# **Backend Implementation Specification [BACK-SPEC-MVP-001]**

## **1. Database Schema**

### **Users**

sql

Copy

CREATE TABLE users (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

email VARCHAR(255) UNIQUE NOT NULL,

password\_hash VARCHAR(255) NOT NULL,

role VARCHAR(50) NOT NULL,

first\_name VARCHAR(100),

last\_name VARCHAR(100),

phone VARCHAR(50),

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

verification\_status VARCHAR(50) DEFAULT 'pending',

profile\_image\_url VARCHAR(255)

);

CREATE TABLE user\_profiles (

user\_id UUID PRIMARY KEY REFERENCES users(id),

bio TEXT,

location JSON,

specializations TEXT[],

experience\_years INTEGER,

certifications TEXT[],

social\_links JSON,

preferences JSON

);

### **Horses**

sql

Copy

CREATE TABLE horses (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

owner\_id UUID REFERENCES users(id),

name VARCHAR(255) NOT NULL,

breed VARCHAR(100) NOT NULL,

date\_of\_birth DATE,

gender VARCHAR(50),

height DECIMAL(4,2),

color VARCHAR(100),

discipline TEXT[],

price\_range JSON,

status VARCHAR(50) DEFAULT 'active',

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE horse\_details (

horse\_id UUID PRIMARY KEY REFERENCES horses(id),

description TEXT,

achievements TEXT[],

medical\_history JSON,

training\_history JSON,

competition\_history JSON,

lineage JSON,

location JSON

);

CREATE TABLE horse\_media (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

horse\_id UUID REFERENCES horses(id),

media\_type VARCHAR(50),

url VARCHAR(255),

caption TEXT,

order\_index INTEGER,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

);

### **Events**

sql

Copy

CREATE TABLE events (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

creator\_id UUID REFERENCES users(id),

title VARCHAR(255) NOT NULL,

description TEXT,

event\_type VARCHAR(100),

start\_date TIMESTAMP WITH TIME ZONE,

end\_date TIMESTAMP WITH TIME ZONE,

location JSON,

max\_participants INTEGER,

status VARCHAR(50) DEFAULT 'scheduled',

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE event\_participants (

event\_id UUID REFERENCES events(id),

user\_id UUID REFERENCES users(id),

status VARCHAR(50),

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

PRIMARY KEY (event\_id, user\_id)

);

### **Deals**

sql

Copy

CREATE TABLE deals (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

horse\_id UUID REFERENCES horses(id),

seller\_id UUID REFERENCES users(id),

buyer\_id UUID REFERENCES users(id),

status VARCHAR(50) DEFAULT 'initiated',

price DECIMAL(10,2),

currency VARCHAR(3) DEFAULT 'USD',

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE deal\_events (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

deal\_id UUID REFERENCES deals(id),

event\_type VARCHAR(100),

description TEXT,

metadata JSON,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

);

## **2. GraphQL Schema**

graphql

Copy

type User {

id: ID!

email: String!

role: String!

firstName: String

lastName: String

phone: String

verificationStatus: String!

profileImageUrl: String

profile: UserProfile

createdAt: DateTime!

updatedAt: DateTime!

}

type UserProfile {

bio: String

location: Location

specializations: [String!]

experienceYears: Int

certifications: [String!]

socialLinks: SocialLinks

preferences: JSON

}

type Horse {

id: ID!

owner: User!

name: String!

breed: String!

dateOfBirth: Date

gender: String

height: Float

color: String

discipline: [String!]

priceRange: PriceRange

status: String!

details: HorseDetails

media: [HorseMedia!]

createdAt: DateTime!

updatedAt: DateTime!

}

type HorseDetails {

description: String

achievements: [String!]

medicalHistory: JSON

trainingHistory: JSON

competitionHistory: JSON

lineage: JSON

location: Location

}

type Event {

id: ID!

creator: User!

title: String!

description: String

eventType: String!

startDate: DateTime!

endDate: DateTime!

location: Location!

maxParticipants: Int

status: String!

participants: [EventParticipant!]

createdAt: DateTime!

updatedAt: DateTime!

}

type Deal {

id: ID!

horse: Horse!

seller: User!

buyer: User

status: String!

price: Float

currency: String!

events: [DealEvent!]

createdAt: DateTime!

updatedAt: DateTime!

}

*# Queries and Mutations will be implemented in the next phase*

## **3. Initial API Implementation Priority**

1. Authentication Service

typescript

Copy

@Injectable()

export class AuthService {

async register(input: RegisterInput): Promise<AuthResponse>

async login(input: LoginInput): Promise<AuthResponse>

async verifyEmail(token: string): Promise<boolean>

async refreshToken(token: string): Promise<AuthResponse>

}

1. Horse Management Service

typescript

Copy

@Injectable()

export class HorseService {

async createHorse(input: CreateHorseInput): Promise<Horse>

async updateHorse(id: string, input: UpdateHorseInput): Promise<Horse>

async getHorse(id: string): Promise<Horse>

async listHorses(filter: HorseFilterInput): Promise<HorsesResponse>

async deleteHorse(id: string): Promise<boolean>

}

1. User Management Service

typescript

Copy

@Injectable()

export class UserService {

async createProfile(input: CreateProfileInput): Promise<UserProfile>

async updateProfile(input: UpdateProfileInput): Promise<UserProfile>

async getProfile(id: string): Promise<UserProfile>

async listUsers(filter: UserFilterInput): Promise<UsersResponse>

}

1. Event Management Service

typescript

Copy

@Injectable()

export class EventService {

async createEvent(input: CreateEventInput): Promise<Event>

async updateEvent(id: string, input: UpdateEventInput): Promise<Event>

async getEvent(id: string): Promise<Event>

async listEvents(filter: EventFilterInput): Promise<EventsResponse>

async registerForEvent(eventId: string, userId: string): Promise<boolean>

}

## **4. Security Implementation**

1. Authentication Middleware

typescript

Copy

@Injectable()

export class AuthMiddleware implements NestMiddleware {

use(req: Request, res: Response, next: Function) {

*// JWT validation*

*// Role verification*

*// Rate limiting*

}

}

1. File Upload Service

typescript

Copy

@Injectable()

export class FileUploadService {

async uploadFile(file: FileUpload): Promise<FileResponse>

async deleteFile(fileId: string): Promise<boolean>

async getSignedUrl(fileId: string): Promise<string>

}

## **5. Testing Requirements**

1. Unit Tests (minimum 80% coverage)
2. Integration Tests for all API endpoints
3. E2E Tests for critical flows:
   * User registration/login
   * Horse profile CRUD
   * Event management
   * Deal creation

## **6. Error Handling**

typescript

Copy

export class ApiException extends Error {

constructor(

public readonly code: string,

public readonly status: number,

public readonly message: string,

public readonly details?: any

) {

super(message);

}

}

*// Error codes and messages will be standardized*