PsychSync - System Architecture & Technical Documentation

Table of Contents

Project Overview

PsychSync is a comprehensive team psychology and optimization platform that helps organizations build high-performing teams through personality assessments, behavioral analysis, and AI-driven recommendations.

Core Features

- **Team Management**: Create, manage, and organize teams
- **Assessment Center**: Multiple personality frameworks (MBTI, Big Five, DISC, etc.)
- **Team Optimizer**: AI-powered team composition recommendations
- **Analytics Dashboard**: Performance metrics and behavioral insights
- **Predictive Analytics**: Team velocity and compatibility predictions

Business Value

- Reduce team formation time by 60%
- Improve team productivity by 35%
- Decrease team conflict incidents by 45%
- Enhance employee satisfaction and retention

System Architecture

High-Level Architecture

```
┌──────────────────────────────┐    ┌──────────────────────────────┐
┌──────────────────────────────┐
│  Web Client   │  │ Mobile App   │  │ Admin Panel  │
│  (React SPA)  │  │  (React      │  │  (React)     │
│               │  │  Native)     │  │              │
└───────────────┬──────────────┘    └───────────────┬──────────────┘
└───────────────┬──────────────┘
       │               │               │
       └───────────────────────────────┼───────────────────────────────┘
                       │
               ┌───────────────────────▼───────────────────────┐
               │    Load Balancer      │
               │    (NGINX/AWS ALB)     │
               └───────────────────────┬───────────────────────┘
                       │
               ┌───────────────────────▼───────────────────────┐
               │    API Gateway        │
               │    (Kong/AWS API GW)   │
               └───────────────────────┬───────────────────────┘
                       │
       ┌───────────────────────────────┼───────────────────────────────┐
       │               │               │
┌───────────────▼──────────────┐    ┌───────────────▼──────────────┐
┌───────────────▼──────────────┐
│ Auth Service  │  │ Core API     │  │ AI/ML Service │
│ (Node.js)     │  │ (Python/Fast │  │ (Python/      │
│               │  │  API)        │  │  TensorFlow)  │
└───────────────┬──────────────┘    └───────────────┬──────────────┘
└───────────────┬──────────────┘
       │               │               │
       │         ┌─────────────▼────────────┐         │
       │         │ Message      │          │
       │         │ Queue        │          │
       │         │ (Redis/      │          │
       │         │  RabbitMQ)   │          │
       │         └──────────────────────────┘         │
       │                       │
       └───────────────────────┬───────────────────────────────────────┘
                   │
            ┌──────────────────▼─────────────────┐
            │  Database       │
            │  Cluster        │
            │                 │
            │  ┌────────────────────────┐  │
            │ │PostgreSQL  │ │
            │ │(Primary)   │ │
            │  └─────────────────┘  │
            │  ┌─────────────────┐  │
            │ │PostgreSQL  │ │
            │ │(Read       │ │
            │ │ Replica)   │ │
            │  └─────────────────┘  │
            │  ┌─────────────────┐  │
            │ │Redis       │ │
            │ │(Cache)     │ │
            │  └─────────────────┘  │
            │  ┌─────────────────┐  │
            │ │Elasticsearch│ │
            │ │(Search)    │ │
            │  └─────────────────┘  │
             └─────────────────────────┘
```

Microservices Architecture

Core Services

1. **Authentication Service**
   - JWT token management
   - OAuth integration
   - User session management
   - Role-based access control
2. **User Management Service**
   - User profiles and preferences
   - Organization management
   - Team membership tracking
3. **Assessment Service**
   - Assessment framework management
   - Question bank administration
   - Response collection and validation
   - Scoring algorithms
4. **Team Management Service**
   - Team CRUD operations
   - Member assignment and roles
   - Team hierarchy management
5. **Analytics Service**

- Data aggregation and processing
- Report generation
- Metrics calculation
- Trend analysis

6. **AI/ML Service**
   - Personality analysis algorithms
   - Team optimization recommendations
   - Predictive modeling
   - Behavioral pattern recognition
7. **Notification Service**
   - Email notifications
   - Push notifications
   - In-app notifications
   - Communication templates

Technology Stack

Frontend

```yaml
Framework: React 18+
Language: TypeScript/JavaScript ES6+
State Management: Context API + useReducer
Routing: React Router v6
Styling: Tailwind CSS
Charts: Recharts/Chart.js
Icons: Heroicons/Lucide React
Build Tool: Vite/Create React App
Testing: Jest + React Testing Library
```

