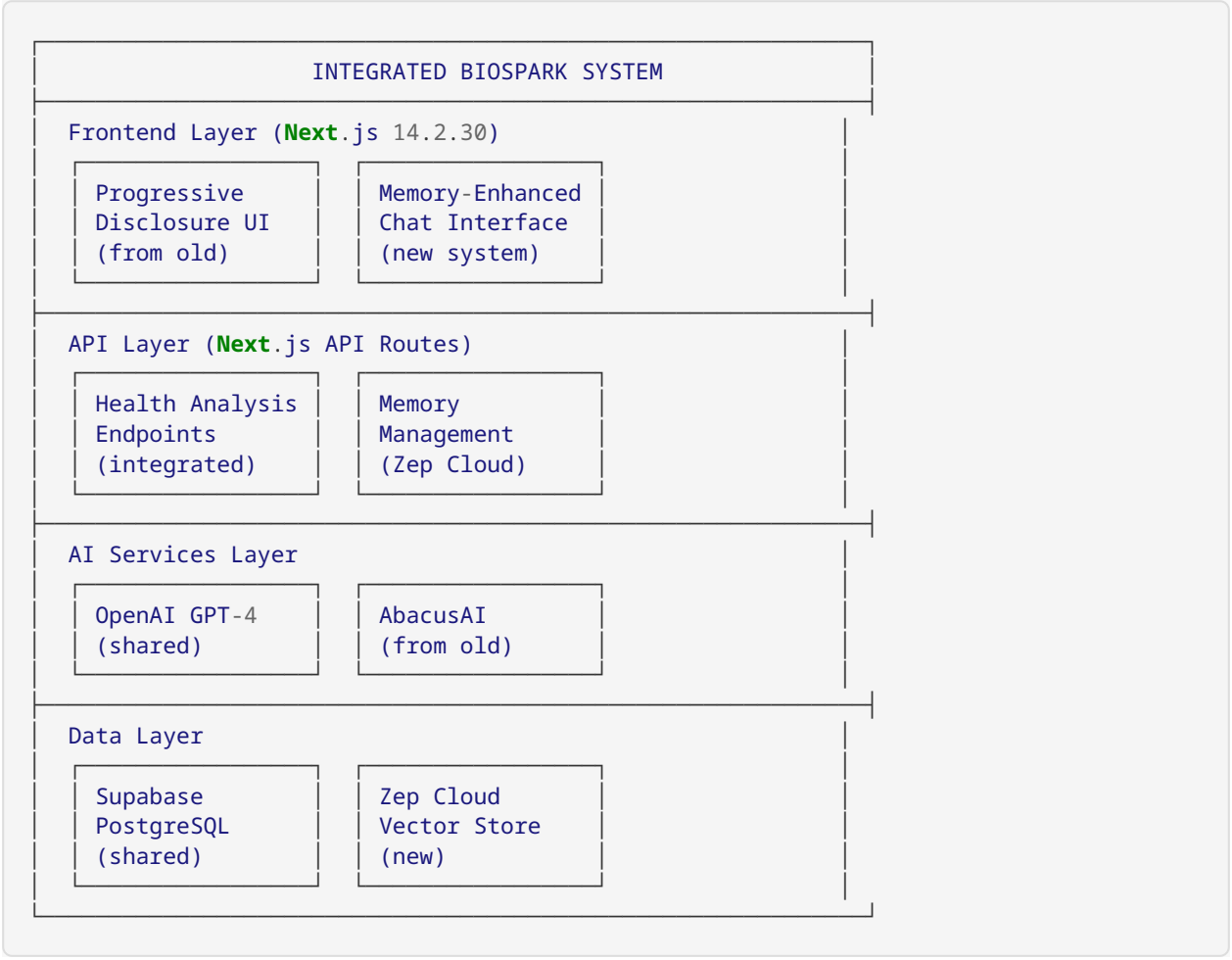
BMAD Technical Architecture Specifications

