AI Mental Health Assistant - Full Stack Architecture Overview

🏗️ **System Architecture Overview**

Technology Stack
- **Frontend:** Flutter Web (Dart) with Material Design 3
- **Backend:** Flask (Python) with SQLAlchemy ORM
- **Database:** PostgreSQL with Redis for session management
- **AI Integration:** Gemini API (Google) with fallback to OpenAI/Perplexity
- **Deployment:** Docker containers with Nginx reverse proxy
- **Production:** Render cloud platform
- **Crisis Detection:** Custom keyword-based system with geography-specific resources

---

🎨 **Frontend Architecture (Flutter Web)**

Core Application Structure

Main Application Entry (`main.dart`)
```dart
// Application Configuration
class MyApp extends StatelessWidget {
  // Material Design 3 Theme
  // Color Scheme: Primary (#667EEA), Secondary (#FF6B6B)
  // MultiProvider setup for state management
}

// Home Page Structure
class HomePage extends StatefulWidget {
  // Bottom Navigation: Chat | Mood Tracker
  // App Bar: Assessment | Tasks | Community buttons
  // IndexedStack for screen management
}
```

State Management (Provider Pattern)
```dart
// Provider Configuration
MultiProvider(
  providers: [
    ChangeNotifierProvider(create: (_) => ChatProvider()),
    ChangeNotifierProvider(create: (_) => MoodProvider()),
    ChangeNotifierProvider(create: (_) => AssessmentProvider()),
    ChangeNotifierProvider(create: (_) => TaskProvider()),
    ChangeNotifierProvider(create: (_) => ProgressProvider()),
  ]
)
```

Core Widgets & Components

1. Chat Interface (`chat_message_widget.dart`)
```dart
class ChatMessageWidget extends StatelessWidget {
  // Message display with user/AI distinction
  // Crisis detection integration
  // Geography-specific crisis resources
  // Risk level visualization
  // Timestamp display
}
```

2. Crisis Resources Widget (`crisis_resources.dart`)
```dart
class CrisisResourcesWidget extends StatelessWidget {
  // Risk level-based styling (none, low, medium, high)
  // Geography-specific helpline numbers
  // One-click calling/texting functionality
  // Crisis message display
  // Color-coded urgency levels
}
```

3. Mood Tracker (`mood_tracker.dart`)
```dart
class MoodTrackerWidget extends StatefulWidget {
  // 5-point mood scale (1-5)
  // Daily mood entry system
  // Mood history visualization
  // Trend analysis
  // Progress tracking
}
```

4. Self-Assessment (`self_assessment_screen.dart`)
```dart
class SelfAssessmentScreen extends StatefulWidget {
  // 6 wellness dimensions
  // 1-10 scale for each dimension
  // Progress visualization
  // Personalized recommendations
}
```

Data Models

Message Model (`message.dart`)
```dart
class Message {
  // Core fields
  final String id;
  final String content;
  final bool isUser;
  final DateTime timestamp;
```

```
  final MessageType type;
  final RiskLevel riskLevel;

  // Crisis-specific fields
  final String? crisisMsg;
  final List<Map<String, dynamic>>? crisisNumbers;

  // Styling methods
  Color getMessageColor(BuildContext context)
  Color getTextColor(BuildContext context)
}

enum RiskLevel { none, low, medium, high }
enum MessageType { text, error, system }
```

Mood Entry Model (`mood_entry.dart`)
```dart
class MoodEntry {
  final String id;
  final int moodLevel; // 1-5
  final String? note;
  final DateTime timestamp;
  final String sessionId;
}
```

State Providers

Chat Provider (`chat_provider.dart`)
```dart
class ChatProvider extends ChangeNotifier {
  List<Message> messages = [];
  bool isLoading = false;

  // Methods
  Future<void> sendMessage(String message, {String? country})
  void addMessage(Message message)
  void setLoading(bool loading)
}
```

Mood Provider (`mood_provider.dart`)
```dart
class MoodProvider extends ChangeNotifier {
  List<MoodEntry> moodEntries = [];

  // Methods
  Future<void> addMoodEntry(int moodLevel, String? note)
  Future<void> loadMoodHistory()
  double getAverageMood()
}
```

API Service Layer (`api_service.dart`)
```dart
class ApiService {
  // Dynamic environment detection
  static String get baseUrl {
    // Auto-detect local vs production
    // localhost:5055 for development
    // ai-mental-health-assistant-tddc.onrender.com for production
  }

  // API endpoints
  Future<Message> sendMessage(String message, {String? country})
  Future<List<MoodEntry>> getMoodHistory()
  Future<void> addMoodEntry(int moodLevel, String? note)
  Future<void> submitSelfAssessment(Map<String, int> assessment)
}
```