Backend

```yaml
API Framework: FastAPI (Python) / Express.js (Node.js)
Authentication: JWT + OAuth 2.0
Database ORM: SQLAlchemy (Python) / Prisma (Node.js)
Validation: Pydantic (Python) / Joi (Node.js)
Documentation: OpenAPI/Swagger
Testing: pytest (Python) / Jest (Node.js)
```

Database

```yaml
Primary Database: PostgreSQL 14+
Cache: Redis 6+
Search Engine: Elasticsearch 7+
Time Series: InfluxDB (for analytics)
```

AI/ML Stack

```yaml
Framework: TensorFlow 2.x / PyTorch
Libraries: scikit-learn, pandas, numpy
NLP: spaCy, NLTK
Model Serving: TensorFlow Serving / MLflow
```

DevOps & Infrastructure

```yaml
Containerization: Docker + Docker Compose
Orchestration: Kubernetes
Cloud Provider: AWS / Google Cloud / Azure
CI/CD: GitHub Actions / GitLab CI
Monitoring: Prometheus + Grafana
Logging: ELK Stack (Elasticsearch, Logstash, Kibana)
API Gateway: Kong / AWS API Gateway
```

Database Design

Core Entities

```sql
```

```sql
-- Users and Organizations
CREATE TABLE organizations (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    domain VARCHAR(100),
    settings JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID REFERENCES organizations(id),
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255),
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    role VARCHAR(50) DEFAULT 'member',
    is_active BOOLEAN DEFAULT true,
    last_login TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Teams
CREATE TABLE teams (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID REFERENCES organizations(id),
    name VARCHAR(255) NOT NULL,
    description TEXT,
    team_type VARCHAR(50),
    status VARCHAR(50) DEFAULT 'active',
    created_by UUID REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE team_members (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    team_id UUID REFERENCES teams(id),
    user_id UUID REFERENCES users(id),
    role VARCHAR(50) DEFAULT 'member',
    joined_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(team_id, user_id)
);
```

```sql
-- Assessment Framework
CREATE TABLE assessment_frameworks (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    description TEXT,
    version VARCHAR(20),
    question_count INTEGER,
    estimated_duration INTEGER, -- in minutes
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE assessment_questions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    framework_id UUID REFERENCES assessment_frameworks(id),
    question_text TEXT NOT NULL,
    question_type VARCHAR(50), -- likert, multiple_choice, binary
    options JSONB, -- for multiple choice questions
    category VARCHAR(100),
    order_index INTEGER,
    is_active BOOLEAN DEFAULT true
);

CREATE TABLE assessments (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id),
    framework_id UUID REFERENCES assessment_frameworks(id),
    status VARCHAR(50) DEFAULT 'in_progress',
    started_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    completed_at TIMESTAMP,
    results JSONB,
    raw_scores JSONB
);

CREATE TABLE assessment_responses (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    assessment_id UUID REFERENCES assessments(id),
    question_id UUID REFERENCES assessment_questions(id),
    response_value TEXT,
    response_score INTEGER,
    answered_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Analytics and Insights
CREATE TABLE team_analytics (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    team_id UUID REFERENCES teams(id),
    metric_type VARCHAR(100),
    metric_value DECIMAL(10,4),
    calculation_date DATE,
    metadata JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE team_insights (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    team_id UUID REFERENCES teams(id),
    insight_type VARCHAR(100),
    insight_data JSONB,
    confidence_score DECIMAL(3,2),
    generated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Indexes and Performance Optimization

```sql
-- User and authentication indexes
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_org_role ON users(organization_id, role);
CREATE INDEX idx_users_active ON users(is_active);

-- Team and membership indexes
CREATE INDEX idx_teams_org ON teams(organization_id);
CREATE INDEX idx_team_members_team ON team_members(team_id);
CREATE INDEX idx_team_members_user ON team_members(user_id);

-- Assessment indexes
CREATE INDEX idx_assessments_user ON assessments(user_id);
CREATE INDEX idx_assessments_framework ON assessments(framework_id);
CREATE INDEX idx_assessments_status ON assessments(status);
CREATE INDEX idx_assessment_responses_assessment ON assessment_responses(assessment_id);

-- Analytics indexes
CREATE INDEX idx_team_analytics_team_date ON team_analytics(team_id, calculation_date);
CREATE INDEX idx_team_insights_team_type ON team_insights(team_id, insight_type);
```

API Architecture

RESTful API Design

Base URL Structure

```
Production: https://api.psychsync.com/v1
Staging: https://staging-api.psychsync.com/v1
Development: http://localhost:8000/v1
```

Authentication Endpoints