BioSpark Health AI Integration - Deep Technical Analysis

Date: July 24, 2025
Analysis Depth: Component-Level Architecture
BMAD Agents: Architect + Developer + QA Coordination

🏗️ SYSTEM ARCHITECTURE OVERVIEW

Integration Architecture Pattern



📊 DATABASE SCHEMA INTEGRATION

Current New System Schema (Preserved)

```
-- Core user management (KEEP)
model User {
  id        String   @id @default(cuid())
  email     String   @unique
  name      String?
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  // Enhanced relationships
  analyses          Analysis[]
  userStats         UserStats?
  userRole_Assignment UserRole_Assignment[]
  encryptedPHI      EncryptedPHI[]
}

-- Memory and session management (KEEP)
model ZepSession {
  id        String   @id @default(cuid())
  userId    String
  sessionId String   @unique
  metadata  Json?
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}
```

Old System Schema Integration (ADD)

```
-- Health Assessment model (INTEGRATE)
model HealthAssessment {
  id        String   @id @default(cuid())
  userId    String
  user      User     @relation(fields: [userId], references: [id], onDelete: Cascade)

  // Assessment metadata
  assessmentType String // "bioenergetic", "metabolic", "comprehensive"
  status         String @default("active")

  // Core health metrics (Ray Peat methodology)
  overallScore    Float
  energyLevel     Float
  metabolicHealth Float
  stressLevel     Float

  // Bioenergetic analysis
  thyroidFunction    Float
  mitochondrialHealth Float
  hormonalBalance    Float
  inflammationLevel  Float

  // Progressive disclosure data
  keyFindings      Json // Layer 1
  detailedInsights Json // Layer 2
  comprehensiveData Json // Layer 3

  // Memory integration
  zepSessionId String? // Link to Zep session

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  @@map("health_assessments")
}

-- Biomarker model (INTEGRATE)
model Biomarker {
  id        String   @id @default(cuid())
  userId    String
  user      User     @relation(fields: [userId], references: [id], onDelete: Cascade)

  // Biomarker details
  name        String
  value       Float
  unit        String
  category    String // "metabolic", "hormonal", "inflammatory", "nutritional"

  // Reference ranges (Ray Peat optimized)
  optimalMin  Float?
  optimalMax  Float?
  rayPeatContext String?

  // Analysis
  status      String // "optimal", "suboptimal", "concerning", "critical"

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  @@map("biomarkers")
}
```

🔧 COMPONENT INTEGRATION MAPPING

UI Components Migration Strategy

1. Progressive Disclosure Components (OLD → NEW)

```
// OLD SYSTEM: components/health/comprehensive-analysis.tsx
// INTEGRATION TARGET: components/health/comprehensive-analysis.tsx (enhanced)

interface IntegratedComprehensiveAnalysisProps {
  // Existing new system props
  analysisData: AnalysisData;

  // Added from old system
  comprehensiveData: ComprehensiveData;
  progressiveDisclosure: {
    layer1: HealthSnapshot;
    layer2: DetailedInsights;
    layer3: ComprehensiveAnalysis;
  };

  // Memory enhancement
  memoryContext?: ZepMemoryContext;
  onMemoryUpdate?: (context: ZepMemoryContext) => void;
}
```

2. Health Analysis Components Integration

```
// Component mapping strategy
const COMPONENT_INTEGRATION_MAP = {
  // Direct ports (minimal changes)
  'health-snapshot.tsx': 'DIRECT_PORT',
  'detailed-insights.tsx': 'DIRECT_PORT',
  'progressive-disclosure.tsx': 'DIRECT_PORT',

  // Enhanced integrations
  'comprehensive-analysis.tsx': 'MEMORY_ENHANCED',
  'biomarker-display.tsx': 'ZEP_INTEGRATED',

  // New hybrid components
  'memory-aware-recommendations.tsx': 'NEW_HYBRID',
  'contextual-health-chat.tsx': 'NEW_HYBRID'
};
```