Configuration (`api_config.dart`)
```dart
class ApiConfig {
  static const String localUrl = 'http://localhost:5055';
  static const String productionUrl = 'https://ai-mental-health-assistant-tddc.onrender.com';

  static String get baseUrl {
    // Dynamic environment detection
    // Automatic local vs production switching
  }
}
```

🔧 **Backend Architecture (Flask)**

Core Application Structure

Main Application (`app.py`)
```python
# Application Factory Pattern
def create_app() -> Flask:
    app = Flask(__name__, static_folder='static', static_url_path='')
    # Database configuration
    # Session management
    # CORS setup
    # Rate limiting
    # Route registration
```

Configuration System
```python
class Config:
    # Environment detection (local, docker, render)
```

```
    ENVIRONMENT = os.getenv('ENVIRONMENT', 'local')
    RENDER = os.getenv('RENDER', 'false').lower() == 'true'
    DOCKER_ENV = os.getenv('DOCKER_ENV', 'false').lower() == 'true'

    # Database configuration
    DATABASE_URL = # Dynamic based on environment

    # AI Provider configuration
    GEMINI_API_KEY = os.getenv('GEMINI_API_KEY')
    OPENAI_API_KEY = os.getenv('OPENAI_API_KEY')
    PPLX_API_KEY = os.getenv('PPLX_API_KEY')
    AI_PROVIDER = os.getenv('AI_PROVIDER', 'gemini')
```

API Endpoints

Core Chat Endpoint
```python
@app.route("/api/chat", methods=["POST"])
@limiter.limit("30 per minute")
def chat():
    # Request processing
    # Crisis detection
    # AI response generation
    # Geography-specific resources
    # Response formatting
```

Mood Tracking Endpoints
```python
@app.route('/api/mood_entry', methods=['POST'])
def add_mood_entry():
    # Mood level validation
    # Database storage
    # Session management

@app.route('/api/mood_history', methods=['GET'])
def get_mood_history():
    # Session-based retrieval
    # Data formatting
    # Privacy protection
```

Self-Assessment Endpoints
```python
@app.route('/api/self_assessment', methods=['POST'])
def submit_self_assessment():
    # 6-dimensional assessment
    # Score calculation
    # Recommendation generation

@app.route('/api/wellness_recommendations', methods=['GET'])
def wellness_recommendations():
    # Personalized recommendations
```

```
    # Mood-based suggestions
    # Progress tracking
```

Crisis Detection Endpoint
```python
@app.route('/api/crisis_detection', methods=['POST'])
def crisis_detection():
    # Keyword analysis
    # Risk level assessment
    # Geography-specific resources
    # Immediate response generation
```

Crisis Detection System

Crisis Detection Logic (`crisis_detection.py`)
```python
def detect_crisis_level(message: str) -> Tuple[str, float, List[str]]:
    # Keyword analysis
    # Risk scoring algorithm
    # Pattern recognition
    # Return: (risk_level, score, keywords)
```

Geography-Specific Resources
```python
CRISIS_RESOURCES_BY_COUNTRY = {
    'in': {  # India
        'crisis_msg': "You're not alone. Help is available 24/7.",
        'crisis_numbers': [
            {'name': 'iCall Helpline', 'number': '022-25521111'},
            {'name': 'AASRA', 'number': '91-22-27546669'},
            {'name': 'Crisis Text Line', 'text': 'HOME to 741741'}
        ]
    },
    'us': {  # United States
        'crisis_msg': "You're not alone. Help is available 24/7.",
        'crisis_numbers': [
            {'name': 'National Suicide Prevention Lifeline', 'number': '988'},
            {'name': 'Crisis Text Line', 'text': 'HOME to 741741'}
        ]
    },
    # Additional countries: uk, ca, au, de, fr, jp, br, mx, generic
}
```

AI Integration

Gemini Provider (`providers/gemini.py`)
```python
def get_gemini_response(message: str, conversation_history: List, risk_level: str = None) -> str:
    # Crisis-aware response generation
    # Risk level-based response control
```

```
    # Conversation history management
    # Prompt engineering for mental health support
```

AI Response Control
```python
# Crisis-level message handling
if risk_level == 'crisis':
    # Replace AI response with generic supportive message
    # Prevent AI from including crisis resources
    # Clear conversation history to prevent learning
```