```yaml
POST /auth/login
POST /auth/register
POST /auth/refresh
POST /auth/logout
GET  /auth/me
POST /auth/forgot-password
POST /auth/reset-password
```

User Management Endpoints

```yaml
GET    /users
GET    /users/{user_id}
PUT    /users/{user_id}
DELETE /users/{user_id}
GET    /users/{user_id}/assessments
GET    /users/{user_id}/teams
```

Team Management Endpoints

```yaml
GET    /teams
POST   /teams
GET    /teams/{team_id}
PUT    /teams/{team_id}
DELETE /teams/{team_id}
GET    /teams/{team_id}/members
POST   /teams/{team_id}/members
DELETE /teams/{team_id}/members/{user_id}
GET    /teams/{team_id}/analytics
GET    /teams/{team_id}/insights
POST   /teams/{team_id}/optimize
```

Assessment Endpoints

```yaml
GET  /assessments/frameworks
GET  /assessments/frameworks/{framework_id}
GET  /assessments/frameworks/{framework_id}/questions
POST /assessments
GET  /assessments/{assessment_id}
PUT  /assessments/{assessment_id}
POST /assessments/{assessment_id}/responses
GET  /assessments/{assessment_id}/results
```

Analytics Endpoints

```yaml
GET /analytics/dashboard
GET /analytics/teams/{team_id}
GET /analytics/organization
GET /analytics/behavioral-trends
GET /analytics/performance-metrics
```

API Response Format

```json
{
  "success": true,
  "data": {
    // Actual response data
  },
  "message": "Operation completed successfully",
  "timestamp": "2024-01-15T10:30:00Z",
  "request_id": "req_123456789"
}
```

Error Response Format

```json
```

```json
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "The provided data is invalid",
    "details": {
      "field": "email",
      "reason": "Invalid email format"
    }
  },
  "timestamp": "2024-01-15T10:30:00Z",
  "request_id": "req_123456789"
}
```

Frontend Architecture

Component Structure

```
src/
├── components/
│   ├── Auth/
│   │   ├── Login.tsx
│   │   ├── Register.tsx
│   │   ├── ForgotPassword.tsx
│   │   └── ProtectedRoute.tsx
│   ├── Dashboard/
│   │   ├── Dashboard.tsx
│   │   ├── DashboardCard.tsx
│   │   ├── MetricsOverview.tsx
│   │   └── QuickActions.tsx
│   ├── Teams/
│   │   ├── TeamList.tsx
│   │   ├── TeamCard.tsx
│   │   ├── TeamDetail.tsx
│   │   ├── CreateTeam.tsx
│   │   ├── TeamMembers.tsx
│   │   └── TeamOptimization.tsx
│   ├── Assessments/
│   │   ├── AssessmentCenter.tsx
│   │   ├── AssessmentList.tsx
│   │   ├── TakeAssessment.tsx
│   │   ├── AssessmentResults.tsx
│   │   └── AssessmentComparison.tsx
│   ├── Analytics/
│   │   ├── Analytics.tsx
│   │   ├── TeamAnalytics.tsx
│   │   ├── BehavioralTrends.tsx
│   │   └── PerformanceCharts.tsx
│   ├── Layout/
│   │   ├── Header.tsx
│   │   ├── Sidebar.tsx
│   │   ├── Footer.tsx
│   │   └── Layout.tsx
│   └── UI/
│       ├── Button.tsx
│       ├── Modal.tsx
│       ├── LoadingSpinner.tsx
│       ├── NotificationContainer.tsx
│       └── FormElements.tsx
├── contexts/
│   ├── AuthContext.tsx
│   ├── TeamContext.tsx
│   ├── NotificationContext.tsx
│   └── ThemeContext.tsx
```

```
├── hooks/
│   ├── useAuth.ts
│   ├── useTeams.ts
│   ├── useAssessments.ts
│   ├── useAnalytics.ts
│   └── useLocalStorage.ts
├── services/
│   ├── api.ts
│   ├── auth.ts
│   ├── teams.ts
│   ├── assessments.ts
│   └── analytics.ts
├── types/
│   ├── auth.ts
│   ├── teams.ts
│   ├── assessments.ts
│   └── analytics.ts
├── utils/
│   ├── helpers.ts
│   ├── validation.ts
│   ├── formatting.ts
│   └── constants.ts
└── styles/
    ├── globals.css
    ├── components.css
    └── utilities.css
```

State Management Architecture