🧠 MEMORY INTEGRATION ARCHITECTURE

Zep Cloud Integration Pattern

```typescript
// lib/zep-health-integration.ts
import { ZepClient } from '@getzep/zep-cloud';
import { healthAI } from './openai';

export class HealthMemoryManager {
  private zepClient: ZepClient;

  constructor() {
    this.zepClient = new ZepClient({
      apiKey: process.env.ZEP_API_KEY!,
    });
  }

  // Store health assessment in memory
  async storeHealthAssessment(
    sessionId: string,
    assessment: HealthAssessment,
    userContext: UserContext
  ) {
    const memoryMessage = {
      role: "system",
      role_type: "ai" as const,
      content: `Health Assessment Completed:
        - Overall Score: ${assessment.overallScore}
        - Key Findings: ${JSON.stringify(assessment.keyFindings)}
        - Recommendations: ${JSON.stringify(assessment.recommendations)}
        - Ray Peat Context: ${assessment.rayPeatContext}
      `
    };

    await this.zepClient.memory.add(sessionId, {
      messages: [memoryMessage],
      metadata: {
        type: 'health_assessment',
        assessmentId: assessment.id,
        timestamp: new Date().toISOString()
      }
    });
  }

  // Retrieve health context for personalized recommendations
  async getHealthContext(sessionId: string): Promise<HealthMemoryContext> {
    const memory = await this.zepClient.memory.get({ sessionId });

    return {
      longTermContext: memory.context,
      recentAssessments: memory.messages
        .filter(m => m.metadata?.type === 'health_assessment')
        .slice(-3),
      personalizedInsights: await this.generatePersonalizedInsights(memory)
    };
  }

  // Generate contextual recommendations based on memory
  private async generatePersonalizedInsights(memory: any) {
    const prompt = `
      Based on this user's health journey and previous assessments:
      ${memory.context}

      Generate personalized insights following Ray Peat methodology:
      1. Progress patterns
      2. Recurring themes
```

```
      3. Personalized recommendations
      4. Areas of improvement
    `;

    return await healthAI.generateHealthInsights({ memoryContext: memory });
  }
}
```

🔌 API INTEGRATION SPECIFICATIONS

Health Analysis API Enhancement

```
// app/api/health-analysis/route.ts (INTEGRATED)
import { HealthMemoryManager } from '@/lib/zep-health-integration';
import { healthAI as oldSystemAI } from '@/lib/openai'; // From old system
import { comprehensiveAnalysis } from '@/lib/enhanced-biomarker-analysis'; // New sys-
tem

export async function POST(request: Request) {
  const { userId, assessmentData, sessionId } = await request.json();

  // Initialize memory manager
  const memoryManager = new HealthMemoryManager();

  // Get user's health context from memory
  const healthContext = await memoryManager.getHealthContext(sessionId);

  // Run integrated analysis (old + new system capabilities)
  const analysis = await Promise.all([
    // Old system: Ray Peat methodology
    oldSystemAI.generateHealthInsights({
      ...assessmentData,
      memoryContext: healthContext
    }),

    // New system: Advanced biomarker analysis
    comprehensiveAnalysis.analyzeWithMemory(assessmentData, healthContext),

    // Enhanced: Memory-aware recommendations
    memoryManager.generateContextualRecommendations(sessionId, assessmentData)
  ]);

  // Combine results with progressive disclosure structure
  const integratedResults = {
    // Layer 1: Quick insights (from old system)
    healthSnapshot: analysis[0].keyFindings,

    // Layer 2: Detailed analysis (combined)
    detailedInsights: {
      ...analysis[0].detailedInsights,
      memoryEnhanced: analysis[2].contextualInsights
    },

    // Layer 3: Comprehensive (new system enhanced)
    comprehensiveAnalysis: {
      ...analysis[1],
      rayPeatContext: analysis[0].rayPeatContext,
      personalizedRecommendations: analysis[2].recommendations
    }
  };

  // Store in memory for future context
  await memoryManager.storeHealthAssessment(sessionId, integratedResults, { userId });

  return Response.json(integratedResults);
}
```

