🧠 Mental Health Bot - AI-Powered Support System

FASTAPI MONGODB OPENAI PYTHON

📋 Table of Contents

🎯 Project Overview

The Mental Health Bot is an AI-powered conversational agent designed to provide personalized mental health support through intelligent conversations. Built with FastAPI and powered by OpenAI's GPT-4o, it offers context-aware responses based on user check-in data and conversation history.

✨ Key Features

- 🤖 **AI-Powered Conversations**: GPT-4o powered responses with therapeutic frameworks (CBT, DBT, Mindfulness)
- 📊 **Context-Aware Support**: Integrates daily check-in data with conversation history
- 🔐 **Secure API Authentication**: API key-based authentication for server-to-server communication

- 🗄️ **MongoDB Integration**: Scalable NoSQL database with connection pooling
- 📝 **Chat Summarization**: Automatic generation of conversation summaries
- 🏥 **Patient Management**: Comprehensive patient data and check-in tracking
- ⚡ **High Performance**: Async/await architecture with connection pooling

🎭 AI Personality: "Kay"

Kay is an empathetic AI companion from KindPath that provides: - **Age-appropriate responses** (Gen Z, Millennials, Gen X/Boomers) - **Therapeutic approaches** (CBT, DBT, Mindfulness, Emotion Regulation) - **Contextual understanding** of patient's mental state and history - **Supportive conversation flow** with validation and gentle guidance

---

🏗️ Architecture

System Architecture Diagram

```
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│   Node.js       │    │   FastAPI       │    │   MongoDB       │
│   Server        │◄──►│   Backend       │◄──►│   Database      │
│   (Client)      │    │   (AI Service)  │    │   (Atlas)       │
└─────────────────┘    └─────────────────┘    └─────────────────┘
        │                      │                      │
        │                      ▼                      │
        │              ┌─────────────────┐            │
        │              │   OpenAI API    │            │
        │              │   (GPT-4o)      │            │
        │              └─────────────────┘            │
        ▼                                             ▼
┌─────────────────┐                          ┌─────────────────┐
│   API Key       │                          │   Connection    │
│   Authentication│                          │   Pooling       │
└─────────────────┘                          └─────────────────┘
```

Component Flow

1. 🔑 **Authentication**: Node.js server authenticates with API key
2. 📊 **Context Building**: Retrieves patient check-in data and chat history
3. 🤖 **AI Processing**: GPT-4o generates personalized responses
4. 💾 **Data Persistence**: Saves conversations and updates summaries
5. 📤 **Response Delivery**: Returns structured JSON response

🛠️ Technology Stack

Backend

- **Framework**: FastAPI (Python 3.11+)
- **Database**: MongoDB Atlas with Motor (async driver)
- **AI/LLM**: OpenAI GPT-4o via LangChain
- **Authentication**: API Key-based authentication
- **Connection Pooling**: Motor with optimized pool settings
- **Validation**: Pydantic for data validation

Infrastructure

- **Database**: MongoDB Atlas (cloud-hosted)
- **Environment**: Python virtual environment
- **Logging**: Structured logging with timestamps
- **Testing**: HTTP test files with REST Client

Dependencies

```
# Core Framework
fastapi
uvicorn[standard]
pydantic
pydantic_settings

# Database
motor          # Async MongoDB driver
pymongo        # MongoDB utilities

# AI/LLM
langchain_openai
langchain_core

# Authentication
python-jose[cryptography]
passlib[bcrypt]

# Utilities
rich           # Beautiful terminal output
redis          # Caching (if needed)
```

🗄️ Database Schema

MongoDB Collections

1. `dailycheckins` Collection

```
{
  "_id": ObjectId,
  "patient": ObjectId,           // Reference to patient
  "checkinType": "Morning|Evening",
  "mentalState": String,
  "energyLevel": Number,
  "sleepQuality": Number,
  "stressLevel": Number,
  "mood": String,
  "anxietyLevel": Number,
  "depressionLevel": Number,
  "executiveTasks": [String],
  "riskLevel": "Low|Moderate|High",
  "message": String,             // AI-generated summary
  "createdAt": Date,
  "updatedAt": Date
}
```

