Biome API & Database Design Specification

Version: 1.0
Date: January 2025
Project: Biome - AI Fitness Form Coaching
Frontend: React + TypeScript + Tailwind CSS

Table of Contents

Overview

Biome is an AI-powered fitness form coaching application that analyzes exercise videos using computer vision and provides real-time feedback. The system uses Google ADK, MediaPipe for pose detection, and Gemini 2.0 for AI coaching.

Key Features

- Video upload and webcam recording
- Real-time pose analysis using MediaPipe
- AI-powered coaching feedback via Gemini 2.0
- Exercise library with 100+ supported movements
- Progress tracking and analytics
- Form quality scoring and recommendations

System Architecture

```
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│   Frontend      │    │   Backend API   │    │   AI Services   │
│   (React)       │◄──►│   (Node.js)     │◄──►│   (Google ADK)  │
└─────────────────┘    └─────────────────┘    └─────────────────┘
                                │
                                ▼
                       ┌─────────────────┐
                       │   Database      │
                       │   (PostgreSQL)  │
                       └─────────────────┘
                                │
                                ▼
                       ┌─────────────────┐
                       │   File Storage  │
                       │   (AWS S3)      │
                       └─────────────────┘
```

Database Schema

Core Tables

1. Users

```sql
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    username VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    profile_image_url TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_login TIMESTAMP,
    is_active BOOLEAN DEFAULT true
);
```

2. Exercises

```sql
CREATE TABLE exercises (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(100) UNIQUE NOT NULL,
    category VARCHAR(50) NOT NULL, -- 'Upper Body', 'Lower Body', 'Core', 'Other'
    description TEXT,
    icon VARCHAR(10), -- Emoji icon
    is_popular BOOLEAN DEFAULT false,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Insert initial exercise data
INSERT INTO exercises (name, category, icon, is_popular) VALUES
('Squat', 'Lower Body', '🏋️', true),
('Push-up', 'Upper Body', '💪', true),
('Deadlift', 'Lower Body', '⚡', true),
('Plank', 'Core', '🧘', true),
('Lunge', 'Lower Body', '🦵', true),
('Pull-up', 'Upper Body', '🔝', true);
```

3. Analysis Sessions

```sql
CREATE TABLE analysis_sessions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    exercise_id UUID REFERENCES exercises(id),
    exercise_name VARCHAR(100) NOT NULL, -- Store custom exercise names
    video_url TEXT NOT NULL,
    video_duration DECIMAL(10,2), -- in seconds
    file_size BIGINT, -- in bytes
    mime_type VARCHAR(100),
    status VARCHAR(20) DEFAULT 'pending', -- 'pending', 'processing', 'completed', 'failed'
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    started_at TIMESTAMP,
    completed_at TIMESTAMP,
    error_message TEXT
);
```

4. Analysis Results

```sql
CREATE TABLE analysis_results (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    session_id UUID REFERENCES analysis_sessions(id) ON DELETE CASCADE,
    overall_score DECIMAL(3,1) NOT NULL, -- 0.0 to 10.0
    total_frames INTEGER NOT NULL,
    processing_time DECIMAL(10,2), -- in seconds
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

5. Form Issues

```sql
CREATE TABLE form_issues (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    result_id UUID REFERENCES analysis_results(id) ON DELETE CASCADE,
    issue_type VARCHAR(100) NOT NULL, -- 'Knee Valgus', 'Back Rounding', etc.
    severity VARCHAR(20) NOT NULL, -- 'severe', 'moderate', 'minor'
    frame_start INTEGER NOT NULL,
    frame_end INTEGER NOT NULL,
    coaching_cue TEXT NOT NULL,
    confidence_score DECIMAL(3,2), -- 0.00 to 1.00
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

