Y3nko Ride-Sharing API Documentation

Overview

Y3nko is a comprehensive ride-sharing GraphQL API built with Node.js, TypeScript, and Apollo Server. The API provides complete functionality for a ride-sharing platform including user management, trip creation, booking system, payments, reviews, and notifications.

🚀 Quick Start

Server Information

- **GraphQL Endpoint**: `http://localhost:4000/graphql`
- **GraphQL Playground**: Available in development mode
- **Authentication**: Firebase Authentication
- **Database**: PostgreSQL (Neon.tech/Supabase compatible)

Starting the Server

```
npm install
npm run dev
```

🔐 Authentication

The API uses Firebase Authentication. Include the Firebase ID token in the Authorization header:

```
Authorization: Bearer <firebase-id-token>
```

Authentication Context

All authenticated requests provide access to the current user through the GraphQL context:

- `user.uid` - Firebase user ID
- `user.email` - User email
- `user.name` - User display name

📊 GraphQL Schema

Core Types

User

```
type User {
  id: ID!
  email: String!
  phone: String
  firstName: String!
  lastName: String!
  profileImageUrl: String
  userType: UserType!
  isVerified: Boolean!
  isActive: Boolean!
  createdAt: String!
  updatedAt: String!
  driverProfile: DriverProfile
}

enum UserType {
  RIDER
  DRIVER
  ADMIN
}
```

DriverProfile

```
type DriverProfile {
  id: ID!
  userId: ID!
  licenseNumber: String!
  licenseExpiry: String!
  licenseImageUrl: String
  drivingExperienceYears: Int
  backgroundCheckStatus: BackgroundCheckStatus!
  averageRating: Float!
  totalTrips: Int!
  isAvailable: Boolean!
  createdAt: String!
  user: User!
  vehicle: Vehicle
}

enum BackgroundCheckStatus {
  PENDING
  APPROVED
  REJECTED
}
```

Trip

```
type Trip {
  id: ID!
  driverId: ID!
  originCity: String!
  destinationCity: String!
  originCoordinates: Coordinates!
  destinationCoordinates: Coordinates!
  departureTime: String!
  availableSeats: Int!
  pricePerSeat: Float!
  tripStatus: TripStatus!
  tripType: TripType!
  returnDepartureTime: String
  description: String
  createdAt: String!
  driver: User!
  bookings: [Booking!]!
}

type Coordinates {
  latitude: Float!
  longitude: Float!
}

enum TripStatus {
  SCHEDULED
  ACTIVE
  COMPLETED
  CANCELLED
}

enum TripType {
  ONE_WAY
  ROUND_TRIP
}
```

Booking

```
type Booking {
  id: ID!
  tripId: ID!
  riderId: ID!
  seatsBooked: Int!
  totalAmount: Float!
  commissionAmount: Float!
  bookingStatus: BookingStatus!
  paymentStatus: PaymentStatus!
  paymentMethod: PaymentMethod!
  pickupLocation: String
  pickupCoordinates: Coordinates
  specialRequests: String
  createdAt: String!
  trip: Trip!
  rider: User!
  payment: Payment
}

enum BookingStatus {
  PENDING
  CONFIRMED
  CANCELLED
  COMPLETED
}

enum PaymentStatus {
  PENDING
  PAID
  FAILED
  REFUNDED
}

enum PaymentMethod {
  PAYSTACK
  BANK_TRANSFER
  CASH
}
```

Payment

```
type Payment {
  id: ID!
  bookingId: ID!
  amount: Float!
  paymentMethod: PaymentMethod!
  paystackReference: String
  paystackTransactionId: String
  paymentStatus: PaymentStatus!
  gatewayResponse: String
  createdAt: String!
  booking: Booking!
}
```

Review

```
type Review {
  id: ID!
  bookingId: ID!
  reviewerId: ID!
  revieweeId: ID!
  rating: Int!
  comment: String
  createdAt: String!
  booking: Booking!
  reviewer: User!
  reviewee: User!
}
```

Notification

```
type Notification {
  id: ID!
  userId: ID!
  title: String!
  message: String!
  notificationType: NotificationType!
  isRead: Boolean!
  metadata: String
  createdAt: String!
  user: User!
}

enum NotificationType {
  BOOKING_CONFIRMED
  TRIP_STARTED
  TRIP_COMPLETED
  PAYMENT_RECEIVED
  REVIEW_RECEIVED
  SYSTEM_UPDATE
}
```