2. `chats` Collection

```
{
  "_id": ObjectId,
  "patient": ObjectId,           // Reference to patient
  "query": String,               // User's message
  "response": String,            // AI's response
  "createdAt": Date,
  "updatedAt": Date
}
```

3. `users` Collection (if needed)

```
{
  "_id": ObjectId,
  "firstName": String,
```

```
  "lastName": String,
  "email": String,
  "age": Number,
  "gender": String,
  "createdAt": Date,
  "updatedAt": Date
}
```

🔌 API Endpoints

Public Endpoints (No Authentication)

Health Check

```
GET /agent/health
```

Response:

```
{
  "status": "healthy",
  "service": "mental-health-agent",
  "api_auth": {
    "api_key_configured": true,
    "api_key_header_name": "X-API-Key",
    "api_key_length": 32,
    "api_key_prefix": "ENEwcpSg..."
  }
}
```

Database Health Check

```
GET /agent/health/db
```

Response:

```
{
  "status": "healthy",
  "database": "connected",
  "connection_pool": {
    "server_version": "7.0.4",
```

```
    "connection_pool_configured": true,
    "max_pool_size": 50,
    "min_pool_size": 5,
    "status": "connected"
  },
  "timestamp": "2025-01-07T20:16:37Z"
}
```

Protected Endpoints (API Key Required)

Kay Bot Conversation

```
POST /agent/kay-bot
X-API-Key: your_api_key_here
Content-Type: application/json

{
  "age": "25",
  "gender": "Female",
  "name": "Sarah",
  "patient_id": "6899521238bcd98456d965e0",
  "message": "I'm feeling anxious today"
}
```

Response:

```
{
  "response": "I hear that you're feeling anxious today, Sarah...",
  "patient_id": "6899521238bcd98456d965e0",
  "chat_saved": true,
  "chat_id": "68aa2d7499cec0c3b0d53a00"
}
```

Chat Summary Generation

```
GET /agent/chat/summary/{patient_id}
X-API-Key: your_api_key_here
```

Response:

```
{
  "patient_id": "6899521238bcd98456d965e0",
```

```
  "document_id": "68aa2d7499cec0c3b0d53a00",
  "summary": "The patient reported feeling anxious during the evening check-in...",
  "update_success": true
}
```

Authentication Test

```
GET /agent/auth/test
X-API-Key: your_api_key_here
```

Response:

```
{
  "status": "success",
  "message": "API key authentication successful",
  "timestamp": "2025-01-07T20:16:37Z"
}
```

🔐 Authentication & Security

API Key Authentication

The system uses **API Key authentication** for secure server-to-server communication between your Node.js server and the FastAPI backend.

How It Works

1. **Key Generation**: Generate a secure API key using the provided script
2. **Key Exchange**: Securely share the key with your Node.js developer
3. **Header Authentication**: Include the key in the `X-API-Key` header
4. **Validation**: FastAPI validates the key before processing requests

Node.js Integration Example

```
const axios = require('axios');

const FASTAPI_URL = 'http://localhost:8000';
const API_KEY = process.env.FASTAPI_API_KEY;
```

```
async function callKayBot(payload) {
  try {
    const response = await axios.post(`${FASTAPI_URL}/agent/kay-bot`, payload, {
      headers: {
        'Content-Type': 'application/json',
        'X-API-Key': API_KEY  // The secret key
      }
    });
    return response.data;
  } catch (error) {
    console.error('FastAPI call failed:', error.response?.data || error.message);
    throw error;
  }
}
```

Error Responses

Missing API Key (401):

```
{
  "detail": "API key is required. Please provide X-API-Key header."
}
```

Invalid API Key (401):

```
{
  "detail": "Invalid API key provided."
}
```

🏢 Services

1. Database Service (`services/db_service.py`)

Handles all MongoDB operations with connection pooling:

```
class DatabaseService:
    @staticmethod
    async def get_patient_checkin_context(patient_id: str) -> Dict[str, Any]:
        """Get most recent check-in data for context building"""

    @staticmethod
    async def get_patient_recent_chats(patient_id: str, limit: int = 20) -> Dict[str,
```

```
        """Get recent chat history with formatted context"""

    @staticmethod
    async def save_chat_message(patient_id: str, query: str, response: str) -> Dict[s
        """Save new conversation to database"""

    @staticmethod
    async def add_checkin_summary(document_id: str, summary_message: str) -> Dict[str
        """[illegible]"""
```