6. Metrics

```sql
CREATE TABLE metrics (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    result_id UUID REFERENCES analysis_results(id) ON DELETE CASCADE,
    metric_name VARCHAR(100) NOT NULL, -- 'Knee Angle', 'Hip Angle', etc.
    actual_value VARCHAR(50) NOT NULL,
    target_value VARCHAR(50) NOT NULL,
    status VARCHAR(20) NOT NULL, -- 'good', 'warning', 'error'
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

7. Strengths

```sql
CREATE TABLE strengths (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    result_id UUID REFERENCES analysis_results(id) ON DELETE CASCADE,
    strength_text TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

8. Recommendations

```sql
CREATE TABLE recommendations (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    result_id UUID REFERENCES analysis_results(id) ON DELETE CASCADE,
    recommendation_text TEXT NOT NULL,
    priority INTEGER DEFAULT 1, -- 1 = highest priority
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

9. User Progress

```sql
CREATE TABLE user_progress (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    exercise_id UUID REFERENCES exercises(id),
    exercise_name VARCHAR(100) NOT NULL,
    average_score DECIMAL(3,1),
    total_analyses INTEGER DEFAULT 0,
    last_analysis_date TIMESTAMP,
    improvement_trend VARCHAR(20), -- 'improving', 'stable', 'declining'
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Indexes

```sql
-- Performance indexes
CREATE INDEX idx_analysis_sessions_user_id ON analysis_sessions(user_id);
CREATE INDEX idx_analysis_sessions_status ON analysis_sessions(status);
CREATE INDEX idx_analysis_sessions_created_at ON analysis_sessions(created_at);
CREATE INDEX idx_form_issues_result_id ON form_issues(result_id);
CREATE INDEX idx_metrics_result_id ON metrics(result_id);
CREATE INDEX idx_user_progress_user_exercise ON user_progress(user_id, exercise_id);
```

API Endpoints

Base URL

```
Production: https://api.biome.app/v1
Development: http://localhost:8000/v1
```

Authentication Endpoints

POST /auth/register

Register a new user

```
Request:
{
  "email": "user@example.com",
  "username": "fitnessuser",
  "password": "securepassword",
  "firstName": "John",
  "lastName": "Doe"
}

Response:
{
  "success": true,
  "data": {
    "user": {
      "id": "uuid",
      "email": "user@example.com",
      "username": "fitnessuser",
      "firstName": "John",
      "lastName": "Doe"
    },
    "token": "jwt_token_here"
  }
}
```

POST /auth/login

Authenticate user

```
Request:
{
  "email": "user@example.com",
  "password": "securepassword"
}

Response:
{
  "success": true,
  "data": {
    "user": { /* user object */ },
    "token": "jwt_token_here"
  }
}
```

Exercise Endpoints

GET /exercises

Get all exercises with optional filtering

```
Query Parameters:
- category: string (optional) - Filter by category
- popular: boolean (optional) - Get only popular exercises
- search: string (optional) - Search by name

Response:
{
  "success": true,
  "data": {
    "exercises": [
      {
        "id": "uuid",
        "name": "Squat",
        "category": "Lower Body",
        "description": "Basic squat exercise",
        "icon": "🏋",
        "isPopular": true
      }
    ]
  }
}
```

GET /exercises/categories

Get exercise categories

```
Response:
{
  "success": true,
  "data": {
    "categories": [
      {
        "name": "Upper Body",
        "count": 15
      },
      {
        "name": "Lower Body",
        "count": 12
      }
    ]
  }
}
```

Analysis Endpoints

POST /analysis/upload

Upload video for analysis

```
Request: (multipart/form-data)
- video: File (required)
- exerciseName: string (required)
- exerciseId: string (optional)

Response:
{
  "success": true,
  "data": {
    "sessionId": "uuid",
    "status": "pending",
    "estimatedProcessingTime": 30
  }
}
```

GET /analysis/sessions/:sessionId

Get analysis session status

```
Response:
{
  "success": true,
  "data": {
    "id": "uuid",
    "status": "processing",
    "progress": 45,
    "estimatedTimeRemaining": 15,
    "createdAt": "2025-01-20T10:00:00Z"
  }
}
```

