Integration Points - Exact Code Locations

Overview

This document provides precise integration points and code locations for implementing the BioSpark Health AI system components.

Core Integration Points

1. Authentication Integration

Location: `/src/auth/`
Files:
- `auth.service.ts` - Core authentication logic
- `jwt.middleware.ts` - JWT token validation
- `auth.controller.ts` - Authentication endpoints

```
// /src/auth/auth.service.ts
export class AuthService {
  // Integration point for external auth providers
  async authenticateUser(credentials: LoginCredentials): Promise<AuthResult> {
    // INTEGRATION: Connect to external identity provider here
    const user = await this.validateCredentials(credentials);
    return this.generateTokens(user);
  }
}
```

2. DeepAgent Framework Integration

Location: `/src/agents/`
Files:
- `agent.orchestrator.ts` - Main orchestration logic
- `agent.factory.ts` - Agent creation and management
- `agent.communication.ts` - Inter-agent communication

```
// /src/agents/agent.orchestrator.ts
export class AgentOrchestrator {
  // INTEGRATION POINT: DeepAgent framework connection
  async initializeAgents(): Promise<void> {
    // Connect to DeepAgent runtime
    this.deepAgentRuntime = new DeepAgentRuntime({
      endpoint: process.env.DEEPAGENT_ENDPOINT,
      apiKey: process.env.DEEPAGENT_API_KEY
    });

    // Initialize agent types
    await this.createAgent('analyst', AnalystAgentConfig);
    await this.createAgent('architect', ArchitectAgentConfig);
    await this.createAgent('developer', DeveloperAgentConfig);
    await this.createAgent('orchestrator', OrchestratorAgentConfig);
  }
}
```

3. BMAD Core System Integration

Location: `/src/bmad/`
Files:
- `bmad.core.ts` - Core BMAD functionality
- `bmad.processor.ts` - Data processing engine
- `bmad.analytics.ts` - Analytics integration

```
// /src/bmad/bmad.core.ts
export class BMADCore {
  // INTEGRATION POINT: BMAD system connection
  async initializeBMAD(): Promise<void> {
    // Connect to BMAD processing engine
    this.bmadEngine = new BMADEngine({
      processingEndpoint: process.env.BMAD_PROCESSING_URL,
      analyticsEndpoint: process.env.BMAD_ANALYTICS_URL,
      credentials: {
        apiKey: process.env.BMAD_API_KEY,
        secretKey: process.env.BMAD_SECRET_KEY
      }
    });

    // Initialize processing pipelines
    await this.setupDataPipelines();
    await this.initializeAnalytics();
  }
}
```

4. Database Integration Points

Location: `/src/database/`
Files:
- `database.config.ts` - Database configuration
- `repositories/` - Data access layer
- `migrations/` - Database schema migrations

```
// /src/database/database.config.ts
export const databaseConfig = {
  // INTEGRATION POINT: Database connection configuration
  primary: {
    host: process.env.DB_HOST,
    port: parseInt(process.env.DB_PORT || '5432'),
    database: process.env.DB_NAME,
    username: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    ssl: process.env.NODE_ENV === 'production'
  },

  // Time-series database for health metrics
  timeseries: {
    host: process.env.TIMESCALE_HOST,
    port: parseInt(process.env.TIMESCALE_PORT || '5432'),
    database: process.env.TIMESCALE_DB,
    username: process.env.TIMESCALE_USER,
    password: process.env.TIMESCALE_PASSWORD
  }
};
```

5. API Gateway Integration

Location: `/src/gateway/`
Files:
- `gateway.config.ts` - Gateway configuration
- `middleware/` - Custom middleware
- `routes/` - Route definitions

```
// /src/gateway/gateway.config.ts
export class GatewayConfig {
  // INTEGRATION POINT: Kong Gateway configuration
  static getKongConfig() {
    return {
      admin_api_uri: process.env.KONG_ADMIN_URL,
      proxy_uri: process.env.KONG_PROXY_URL,
      plugins: [
        {
          name: 'rate-limiting',
          config: {
            minute: 1000,
            policy: 'local'
          }
        },
        {
          name: 'jwt',
          config: {
            secret_is_base64: false,
            key_claim_name: 'iss'
          }
        }
      ]
    };
  }
}
```

Frontend Integration Points

1. React Components Integration

Location: `/frontend/src/components/`
Files:
- `Dashboard/` - Main dashboard components
- `Agents/` - Agent interaction components
- `Analytics/` - Data visualization components

```
// /frontend/src/components/Dashboard/HealthDashboard.tsx
export const HealthDashboard: React.FC = () => {
  // INTEGRATION POINT: Real-time data connection
  const { data, loading, error } = useHealthData({
    endpoint: '/api/v1/health/dashboard',
    realtime: true,
    refreshInterval: 5000
  });

  // INTEGRATION POINT: Agent communication
  const { sendMessage, messages } = useAgentCommunication({
    agentType: 'analyst',
    channel: 'health-analysis'
  });

  return (
    <div className="health-dashboard">
      {/* Dashboard content */}
    </div>
  );
};
```

2. State Management Integration

Location: `/frontend/src/store/`
Files:
- `auth.store.ts` - Authentication state
- `agents.store.ts` - Agent state management
- `health.store.ts` - Health data state

```
// /frontend/src/store/auth.store.ts
export const useAuthStore = create<AuthState>((set, get) => ({
  // INTEGRATION POINT: Authentication state management
  user: null,
  token: null,
  isAuthenticated: false,

  login: async (credentials: LoginCredentials) => {
    // Connect to authentication API
    const response = await fetch('/api/v1/auth/login', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(credentials)
    });

    const { token, user } = await response.json();
    set({ token, user, isAuthenticated: true });
  }
}));
```