2. OpenAI Service (`services/openai_service.py`)

Manages AI interactions with GPT-4o:

```
class LLMService:
    async def get_chat_summary(self, context_string: str) -> str:
        """Generate conversation summary using GPT-4o"""

    async def generate_kay_response(
        self,
        user_message: str,
        patient_name: str,
        patient_age: str,
        patient_gender: str,
        checkin_context: str,
        conversational_context: str
    ) -> str:
        """Generate personalized Kay bot response"""
```

3. API Authentication Service (`services/api_auth_service.py`)

Handles API key validation and security:

```
class APIAuthService:
    @staticmethod
    async def verify_api_key(api_key: str) -> str:
        """Verify API key from request header"""

    @staticmethod
    def is_api_key_configured() -> bool:
        """Check if API key is properly configured"""

    @staticmethod
    def get_api_key_info() -> dict:
        """Get API key configuration information"""
```

⚙️ Configuration

Environment Variables

Create a `.env` file in the project root:

```
# MongoDB Configuration
MONGODB_URL=mongodb+srv://username:password@cluster.mongodb.net/?retryWrites=true&w=m
MONGODB_DATABASE=KindPath_DB

# OpenAI Configuration
OPENAI_API_KEY=sk-your-openai-api-key-here

# API Authentication
SERVER_API_KEY=your-secret-api-key-here

# Server Configuration (Optional)
HOST=0.0.0.0
PORT=8000
RELOAD=true
```

Configuration Class (`config.py`)

```
class Settings(BaseSettings):
    # API settings
    api_title: str = "Mental Health Bot API"
    api_description: str = "A FastAPI application for mental health support"
    api_version: str = "1.0.0"

    # Server settings
    host: str = "0.0.0.0"
    port: int = 8000
    reload: bool = True

    # CORS settings
    cors_origins: List[str] = ["*"]
    cors_allow_credentials: bool = True
    cors_allow_methods: List[str] = ["*"]
    cors_allow_headers: List[str] = ["*"]

    # OpenAI settings
    openai_api_key: str = ""
```

```
    # MongoDB settings
    mongodb_url: str
    mongodb_database: str

    # API Authentication settings
    server_api_key: str = ""
```

🚀 Installation & Setup

Prerequisites

- Python 3.11+
- MongoDB Atlas account
- OpenAI API key
- Git

1. Clone Repository

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

2. Create Virtual Environment

```
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
```

3. Install Dependencies

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

4. Generate API Key

```
python -c "import secrets; print('SERVER_API_KEY=' + secrets.token_urlsafe(32))"
```

5. Configure Environment

Create `.env` file with your configuration:

```
MONGODB_URL=mongodb+srv://akausar_db_user:9rsKr68RjYbrBJgr@kindpath-clustor.gyyafs0.r
MONGODB_DATABASE=KindPath_DB
OPENAI_API_KEY=sk-your-openai-key-here
SERVER_API_KEY=your-generated-api-key-here
```

6. Start Server

```
uvicorn main:app --reload --host 0.0.0.0 --port 8000
```

7. Verify Installation

```
# Test health endpoint
curl http://localhost:8000/agent/health

# Test database connection
curl http://localhost:8000/agent/health/db
```

🧪 Testing

Test Files

The project includes comprehensive test files in the `testing/` directory:

1. API Authentication Tests (`api_auth_tests.http`)

```
### Test with valid API key
GET http://localhost:8000/agent/auth/test
X-API-Key: your_api_key_here

### Test with invalid API key
GET http://localhost:8000/agent/auth/test
X-API-Key: invalid_key
```

2. Kay Bot Tests (`kay_bot_tests.http`)

```
### Test Kay bot conversation
POST http://localhost:8000/agent/kay-bot
Content-Type: application/json
X-API-Key: your_api_key_here

{
    "age": "25",
    "gender": "Female",
    "name": "Sarah",
    "patient_id": "6899521238bcd98456d965e0",
    "message": "I'm feeling anxious today"
}
```

3. Chat Summary Tests (`chat_summary_tests.http`)

```
### Test chat summary generation
GET http://localhost:8000/agent/chat/summary/6899521238bcd98456d965e0
X-API-Key: your_api_key_here
```