GET /analysis/sessions/:sessionId/results

Get analysis results

```
Response:
{
  "success": true,
  "data": {
    "session": {
      "id": "uuid",
      "exerciseName": "Squat",
      "videoUrl": "https://s3.../video.mp4",
      "createdAt": "2025-01-20T10:00:00Z"
    },
    "result": {
      "overallScore": 7.2,
      "totalFrames": 120,
      "processingTime": 25.5
    },
    "issues": [
      {
        "type": "Knee Valgus",
        "severity": "moderate",
        "frameStart": 23,
        "frameEnd": 45,
        "coachingCue": "Your right knee is collapsing inward 12°. Push both knees out to track over your toes.",
        "confidenceScore": 0.85
      }
    ],
    "metrics": [
      {
        "name": "Knee Angle (bottom)",
        "actual": "87°",
        "target": "90°",
        "status": "warning"
      }
    ],
    "strengths": [
      "Consistent tempo (good control)",
      "Balanced left/right symmetry"
    ],
    "recommendations": [
      "Strengthening glutes (knee stability)",
      "Ankle mobility drills"
    ]
  }
}
```

GET /analysis/sessions

Get user's analysis history

```
Query Parameters:
- page: number (default: 1)
- limit: number (default: 10)
- exerciseId: string (optional)
- dateFrom: string (optional)
- dateTo: string (optional)

Response:
{
  "success": true,
  "data": {
    "sessions": [
      {
        "id": "uuid",
        "exerciseName": "Squat",
        "overallScore": 7.2,
        "status": "completed",
        "createdAt": "2025-01-20T10:00:00Z"
      }
    ],
    "pagination": {
      "page": 1,
      "limit": 10,
      "total": 25,
      "totalPages": 3
    }
  }
}
```

Progress Endpoints

GET /progress/summary

Get user's progress summary

```
Response:
{
  "success": true,
  "data": {
    "totalAnalyses": 15,
    "averageScore": 7.8,
    "improvementTrend": "improving",
    "favoriteExercise": "Squat",
    "recentScores": [7.2, 7.5, 8.1, 7.9, 8.3]
  }
}
```

GET /progress/exercises

Get progress by exercise

```
Response:
{
  "success": true,
  "data": {
    "exercises": [
      {
        "exerciseName": "Squat",
        "averageScore": 7.8,
        "totalAnalyses": 5,
        "lastAnalysisDate": "2025-01-20T10:00:00Z",
        "improvementTrend": "improving"
      }
    ]
  }
}
```

User Endpoints

GET /user/profile

Get user profile

```
Response:
{
  "success": true,
  "data": {
    "id": "uuid",
    "email": "user@example.com",
    "username": "fitnessuser",
    "firstName": "John",
    "lastName": "Doe",
    "profileImageUrl": "https://s3.../profile.jpg",
    "createdAt": "2025-01-15T10:00:00Z",
    "lastLogin": "2025-01-20T10:00:00Z"
  }
}
```

PUT /user/profile

Update user profile

```
Request:
{
  "firstName": "John",
  "lastName": "Doe",
  "profileImage": File (optional)
}

Response:
{
  "success": true,
  "data": {
    "user": { /* updated user object */ }
  }
}
```

