password);

            // Redirect based on user role
            if (user.role === 'ADMIN') {
                navigate('/admin');
            } else {
                navigate('/feedback');
            }
        } catch (err) {
            setError('Invalid email or password');
        } finally {
            setLoading(false);
        }
    };

    return (
        <Container maxWidth="sm" sx={{ mt: 8 }}>
            <Paper elevation={3} sx={{ p: 4 }}>
                <Typography variant="h4" align="center" gutterBottom>
                    Login
                </Typography>

                {error && (
                    <Alert severity="error" sx={{ mb: 2 }}>
                        {error}
                    </Alert>
                )}

                <form onSubmit={handleSubmit}>
                    <TextField
                        fullWidth
                        label="Email"
                        type="email"
                        value={email}
                        onChange={(e) => setEmail(e.target.value)}
                        margin="normal"
                        required
                    />

                    <TextField
                        fullWidth
                        label="Password"
                        type="password"
                        value={password}
                        onChange={(e) => setPassword(e.target.value)}
                        margin="normal"
                        required
                    />

                    <Button
                        type="submit"
                        fullWidth
                        variant="contained"
                        sx={{ mt: 3 }}
                        disabled={loading}
                    >
                        {loading ? 'Logging in...' : 'Login'}
                    </Button>
                </form>
            </Paper>
        </Container>
    );
};
```

### Custom React Hook
```javascript
// useFetchDrivers.js - Data fetching hook
import { useState, useEffect } from 'react';
import { driverService } from '../services/driverService';

export const useFetchDrivers = () => {
    const [drivers, setDrivers] = useState([]);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);

    useEffect(() => {
        const fetchDrivers = async () => {
            try {
                setLoading(true);
                const response = await driverService.getAllDrivers();
                setDrivers(response.data);
                setError(null);
            } catch (err) {
                console.error('Failed to fetch drivers:', err);
                setError('Failed to load drivers');
                setDrivers([]);
            } finally {
                setLoading(false);
            }
        };

        fetchDrivers();
    }, []);

    const refetch = () => {
        fetchDrivers();
    };

    return { drivers, loading, error, refetch };
};
```

## Troubleshooting

### Common Issues and Solutions

#### 1. Authentication Issues

Problem: Login fails with 401 Unauthorized

Solution:

```bash
# Check password hashes in database
docker exec -it sentiment-postgres psql -U sentiment_user -d sentiment_db
\x on
SELECT email, length(password) as pwd_len FROM users WHERE email =
'admin@moveinsync.com';


# Verify hash format (should be 60 characters starting with $2a$)
```

#### 2. CORS Issues

Problem: Frontend can't connect to backend

Solution:

```java
// Add CORS configuration in SecurityConfig.java
@Bean
public CorsConfigurationSource corsConfigurationSource() {
    CorsConfiguration configuration = new CorsConfiguration();
    configuration.setAllowedOriginPatterns(Arrays.asList("*"));
    configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE"));
    configuration.setAllowedHeaders(Arrays.asList("*"));
    configuration.setAllowCredentials(true);

    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration("/**", configuration);
    return source;
}
```

#### 3. Database Connection Issues

Problem: Backend can't connect to PostgreSQL

Solution:

```bash
# Check Docker container status
docker ps
docker logs sentiment-postgres

# Test database connection
docker exec -it sentiment-postgres psql -U sentiment_user -d sentiment_db -c "\l"
```

#### 4. Build Issues
Problem: Maven build fails
Solution:
```bash
# Clean and rebuild
mvn clean
mvn compile
mvn package -DskipTests

# Check Java version
java -version
mvn -version
```

#### 5. Frontend Issues
Problem: React app won't start
Solution:
```bash
# Clear node modules and reinstall
rm -rf node_modules package-lock.json
npm install

# Check Node version
```

node -version

npm -version

### Performance Optimization Tips

1. **Database**: Add indexes on frequently queried columns
2. **Frontend**: Implement lazy loading for large datasets
3. **Backend**: Use pagination for API responses
4. **Caching**: Implement Redis for session storage
5. **Security**: Regular security audits and dependency updates

### Monitoring and Logging

```java
// Add structured logging
@Slf4j
@RestController
public class AuthController {

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody LoginRequest request) {
        log.info("Login attempt for email: {}", request.getEmail());

        try {
            // Authentication logic
            log.info("Login successful for user: {}", user.getEmail());
        } catch (BadCredentialsException e) {
            log.warn("Login failed for email: {}", request.getEmail());
        }
    }
}
```

## Conclusion

This documentation provides a comprehensive overview of the Sentiment Analysis Engine project, including:

- **Complete system architecture** with clear component relationships
- **Detailed implementation examples** for both frontend and backend
- **Security best practices** with JWT authentication and role-based access
- **Database design** with proper relationships and constraints
- **Deployment instructions** for development and production environments
- **Troubleshooting guides** for common issues

The system is designed for scalability, maintainability, and security, following modern web development best practices and enterprise-grade architecture patterns.

For additional support or feature requests, please refer to the project repository or contact the development team.

Document Version: 1.0

Last Updated: November 11, 2025

Author: Yash Sharma