4. Connection Pool Tests (`connection_pool_tests.http`)

```
### Test connection pooling
GET http://localhost:8000/agent/health/db
```

Running Tests

1. **Install REST Client Extension** in VS Code
2. **Update API keys** in test files
3. **Run individual tests** by clicking "Send Request"
4. **Verify responses** match expected formats

🚀 Deployment

Production Configuration

1. **Environment Variables**: Set production values

2. **API Keys**: Use strong, unique keys
3. **CORS**: Configure appropriate origins
4. **Logging**: Set appropriate log levels
5. **Connection Pooling**: Optimize for production load

Docker Deployment (Optional)

```
FROM python:3.11-slim

WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .
EXPOSE 8000

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Health Monitoring

Monitor these endpoints in production:

- `GET /agent/health` - Basic health check
- `GET /agent/health/db` - Database and connection pool status

📚 API Documentation

Interactive Documentation

FastAPI automatically generates interactive API documentation:

- **Swagger UI**: `http://localhost:8000/docs`
- **ReDoc**: `http://localhost:8000/redoc`

API Reference

Request/Response Models

KayBotPayload:

```
class KayBotPayload(BaseModel):
    age: str
    gender: str
    name: str
    patient_id: str
    message: str
```

ChatSummaryResponse:

```
class ChatSummaryResponse(BaseModel):
    patient_id: str
    document_id: str
    summary: str
    update_success: bool
```

Error Handling

All endpoints return consistent error responses:

```
{
  "detail": "Error message",
  "status_code": 400
}
```

🔧 Troubleshooting

Common Issues

1. MongoDB Connection Issues

```
# Check connection string format
# Verify network access in MongoDB Atlas
# Check firewall settings
```

2. API Key Authentication Failures

```
# Verify SERVER_API_KEY is set in .env
# Check X-API-Key header is included
```

```
# Ensure key matches on both sides
```

3. OpenAI API Issues

```
# Verify OPENAI_API_KEY is valid
# Check API quota and billing
# Ensure model access (GPT-4o)
```

4. Connection Pool Issues

```
# Check MongoDB Atlas connection limits
# Monitor connection pool stats via /agent/health/db
# Adjust pool settings if needed
```

Debug Mode

Enable debug logging:

```
import logging
logging.basicConfig(level=logging.DEBUG)
```

Logs

Check application logs for detailed error information:

```
# Application logs include:
# - Database connection status
# - API authentication attempts
# - OpenAI API calls
# - Error details and stack traces
```

📈 Performance & Scalability

Connection Pooling

- **Max Pool Size**: 50 connections
- **Min Pool Size**: 5 connections

- **Idle Timeout**: 30 seconds
- **Queue Timeout**: 5 seconds

Optimization Tips

1. **Database Indexing**: Ensure proper indexes on patient_id fields
2. **Response Caching**: Consider Redis for frequently accessed data
3. **Rate Limiting**: Implement rate limiting for production
4. **Monitoring**: Set up application performance monitoring

🤝 Contributing

Development Setup

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Add tests for new functionality
5. Submit a pull request

Code Style

- Follow PEP 8 guidelines
- Use type hints
- Add docstrings to functions
- Include error handling

📄 License

🆘 Support

For support and questions:

1. Check the troubleshooting section

2. Review the API documentation
3. Check application logs
4. Create an issue in the repository

🎯 Roadmap

Planned Features

- [ ] **User Authentication**: JWT-based user authentication
- [ ] **Analytics Dashboard**: Conversation analytics and insights
- [ ] **Multi-language Support**: Support for multiple languages
- [ ] **Voice Integration**: Voice-to-text and text-to-speech
- [ ] **Mobile App**: Native mobile application
- [ ] **Advanced AI**: Custom model fine-tuning

Technical Improvements

- [ ] **GraphQL API**: More flexible data querying
- [ ] **Microservices**: Service decomposition
- [ ] **Event Streaming**: Real-time analytics
- [ ] **Machine Learning**: Custom model training

Built with ❤️ for mental health support

This system is designed to provide supportive conversations and should not replace professional mental health care. Always encourage users to seek professional help when needed.