Data Models

TypeScript Interfaces (for reference)

```
// User Models
interface User {
  id: string;
  email: string;
  username: string;
  firstName?: string;
  lastName?: string;
  profileImageUrl?: string;
  createdAt: string;
  updatedAt: string;
  lastLogin?: string;
  isActive: boolean;
}

// Exercise Models
interface Exercise {
  id: string;
  name: string;
  category: "Upper Body" | "Lower Body" | "Core" | "Other";
  description?: string;
  icon?: string;
  isPopular: boolean;
  createdAt: string;
  updatedAt: string;
}

// Analysis Models
interface AnalysisSession {
  id: string;
  userId: string;
  exerciseId?: string;
  exerciseName: string;
  videoUrl: string;
  videoDuration?: number;
  fileSize?: number;
  mimeType?: string;
  status: "pending" | "processing" | "completed" | "failed";
  createdAt: string;
  startedAt?: string;
  completedAt?: string;
  errorMessage?: string;
}

interface AnalysisResult {
  id: string;
  sessionId: string;
  overallScore: number;
  totalFrames: number;
  processingTime?: number;
  createdAt: string;
}

interface FormIssue {
  id: string;
  resultId: string;
  issueType: string;
  severity: "severe" | "moderate" | "minor";
  frameStart: number;
  frameEnd: number;
  coachingCue: string;
  confidenceScore?: number;
  createdAt: string;
}

interface Metric {
  id: string;
  resultId: string;
  metricName: string;
  actualValue: string;
  targetValue: string;
  status: "good" | "warning" | "error";
  createdAt: string;
}

// API Response Models
interface ApiResponse<T> {
  success: boolean;
  data?: T;
  error?: {
    code: string;
    message: string;
    details?: any;
  };
  pagination?: {
    page: number;
    limit: number;
    total: number;
    totalPages: number;
  };
}
```

Authentication & Authorization

JWT Token Structure

```
{
  "header": {
    "alg": "HS256",
    "typ": "JWT"
  },
  "payload": {
    "sub": "user_id",
    "email": "user@example.com",
    "username": "fitnessuser",
    "iat": 1640995200,
    "exp": 1641081600,
    "role": "user"
  }
}
```

Authorization Levels

- **Public**: Landing page, How it works, Exercise list
- **User**: Upload videos, view analysis results, progress tracking
- **Admin**: User management, system analytics

Security Headers

```
Authorization: Bearer <jwt_token>
Content-Type: application/json
X-Requested-With: XMLHttpRequest
```

File Storage

AWS S3 Bucket Structure

```
biome-videos/
├── users/
│   └── {user_id}/
│       └── sessions/
│           └── {session_id}/
│               ├── original.mp4
│               ├── processed.mp4
│               └── thumbnails/
│                   └── thumbnail.jpg
└── temp/
    └── {temp_id}/
        └── upload.mp4
```

File Upload Process

1. **Pre-signed URL**: Generate temporary upload URL
2. **Direct Upload**: Client uploads directly to S3
3. **Processing**: Move to permanent location after analysis
4. **Cleanup**: Remove temporary files after 24 hours

File Size Limits

- **Maximum file size**: 100MB
- **Supported formats**: MP4, MOV, AVI, WebM
- **Maximum duration**: 5 minutes
- **Resolution**: 720p minimum recommended

Real-time Communication

WebSocket Events

Connection

```
// Client connects to analysis session
ws://localhost:8000/ws/analysis/{sessionId}
```

Events

```
// Progress updates
{
  "type": "progress",
  "data": {
    "sessionId": "uuid",
    "progress": 45,
    "currentStep": "vision_processing",
    "estimatedTimeRemaining": 15
  }
}

// Agent status updates
{
  "type": "agent_status",
  "data": {
    "sessionId": "uuid",
    "visionAgent": "processing",
    "coachingAgent": "waiting",
    "tasks": [
      {
        "name": "Extracting body landmarks",
        "status": "complete"
      }
    ]
  }
}

// Analysis completion
{
  "type": "analysis_complete",
  "data": {
    "sessionId": "uuid",
    "resultId": "uuid",
    "overallScore": 7.2
  }
}

// Error notifications
{
  "type": "error",
  "data": {
    "sessionId": "uuid",
    "error": "Video processing failed",
    "code": "PROCESSING_ERROR"
  }
}
```

Error Handling

HTTP Status Codes

- **200**: Success
- **201**: Created
- **400**: Bad Request
- **401**: Unauthorized
- **403**: Forbidden
- **404**: Not Found
- **413**: Payload Too Large
- **422**: Unprocessable Entity
- **429**: Too Many Requests
- **500**: Internal Server Error