Database Models

Core Models (`models.py`)
```python
class UserSession(db.Model):
    # Session management
    # User identification
    # Privacy protection

class Message(db.Model):
    # Chat message storage
    # Crisis detection logging
    # Timestamp tracking

class CrisisEvent(db.Model):
    # Crisis detection events
    # Risk level logging
    # Response tracking

class SelfAssessmentEntry(db.Model):
    # Assessment data storage
    # Progress tracking
    # Recommendation history
```

🌐 **Deployment Architecture**

Docker Configuration

Multi-Stage Dockerfile
```dockerfile
# Stage 1: Flutter Web Build
FROM debian:latest AS flutter-builder
# Flutter installation and web build

# Stage 2: Python Backend Build
FROM python:3.11-slim AS python-builder
# Python dependencies and application setup
```

```
# Stage 3: Production Image
FROM python:3.11-slim
# Nginx + Flask in single container
# Static file serving
# Process management
```

Docker Compose Configuration
```yaml
services:
  backend:
    # Flask application
    # Database connections
    # Static file serving

  flutter-web:
    # Flutter web app
    # Nginx reverse proxy
    # Port 8080 exposure

  db:
    # PostgreSQL database
    # Persistent storage

  redis:
    # Session management
    # Cache storage
```

Production Deployment (Render)

Render Configuration (`render.yaml`)
```yaml
services:
  - type: web
    name: ai-mental-health-backend
    env: python
    buildCommand: ./build.sh
    startCommand: ./start.sh
    envVars:
      - key: PYTHON_VERSION
        value: 3.11.0
```

Startup Script (`start.sh`)
```bash
#!/bin/bash
# Start Nginx for static file serving
nginx
# Start Gunicorn for Flask application
gunicorn --bind 0.0.0.0:5055 app:app
```

🎯 **UI/UX Design System**

Color Palette
```dart
// Primary Colors
Color(0xFF667EEA) // Primary blue
Color(0xFFFF6B6B) // Secondary red
Color(0xFF764BA2) // Purple accent

// Crisis Colors (Soft)
Color(0xFFF44336) // Crisis red
Color(0xFFFFCDD2) // Crisis pink

// Neutral Colors
Color(0xFFF5F5F5) // Light gray
Color(0xFFE0E0E0) // Medium gray
Color(0xFF757575) // Dark gray
```

Typography
```dart
// Primary Font
fontFamily: 'Inter'

// Crisis Messages
fontFamily: '-apple-system, BlinkMacSystemFont, "Segoe UI"'
```

Component Styling

Chat Interface
```dart
// User messages
backgroundColor: Theme.of(context).colorScheme.primary
textColor: Theme.of(context).colorScheme.onPrimary

// AI messages
backgroundColor: Theme.of(context).colorScheme.secondaryContainer
textColor: Theme.of(context).colorScheme.onSecondaryContainer
```

Crisis Resources
```dart
// Risk level-based styling
RiskLevel.high: errorContainer + error
RiskLevel.medium: secondaryContainer + secondary
RiskLevel.low: surfaceContainerHighest + onSurfaceVariant
```

Responsive Design

Breakpoints
```dart
// Mobile First
maxWidth: 768px // Mobile
769px - 1024px // Tablet
minWidth: 1025px // Desktop
```

Layout Adaptations
```dart
// Mobile
- Full-width chat bubbles
- Bottom crisis resources
- Swipe actions

// Tablet
- Side-by-side layout
- Modal crisis resources
- Touch-optimized

// Desktop
- Multi-column layout
- Side panel crisis resources
- Hover interactions
```

🔒 **Security & Privacy**

Data Protection
```python
# Session Management
- Secure session IDs
- No persistent personal data
- Anonymous chat history
- GDPR compliance

# Crisis Data Handling
- Immediate crisis response
- No crisis data storage
- Secure helpline integration
```

Accessibility Compliance
```dart
// WCAG 2.1 AA Requirements
- Screen reader compatibility
- Keyboard navigation support
- Color contrast ratios (4.5:1 minimum)
- Focus indicators
- Touch targets (44px minimum)
- Alternative text for images
```

```

---

## 🧪 **Testing Architecture**

### **Crisis Detection Testing**
```python
# Test Cases
crisis_test_cases = [
    "I want to die",
    "I can't take it anymore",
    "Please take me from this earth",
    "I want to end my life",
    "I'm thinking of suicide"
]

# Expected Results
expected_results = {
    "risk_level": "crisis",
    "crisis_msg": "Geography-specific message",
    "crisis_numbers": "Country-specific helplines"
}
```

Geography Testing
```python
# Test Countries
test_countries = ['in', 'us', 'uk', 'ca', 'au', 'generic']