```typescript
```

```
// Global State Structure
interface AppState {
  auth: {
    user: User | null;
    isAuthenticated: boolean;
    isLoading: boolean;
    error: string | null;
  };
  teams: {
    teams: Team[];
    currentTeam: Team | null;
    loading: boolean;
    error: string | null;
  };
  assessments: {
    assessments: Assessment[];
    frameworks: AssessmentFramework[];
    loading: boolean;
    error: string | null;
  };
  notifications: {
    notifications: Notification[];
  };
  ui: {
    sidebarOpen: boolean;
    theme: 'light' | 'dark';
    loading: boolean;
  };
}
```

Security Architecture

Authentication & Authorization

1. **JWT Tokens**
   - Access tokens (15 minutes expiry)
   - Refresh tokens (7 days expiry)
   - Secure HTTP-only cookies
2. **Role-Based Access Control (RBAC)**

   ```
   typescript
   ```

```
enum UserRole {
  SUPER_ADMIN = 'super_admin',
  ORG_ADMIN = 'org_admin',
  TEAM_LEAD = 'team_lead',
  MEMBER = 'member'
}

interface Permission {
  resource: string;
  action: string;
  conditions?: Record<string, any>;
}
```

3. **Data Encryption**
   - TLS 1.3 for data in transit
   - AES-256 for data at rest
   - bcrypt for password hashing

Security Measures

1. **Input Validation**
   - Schema validation on all API endpoints
   - XSS protection
   - SQL injection prevention
2. **Rate Limiting**
   - API endpoint rate limiting
   - Authentication attempt limiting
   - DDoS protection
3. **Audit Logging**
   - User action tracking
   - Data access logging
   - Security event monitoring

Deployment Architecture

Environment Configuration

Development

```
yaml
```

```yaml
Database: PostgreSQL (local)
Cache: Redis (local)
API: FastAPI dev server
Frontend: React dev server with HMR
Monitoring: Local logging only
```

Staging

```yaml
Infrastructure: Docker containers
Database: PostgreSQL (managed service)
Cache: Redis (managed service)
Load Balancer: NGINX
SSL: Let's Encrypt
Monitoring: Basic metrics
```

Production

```yaml
Infrastructure: Kubernetes cluster
Database: PostgreSQL cluster with read replicas
Cache: Redis cluster
CDN: CloudFlare/AWS CloudFront
Load Balancer: AWS ALB / GCP Load Balancer
SSL: Commercial certificate
Monitoring: Full observability stack
```

CI/CD Pipeline

```yaml
```

```yaml
# GitHub Actions workflow
name: PsychSync CI/CD
on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
      - name: Setup Node.js
      - name: Install dependencies
      - name: Run linting
      - name: Run tests
      - name: Generate coverage report

  security:
    runs-on: ubuntu-latest
    steps:
      - name: Security audit
      - name: Dependency vulnerability scan
      - name: SAST analysis

  build:
    needs: [test, security]
    runs-on: ubuntu-latest
    steps:
      - name: Build Docker images
      - name: Push to registry

  deploy:
    needs: build
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    steps:
      - name: Deploy to staging
      - name: Run integration tests
      - name: Deploy to production
      - name: Health check
```

Performance Considerations

Backend Optimization

1. **Database Optimization**
   - Query optimization with proper indexing
   - Connection pooling
   - Read replicas for analytics queries
   - Query result caching
2. **API Performance**
   - Response compression (gzip)
   - HTTP/2 support
   - Pagination for large datasets
   - Async processing for heavy operations
3. **Caching Strategy**
   - Redis for session and application cache
   - CDN for static assets
   - API response caching
   - Database query result caching

Frontend Optimization

1. **Bundle Optimization**
   - Code splitting by routes
   - Tree shaking for unused code
   - Lazy loading of components
   - Webpack bundle analysis
2. **Performance Monitoring**
   - Core Web Vitals tracking
   - Performance budget enforcement
   - Real User Monitoring (RUM)
   - Synthetic monitoring

Scalability Plan

Horizontal Scaling

1. **Microservices Decomposition**
   - Service separation by domain
   - Independent deployment pipelines

- Service mesh for communication
- Circuit breakers for resilience

2. **Database Scaling**
   - Read replicas for query distribution
   - Sharding strategy for large datasets
   - Connection pooling optimization
   - Query performance monitoring
3. **Auto-scaling Configuration**

```yaml
# Kubernetes HPA configuration
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: psychsync-api-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: psychsync-api
  minReplicas: 3
  maxReplicas: 50
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
```

Performance Targets

- **API Response Time**: < 200ms (95th percentile)
- **Page Load Time**: < 2 seconds (initial load)
- **Database Query Time**: < 50ms (average)
- **Availability**: 99.9% uptime

- **Concurrent Users**: 10,000+
- **Throughput**: 1,000 requests/second