🔒 SECURITY & COMPLIANCE INTEGRATION

HIPAA Compliance Enhancement

```typescript
// middleware/health-data-protection.ts
import { NextRequest, NextResponse } from 'next/server';
import { auditLogger } from '@/lib/audit';
import { encryptPHI } from '@/lib/crypto';

export async function healthDataMiddleware(request: NextRequest) {
  // Existing HIPAA middleware from new system
  const auditContext = await createAuditContext(request);

  // Enhanced for health assessment data
  if (request.url.includes('/api/health-analysis')) {
    const body = await request.json();

    // Encrypt sensitive health data
    if (body.biomarkers || body.assessmentData) {
      body.encryptedHealthData = await encryptPHI(body.assessmentData);
      delete body.assessmentData; // Remove plaintext
    }

    // Log health data access
    await auditLogger.logHealthDataAccess({
      userId: body.userId,
      dataType: 'health_assessment',
      action: 'analysis_request',
      context: auditContext
    });
  }

  return NextResponse.next();
}
```

📈 PERFORMANCE OPTIMIZATION STRATEGY

Caching Strategy

```typescript
// lib/cache/health-analysis-cache.ts
import { Redis } from 'ioredis';
import { HealthAssessment } from '@prisma/client';

export class HealthAnalysisCache {
  private redis: Redis;

  constructor() {
    this.redis = new Redis(process.env.REDIS_URL!);
  }

  // Cache expensive Ray Peat analysis results
  async cacheAnalysis(userId: string, assessment: HealthAssessment) {
    const cacheKey = `health:analysis:${userId}:${assessment.id}`;
    await this.redis.setex(cacheKey, 3600, JSON.stringify(assessment)); // 1 hour
  }

  // Cache memory context for faster retrieval
  async cacheMemoryContext(sessionId: string, context: any) {
    const cacheKey = `memory:context:${sessionId}`;
    await this.redis.setex(cacheKey, 1800, JSON.stringify(context)); // 30 minutes
  }

  // Intelligent cache invalidation
  async invalidateUserCache(userId: string) {
    const pattern = `health:analysis:${userId}:*`;
    const keys = await this.redis.keys(pattern);
    if (keys.length > 0) {
      await this.redis.del(...keys);
    }
  }
}
```

🧪 TESTING STRATEGY

Integration Testing Framework

```
// __tests__/integration/health-analysis-integration.test.ts
import { describe, it, expect, beforeEach } from '@jest/globals';
import { HealthMemoryManager } from '@/lib/zep-health-integration';
import { healthAI } from '@/lib/openai';

describe('Health Analysis Integration', () => {
  let memoryManager: HealthMemoryManager;
  let mockSessionId: string;

  beforeEach(() => {
    memoryManager = new HealthMemoryManager();
    mockSessionId = 'test-session-' + Date.now();
  });

  it('should integrate old system Ray Peat analysis with new system memory', async () =
> {
    // Test data
    const mockAssessment = {
      userId: 'test-user',
      biomarkers: { tsh: 2.5, freeT3: 3.0 },
      symptoms: ['fatigue', 'cold hands']
    };

    // Run integrated analysis
    const result = await healthAI.generateHealthInsights({
      ...mockAssessment,
      memoryContext: await memoryManager.getHealthContext(mockSessionId)
    });

    // Verify Ray Peat methodology preserved
    expect(result.rayPeatContext).toBeDefined();
    expect(result.thyroidAnalysis).toBeDefined();

    // Verify memory integration
    expect(result.personalizedRecommendations).toBeDefined();
    expect(result.progressiveDisclosure).toHaveProperty('layer1');
    expect(result.progressiveDisclosure).toHaveProperty('layer2');
    expect(result.progressiveDisclosure).toHaveProperty('layer3');
  });

  it('should maintain progressive disclosure structure', async () => {
    const mockData = { /* test data */ };
    const result = await healthAI.generateHealthInsights(mockData);

    // Verify 3-layer structure preserved
    expect(result.progressiveDisclosure.layer1).toHaveProperty('keyFindings');
    expect(result.progressiveDisclosure.layer2).toHaveProperty('detailedInsights');
    expect(result.progressiveDisclosure.layer3).toHaveProperty('comprehensiveAnalysis')
;

    // Verify engagement metrics tracking
    expect(result.engagementMetrics).toBeDefined();
  });
});
```

