CS-AI: AI-Powered Operations Platform

Technical Documentation and Architecture Guide

Table of Contents

1. Introduction

CS-AI is an AI-powered operations platform designed to automate and optimize support, compliance, and operational workflows. The platform achieves:

- 95%+ CSAT scores through AI-powered support automation
- Streamlined compliance processes with automated checks
- Efficient operational task management
- Real-time analytics and insights

Key Features

Support Automation

- Automated ticket routing and classification
- AI-powered response generation
- Sentiment analysis for customer interactions
- Real-time CSAT monitoring

Compliance Management

- Automated document verification
- Real-time compliance checks
- Audit trail generation
- Regulatory requirement tracking

Operations Optimization

- Workflow automation
- Resource allocation
- Performance analytics
- Task prioritization

2. System Architecture

High-Level Architecture

```
[Client Browsers] ←→ [CDN] ←→ [Frontend (React)] ←→ [Load Balancer] ←→ [Backend API
Cluster]
                                                                      ↕
                                                               [Service Layer]
                                                                      ↕
                                                          [Data Access Layer]
                                                                      ↕
                                                     [Database & Cache Layer]
```

Technology Stack

Frontend

- **Framework**: React.js
- **UI Library**: Material-UI (MUI)
- **State Management**: Context API
- **Routing**: React Router v6
- **Animations**: Framer Motion
- **HTTP Client**: Axios
- **Form Handling**: React Hook Form

Backend

- **Runtime**: Node.js
- **Framework**: Express.js
- **ORM**: Prisma
- **Database**: PostgreSQL
- **Caching**: Redis
- **Authentication**: JWT
- **API Documentation**: Swagger/OpenAPI

3. Frontend Architecture

Component Structure

```
src/
├── components/
│   ├── common/          # Reusable UI components
│   │   ├── LoadingScreen
│   │   ├── Logo
│   │   └── WorkflowDiagram
│   │
│   ├── dashboard/       # Dashboard-specific components
│   │   ├── Agents
│   │   ├── Sidebar
│   │   └── TopNav
│   │
│   └── layout/         # Layout components
│       └── DashboardLayout
│
├── views/              # Page components
│   ├── Landing
│   ├── Login
│   ├── Dashboard
│   └── Settings
│
├── routes/            # Routing configuration
├── theme/             # MUI theme customization
└── services/          # API services
```

Key Components

WorkflowDiagram Component

```
// Animated SVG diagram showing AI workflow
const WorkflowDiagram = () => {
  // Animation variants
  const pathVariants = {
    hidden: { pathLength: 0 },
    visible: { pathLength: 1 }
  };

  return (
    <motion.svg>
      {/* Input Node */}
      {/* AI Processing Node */}
      {/* Output Node */}
    </motion.svg>
  );
};
```

Dashboard Layout

```
const DashboardLayout = ({ children }) => {
  return (
    <Box sx={{ display: 'flex' }}>
      <Sidebar />
      <Box component="main">
        <TopNav />
        {children}
      </Box>
    </Box>
  );
};
```

State Management

- Context API for global state
- Local state with useState for component-level state
- Custom hooks for reusable logic

Routing Structure

```
const routes = [
  {
    path: '/',
    element: <Landing />
  },
  {
    path: '/dashboard',
    element: <DashboardLayout>,
    children: [
      { path: 'overview', element: <Overview /> },
      { path: 'agents', element: <Agents /> }
    ]
  }
];
```

4. Backend Architecture

Layer Architecture

```
src/
├── routes/           # Route handlers
├── controllers/      # Business logic
├── services/        # Service layer
├── models/          # Data models
├── middleware/      # Custom middleware
└── utils/           # Utility functions
```

API Structure

Authentication Routes

```
router.post('/auth/login', authController.login);
router.post('/auth/refresh', authController.refreshToken);
router.post('/auth/logout', authController.logout);
```

Support Routes

```
router.post('/support/analyze', supportController.analyzeTicket);
router.get('/support/metrics', supportController.getMetrics);
```

Compliance Routes

```
router.post('/compliance/check', complianceController.checkDocument);
router.get('/compliance/audit-trail', complianceController.getAuditTrail);
```

Middleware Implementation

Authentication Middleware

```
const authMiddleware = async (req, res, next) => {
  try {
    const token = req.headers.authorization?.split(' ')[1];
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (error) {
    res.status(401).json({ message: 'Unauthorized' });
  }
};
```

Rate Limiting

```
const rateLimiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 100
});
```

5. Database Schema

Core Tables

Users

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  role VARCHAR(50) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Support Tickets

```
CREATE TABLE support_tickets (
  id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users(id),
  title VARCHAR(255) NOT NULL,
  content TEXT NOT NULL,
  status VARCHAR(50) NOT NULL,
  priority VARCHAR(50) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Compliance Records

```
CREATE TABLE compliance_records (
  id SERIAL PRIMARY KEY,
  document_type VARCHAR(100) NOT NULL,
  status VARCHAR(50) NOT NULL,
  verified_at TIMESTAMP,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

6. API Documentation

Authentication

Login

```
POST /api/v1/auth/login
Body: {
  "email": "user@example.com",
  "password": "password123"
}
Response: {
  "token": "jwt_token",
  "refreshToken": "refresh_token"
}
```

Support Ticket Analysis

```
POST /api/v1/support/analyze
Body: {
  "ticketId": "123",
  "content": "ticket_content"
}
Response: {
  "sentiment": "positive",
  "priority": "high",
  "suggestedResponse": "..."
}
```

7. Security Implementation

Authentication Flow

1. User submits credentials
2. Server validates and issues JWT
3. Client stores token securely
4. Token included in Authorization header
5. Server validates token on protected routes

Data Protection

- Password hashing using bcrypt
- Environment variable encryption
- HTTPS enforcement
- XSS protection
- CSRF tokens

8. Deployment Guide

Prerequisites

- Node.js v16+
- PostgreSQL 13+
- Redis 6+
- SSL certificate

Environment Setup

1. Configure environment variables
2. Set up database
3. Initialize Redis
4. Configure SSL

Deployment Steps

1. Build frontend
2. Deploy backend services
3. Configure nginx
4. Set up monitoring
5. Enable backups

9. Monitoring and Analytics

Key Metrics

- Response time
- Error rates
- CSAT scores
- System usage

- API performance

Logging

- Application logs
- Error tracking
- User activity
- Performance metrics

10. Scaling Considerations

Horizontal Scaling

- Load balancing
- Database replication
- Cache distribution
- Microservices architecture

Performance Optimization

- Code splitting
- Lazy loading
- Cache strategies
- Database indexing