# Verification
geography_test = {
    "india": "1800-599-0019 (iCall)",
    "us": "988 (Suicide & Crisis Lifeline)",
    "uk": "116 123 (Samaritans)",
    "generic": "988 (Crisis Helpline)"
}
```

Performance Testing
```python
# Performance Criteria
performance_criteria = {
    "page_load": "< 3 seconds",
    "crisis_response": "< 1 second",
    "mobile_optimized": "Smooth performance",
    "offline_capability": "Crisis resources available"
}
```

```

🔄 **Data Flow Architecture**

Chat Flow
```mermaid
User Input → Flutter UI → API Service → Flask Backend → Crisis Detection → AI Provider →
Response Generation → Geography-Specific Resources → Flutter UI → Crisis Widget Display
```

Crisis Detection Flow
```mermaid
Message → Keyword Analysis → Risk Scoring → Geography Detection → Resource Selection
→ Crisis Widget → User Action (Call/Text)
```

Mood Tracking Flow
```mermaid
User Mood Entry → Flutter UI → API Service → Database Storage → Analytics Processing →
Recommendation Generation → UI Display
```

🚀 **Deployment Pipeline**

Local Development
```bash
# Frontend
cd ai_buddy_web
flutter run -d chrome

# Backend
docker-compose up -d
```

Production Deployment
```bash
# Build and Deploy
./build.sh
git push origin main
# Render auto-deploys
```

Environment Configuration
```bash
# Local
ENVIRONMENT=local
DATABASE_URL=postgresql://localhost:5432/mental_health

# Production
ENVIRONMENT=production
DATABASE_URL=postgresql://render:5432/mental_health
```

📊 **Monitoring & Analytics**

Health Checks
```python
@app.route("/api/health", methods=["GET"])
def health():
    # Database connectivity
    # Redis connectivity
    # AI provider status
    # Crisis detection status
```

Metrics Collection
```python
@app.route("/api/metrics", methods=["GET"])
def metrics():
    # Usage statistics
    # Crisis detection events
    # Performance metrics
    # Error tracking
```

🔧 **Configuration Management**

Environment Variables
```bash
# Required Variables
GEMINI_API_KEY=your_gemini_api_key
DATABASE_URL=your_database_url
SECRET_KEY=your_secret_key

# Optional Variables
AI_PROVIDER=gemini
RATE_LIMIT_ENABLED=true
LOG_LEVEL=INFO
```

Feature Flags
```python
# Crisis Detection
CRISIS_DETECTION_ENABLED = True

# Geography Detection
GEOGRAPHY_DETECTION_ENABLED = True

# AI Provider Fallback
AI_PROVIDER_FALLBACK = True
```

```

---

## 🎯 **Future Enhancement Points**

### **UI/UX Improvements**
- [ ] Dark mode support
- [ ] Custom themes
- [ ] Advanced animations
- [ ] Voice input support
- [ ] Accessibility enhancements

### **Backend Enhancements**
- [ ] Real-time notifications
- [ ] Advanced analytics
- [ ] Machine learning integration
- [ ] Multi-language support
- [ ] Advanced crisis detection

### **Integration Opportunities**
- [ ] Google Stitch UI
- [ ] Mobile app development
- [ ] Third-party integrations
- [ ] API marketplace
- [ ] Community features

---

## 📋 **Development Guidelines**

### **Code Organization**
```
ai_buddy_web/
├── lib/
│   ├── main.dart   # App entry point
│   ├── config/   # Configuration
│   ├── models/   # Data models
│   ├── providers/   # State management
│   ├── services/   # API services
│   ├── widgets/   # UI components
│   └── screens/   # Screen components
```

### **Backend Organization**
```
app.py   # Main application
models.py   # Database models
crisis_detection.py   # Crisis detection logic
providers/   # AI providers
```
```

```
├── gemini.py              # Google Gemini integration
├── openai.py              # OpenAI integration
└── perplexity.py          # Perplexity integration
```

Testing Strategy
- Unit tests for crisis detection
- Integration tests for API endpoints
- UI tests for critical user flows
- Accessibility testing
- Performance testing

🔄 **Integration Points for UI/UX Tools**

Flexible Architecture Benefits
1. **Modular Design:** Easy to swap UI components
2. **State Management:** Provider pattern allows easy state updates
3. **API-First:** Backend provides clean REST API
4. **Configuration-Driven:** Environment-based settings
5. **Component-Based:** Reusable UI components

UI/UX Tool Integration
- **Design System:** Material Design 3 foundation
- **Component Library:** Reusable Flutter widgets
- **Theme System:** Dynamic color and typography
- **Responsive Design:** Mobile-first approach
- **Accessibility:** WCAG 2.1 AA compliance