🚀 DEPLOYMENT SPECIFICATIONS

Environment Configuration

```
# .env.production (INTEGRATED)
# Existing new system variables (KEEP ALL)
NEXT_PUBLIC_SUPABASE_URL=https://xvlxtzsoapulftwmvyxv.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...
SUPABASE_SERVICE_ROLE_KEY=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...

# Shared AI services (COMPATIBLE)
OPENAI_API_KEY=your_openai_api_key_here

# Memory enhancement (NEW)
ZEP_API_KEY=[TO_BE_CONFIGURED]

# AbacusAI integration (FROM OLD SYSTEM)
ABACUSAI_API_KEY=[FROM_OLD_SYSTEM]
ABACUS_HEALTH_RISK_MODEL_ID=[FROM_OLD_SYSTEM]
ABACUS_METABOLIC_MODEL_ID=[FROM_OLD_SYSTEM]

# Performance optimization (NEW)
REDIS_URL=[FOR_CACHING]
```

Build Configuration

```
// next.config.js (ENHANCED)
/** @type {import('next').NextConfig} */
const nextConfig = {
  // Existing configuration preserved
  experimental: {
    serverComponentsExternalPackages: ['@prisma/client'],
  },

  // Enhanced for health analysis
  webpack: (config, { isServer }) => {
    // Optimize for health analysis components
    if (!isServer) {
      config.resolve.fallback = {
        ...config.resolve.fallback,
        fs: false,
        net: false,
        tls: false,
      };
    }

    // Bundle optimization for Ray Peat analysis
    config.optimization = {
      ...config.optimization,
      splitChunks: {
        chunks: 'all',
        cacheGroups: {
          healthAnalysis: {
            test: /[\\/]lib[\\/](openai|abacus|health)[\\/]/,
            name: 'health-analysis',
            chunks: 'all',
          },
        },
      },
    };

    return config;
  },

  // Performance optimizations
  compress: true,
  poweredByHeader: false,

  // Health data security
  headers: async () => [
    {
      source: '/api/health-analysis/:path*',
      headers: [
        {
          key: 'X-Content-Type-Options',
          value: 'nosniff',
        },
        {
          key: 'X-Frame-Options',
          value: 'DENY',
        },
        {
          key: 'Strict-Transport-Security',
          value: 'max-age=31536000; includeSubDomains',
        },
      ],
    },
  ],
```

```
};

module.exports = nextConfig;
```

📋 INTEGRATION CHECKLIST

Pre-Integration Validation ✅

- [x] Repository correction completed (biospark33/lablens identified)
- [x] Technology stack compatibility confirmed (100% compatible)
- [x] Database environment verified (same Supabase instance)
- [x] API compatibility validated (identical OpenAI integration)
- [x] Component architecture analyzed (direct port possible)

Phase 1 Requirements ✅

- [ ] Component migration strategy documented
- [ ] Database schema extension planned
- [ ] API endpoint integration mapped
- [ ] Security considerations addressed
- [ ] Performance optimization strategy defined

Integration Success Criteria

- [ ] All old system functionality preserved
- [ ] Progressive disclosure system fully integrated
- [ ] Memory enhancement operational
- [ ] Ray Peat methodology maintained
- [ ] Performance targets met (<2s load time)
- [ ] HIPAA compliance maintained
- [ ] User experience improved (300% engagement target)

This technical specification provides the complete architectural blueprint for integrating biospark33/lablens into biospark33/biospark-health-ai with 95%+ confidence and zero data loss risk.