Error Response Format

```
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid video format",
    "details": {
      "field": "video",
      "reason": "Unsupported file type"
    }
  }
}
```

Common Error Codes

- **VALIDATION_ERROR**: Input validation failed
- **AUTHENTICATION_ERROR**: Invalid credentials
- **AUTHORIZATION_ERROR**: Insufficient permissions
- **FILE_TOO_LARGE**: Video exceeds size limit
- **UNSUPPORTED_FORMAT**: Invalid video format
- **PROCESSING_ERROR**: AI analysis failed
- **RATE_LIMIT_EXCEEDED**: Too many requests
- **SESSION_NOT_FOUND**: Analysis session not found

Performance Considerations

Database Optimization

- **Connection pooling**: 20-50 connections
- **Query optimization**: Use indexes and explain plans
- **Caching**: Redis for session data and results
- **Partitioning**: Partition analysis_sessions by date

API Performance

- **Rate limiting**: 100 requests/minute per user
- **Response compression**: Gzip compression
- **CDN**: CloudFront for static assets
- **Caching**: 5-minute cache for exercise data

File Processing

- **Async processing**: Queue-based video analysis
- **Progress tracking**: Real-time WebSocket updates
- **Error recovery**: Retry failed analyses
- **Cleanup**: Automated temp file removal

Monitoring

- **Health checks**: `/health` endpoint
- **Metrics**: Prometheus + Grafana
- **Logging**: Structured JSON logs
- **Alerts**: Slack notifications for errors

Deployment Considerations

Environment Variables

```
# Database
DATABASE_URL=postgresql://user:pass@localhost:5432/biome
REDIS_URL=redis://localhost:6379

# AWS
AWS_ACCESS_KEY_ID=your_key
AWS_SECRET_ACCESS_KEY=your_secret
AWS_REGION=us-east-1
S3_BUCKET_NAME=biome-videos

# JWT
JWT_SECRET=your_jwt_secret
JWT_EXPIRES_IN=24h

# Google ADK
GOOGLE_ADK_API_KEY=your_api_key
GEMINI_API_KEY=your_gemini_key

# Server
PORT=8000
NODE_ENV=production
```

Docker Configuration

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
COPY . .
EXPOSE 8000
CMD ["npm", "start"]
```

Health Check Endpoint

```
GET /health
{
  "status": "healthy",
  "timestamp": "2025-01-20T10:00:00Z",
  "services": {
    "database": "connected",
    "redis": "connected",
    "s3": "connected",
    "ai_services": "available"
  }
}
```

Testing Strategy

Unit Tests

- API endpoint testing
- Database model validation
- Business logic testing

Integration Tests

- End-to-end analysis workflow
- File upload and processing
- WebSocket communication

Load Testing

- Concurrent video uploads
- Database query performance
- WebSocket connection limits

Security Considerations

Data Protection

- **Encryption**: TLS 1.3 for all communications
- **Storage**: Encrypted S3 buckets
- **Database**: Encrypted at rest
- **PII**: Minimal data collection

Input Validation

- **File validation**: MIME type and size checks
- **SQL injection**: Parameterized queries
- **XSS protection**: Input sanitization
- **CSRF protection**: Token validation

Access Control

- **JWT tokens**: Short expiration times
- **Rate limiting**: Prevent abuse
- **CORS**: Restricted origins
- **API keys**: Rotated regularly

This specification provides a comprehensive foundation for implementing the Biome backend API and database. The design supports the frontend requirements while maintaining scalability, security, and performance.

Next Steps:

1. Set up PostgreSQL database with the provided schema
2. Implement authentication and user management
3. Create file upload and processing pipeline
4. Integrate with Google ADK and MediaPipe
5. Implement real-time WebSocket communication
6. Add monitoring and logging
7. Deploy to production environment