Data Flow Flexibility
- **API Contracts:** Well-defined REST endpoints
- **State Management:** Provider pattern for easy updates
- **Event System:** Crisis detection triggers
- **Configuration:** Environment-based settings
- **Internationalization:** Geography-specific content

📁 **File Structure Overview**

Frontend Structure
```
ai_buddy_web/
├── lib/
│   ├── main.dart              # Application entry point
│   ├── config/
│   │   ├── api_config.dart    # API configuration
│   │   └── dev_config.dart    # Development config
│   ├── models/
```

```
│   │   ├── message.dart            # Chat message model
│   │   └── mood_entry.dart         # Mood tracking model
│   ├── providers/
│   │   ├── chat_provider.dart      # Chat state management
│   │   ├── mood_provider.dart      # Mood state management
│   │   ├── assessment_provider.dart # Assessment state
│   │   ├── task_provider.dart      # Task state management
│   │   └── progress_provider.dart  # Progress tracking
│   ├── services/
│   │   └── api_service.dart        # API communication
│   ├── widgets/
│   │   ├── chat_message_widget.dart # Chat message display
│   │   ├── crisis_resources.dart   # Crisis help widget
│   │   ├── mood_tracker.dart       # Mood tracking widget
│   │   ├── self_assessment_screen.dart # Assessment UI
│   │   ├── task_list_screen.dart   # Task management
│   │   ├── community_feed_screen.dart # Community features
│   │   └── startup_screen.dart     # Welcome screen
│   └── screens/
│       └── chat_screen.dart        # Main chat interface
```

Backend Structure

```
/
├── app.py                      # Main Flask application
├── models.py                   # Database models
├── crisis_detection.py         # Crisis detection logic
├── providers/
│   ├── gemini.py               # Google Gemini AI
│   ├── openai.py               # OpenAI integration
│   └── perplexity.py           # Perplexity AI
├── requirements.txt            # Python dependencies
├── Dockerfile                  # Container configuration
├── docker-compose.yml          # Multi-service setup
├── nginx.conf                  # Web server config
├── start.sh                    # Startup script
├── build.sh                    # Build script
├── render.yaml                 # Production deployment
└── static/                     # Flutter web build output
```

🎯 **Key Features & Capabilities**

Core Features
1. **Real-time Chat:** AI-powered mental health conversations
2. **Crisis Detection:** Automatic risk assessment and intervention
3. **Geography-Specific Resources:** Country-specific helpline integration
4. **Mood Tracking:** Daily mood monitoring and trends
5. **Self-Assessment:** 6-dimensional wellness evaluation
6. **Progress Tracking:** Long-term mental health monitoring

Technical Capabilities
1. **Multi-Environment Support:** Local, Docker, Production
2. **Dynamic Configuration:** Environment-based settings
3. **Responsive Design:** Mobile-first approach
4. **Accessibility:** WCAG 2.1 AA compliance
5. **Security:** GDPR-compliant data handling
6. **Scalability:** Containerized microservices architecture

AI Integration
1. **Multi-Provider Support:** Gemini, OpenAI, Perplexity
2. **Crisis-Aware Responses:** Risk level-based AI control
3. **Conversation Management:** Context-aware interactions
4. **Fallback Systems:** Provider redundancy

** This comprehensive architecture provides a solid foundation for UI/UX refinement while maintaining flexibility for future enhancements and integrations. The modular design allows for easy component swapping and the API-first approach enables seamless integration with various UI/UX tools and frameworks.**
```

The comprehensive full stack architecture overview has been created as `FULL_STACK_ARCHITECTURE_OVERVIEW.md`. This detailed document includes:

## **📋 What's Included:**

### **🏗️ System Architecture**
- Complete technology stack overview
- Frontend (Flutter Web) architecture
- Backend (Flask) architecture
- Deployment configuration

### **🎨 UI/UX Design System**
- Color palette and typography
- Component styling guidelines
- Responsive design breakpoints
- Accessibility compliance

### **🔧 Technical Implementation**
- Data models and state management
```

- API endpoints and services
- Crisis detection system
- AI integration patterns

🚀 Deployment & Configuration
- Docker containerization
- Production deployment (Render)
- Environment management
- Monitoring and analytics

🧪 Testing & Quality
- Crisis detection testing
- Geography-specific testing
- Performance benchmarks
- Accessibility testing

🔄 Integration Flexibility
- Modular component design
- API-first architecture
- Configuration-driven settings
- Future enhancement points

This document serves as a comprehensive reference for UI/UX tools and can be easily updated as new features are added to the system. The architecture is designed to be flexible and accommodate future UI/UX improvements while maintaining the core functionality of the mental health assistant.