External System Integration Points

1. EHR System Integration

Location: `/src/integrations/ehr/`
Files:
- `fhir.client.ts` - FHIR client implementation

- `ehr.mapper.ts` - Data mapping utilities
- `ehr.sync.ts` - Synchronization logic

```typescript
// /src/integrations/ehr/fhir.client.ts
export class FHIRClient {
  // INTEGRATION POINT: EHR system connection
  constructor(config: FHIRConfig) {
    this.baseUrl = config.baseUrl;
    this.credentials = config.credentials;
    this.client = new FHIRHttpClient({
      baseUrl: this.baseUrl,
      auth: {
        type: 'oauth2',
        clientId: config.clientId,
        clientSecret: config.clientSecret
      }
    });
  }

  async syncPatientData(patientId: string): Promise<PatientData> {
    // Fetch patient data from EHR system
    const patient = await this.client.read('Patient', patientId);
    const observations = await this.client.search('Observation', {
      patient: patientId,
      _sort: '-date'
    });

    return this.mapToInternalFormat(patient, observations);
  }
}
```

2. Medical Device Integration

Location: `/src/integrations/devices/`
Files:
- `device.manager.ts` - Device management
- `iot.client.ts` - IoT device communication
- `data.processor.ts` - Device data processing

```
// /src/integrations/devices/iot.client.ts
export class IoTClient {
  // INTEGRATION POINT: Medical device data ingestion
  async connectToDevices(): Promise<void> {
    // Connect to IoT platform
    this.mqttClient = mqtt.connect(process.env.IOT_BROKER_URL, {
      username: process.env.IOT_USERNAME,
      password: process.env.IOT_PASSWORD
    });

    // Subscribe to device data topics
    this.mqttClient.subscribe('devices/+/vitals');
    this.mqttClient.subscribe('devices/+/alerts');

    this.mqttClient.on('message', this.handleDeviceMessage.bind(this));
  }

  private async handleDeviceMessage(topic: string, message: Buffer): Promise<void> {
    const deviceData = JSON.parse(message.toString());
    await this.processDeviceData(topic, deviceData);
  }
}
```

Configuration Integration Points

1. Environment Configuration

Location: `/config/`

Files:
- `development.env` - Development environment
- `staging.env` - Staging environment
- `production.env` - Production environment

```
# /config/production.env
# INTEGRATION POINT: Production configuration

# Database Configuration
DB_HOST=biospark-postgres.internal
DB_PORT=5432
DB_NAME=biospark_prod
DB_USER=biospark_user
DB_PASSWORD=${VAULT_DB_PASSWORD}

# DeepAgent Configuration
DEEPAGENT_ENDPOINT=https://deepagent.biospark.ai
DEEPAGENT_API_KEY=${VAULT_DEEPAGENT_KEY}

# BMAD Configuration
BMAD_PROCESSING_URL=https://bmad-processor.biospark.ai
BMAD_ANALYTICS_URL=https://bmad-analytics.biospark.ai
BMAD_API_KEY=${VAULT_BMAD_KEY}

# External Integrations
EHR_FHIR_ENDPOINT=https://ehr.hospital.com/fhir
EHR_CLIENT_ID=${VAULT_EHR_CLIENT_ID}
EHR_CLIENT_SECRET=${VAULT_EHR_CLIENT_SECRET}
```

2. Kubernetes Configuration

Location: `/k8s/`

Files:
- `deployments/` - Application deployments
- `services/` - Service definitions
- `configmaps/` - Configuration maps

```yaml
# /k8s/deployments/api-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: biospark-api
spec:
  replicas: 3
  selector:
    matchLabels:
      app: biospark-api
  template:
    metadata:
      labels:
        app: biospark-api
    spec:
      containers:
      - name: api
        image: biospark/api:latest
        ports:
        - containerPort: 3000
        env:
        # INTEGRATION POINT: Environment variables from ConfigMap and Secrets
        - name: DB_HOST
          valueFrom:
            configMapKeyRef:
              name: biospark-config
              key: db-host
        - name: DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: biospark-secrets
              key: db-password
```

Testing Integration Points

1. Unit Test Integration

Location: `/tests/unit/`

Files:
- `auth.test.ts` - Authentication tests
- `agents.test.ts` - Agent functionality tests
- `bmad.test.ts` - BMAD core tests

```typescript
// /tests/unit/auth.test.ts
describe('AuthService', () => {
  // INTEGRATION POINT: Mock external dependencies
  beforeEach(() => {
    jest.clearAllMocks();
    mockDeepAgentRuntime.mockClear();
    mockBMADEngine.mockClear();
  });

  it('should authenticate user successfully', async () => {
    const authService = new AuthService();
    const result = await authService.authenticateUser({
      email: 'test@example.com',
      password: 'password123'
    });

    expect(result.token).toBeDefined();
    expect(result.user).toBeDefined();
  });
});
```

2. Integration Test Points

Location: `/tests/integration/`

Files:

- `api.integration.test.ts` - API integration tests
- `database.integration.test.ts` - Database integration tests
- `agents.integration.test.ts` - Agent integration tests

```typescript
// /tests/integration/api.integration.test.ts
describe('API Integration Tests', () => {
  // INTEGRATION POINT: Test against real services
  beforeAll(async () => {
    await setupTestDatabase();
    await startTestServices();
  });

  it('should handle complete patient workflow', async () => {
    // Test complete integration flow
    const patient = await createTestPatient();
    const analysis = await requestHealthAnalysis(patient.id);
    const results = await getAnalysisResults(analysis.id);

    expect(results.status).toBe('completed');
    expect(results.insights).toBeDefined();
  });
});
```

This integration points document provides exact code locations and implementation details for seamless system integration.