🔍 Queries

User Queries

Get Current User

```
query Me {
  me {
    id
    email
    firstName
    lastName
    userType
    driverProfile {
      id
      averageRating
      totalTrips
      isAvailable
    }
  }
}
```

Get User by ID

```
query GetUser($id: ID!) {
  user(id: $id) {
    id
    firstName
    lastName
    userType
    isVerified
  }
}
```

Trip Queries

Get Trip by ID

```
query GetTrip($id: ID!) {
  trip(id: $id) {
    id
    originCity
    destinationCity
    departureTime
    availableSeats
    pricePerSeat
    tripStatus
    driver {
      firstName
      lastName
      driverProfile {
        averageRating
      }
    }
  }
}
```

Get My Trips

```
query MyTrips($status: TripStatus) {
  myTrips(status: $status) {
    id
    originCity
    destinationCity
    departureTime
    tripStatus
    bookings {
      id
      seatsBooked
      bookingStatus
    }
  }
}
```

Booking Queries

Get My Bookings

```
query MyBookings($status: BookingStatus) {
  myBookings(status: $status) {
    id
    seatsBooked
    totalAmount
    bookingStatus
    paymentStatus
    trip {
      originCity
      destinationCity
      departureTime
      driver {
        firstName
        lastName
      }
    }
  }
}
```

Get Trip Bookings

```
query TripBookings($tripId: ID!) {
  tripBookings(tripId: $tripId) {
    id
    seatsBooked
    bookingStatus
    rider {
      firstName
      lastName
    }
  }
}
```

Payment Queries

Get Payment History

```
query PaymentHistory($first: Int, $after: String) {
  paymentHistory(first: $first, after: $after) {
    id
    amount
    paymentMethod
    paymentStatus
    createdAt
    booking {
      trip {
        originCity
        destinationCity
      }
    }
  }
}
```

Notification Queries

Get My Notifications

```
query MyNotifications($unreadOnly: Boolean) {
  myNotifications(unreadOnly: $unreadOnly) {
    id
    title
    message
    notificationType
    isRead
    createdAt
  }
}
```

Review Queries

Get User Reviews

```
query UserReviews($userId: ID!, $first: Int) {
  userReviews(userId: $userId, first: $first) {
    id
    rating
    comment
    createdAt
    reviewer {
      firstName
      lastName
    }
  }
}
```

🖉 Mutations

User Mutations

Create User

```
mutation CreateUser($input: CreateUserInput!) {
  createUser(input: $input) {
    id
    email
    firstName
    lastName
    userType
  }
}
```

Input:

```
input CreateUserInput {
  email: String!
  phone: String
  firstName: String!
  lastName: String!
  userType: UserType!
}
```

Update User

```
mutation UpdateUser($input: UpdateUserInput!) {
  updateUser(input: $input) {
    id
    firstName
    lastName
    profileImageUrl
  }
}
```

Create Driver Profile

```
mutation CreateDriverProfile($input: CreateDriverProfileInput!) {
  createDriverProfile(input: $input) {
    id
    licenseNumber
    licenseExpiry
    backgroundCheckStatus
  }
}
```

Toggle Driver Availability

```
mutation ToggleDriverAvailability {
  toggleDriverAvailability {
    id
    isAvailable
  }
}
```

Trip Mutations

Create Trip

```
mutation CreateTrip($input: CreateTripInput!) {
  createTrip(input: $input) {
    id
    originCity
    destinationCity
    departureTime
    availableSeats
    pricePerSeat
    tripStatus
  }
}
```

Input:

```
input CreateTripInput {
  originCity: String!
  destinationCity: String!
  originCoordinates: CoordinatesInput!
  destinationCoordinates: CoordinatesInput!
  departureTime: String!
  availableSeats: Int!
  pricePerSeat: Float!
  tripType: TripType!
  returnDepartureTime: String
  description: String
}

input CoordinatesInput {
  latitude: Float!
  longitude: Float!
}
```

Cancel Trip

```
mutation CancelTrip($id: ID!) {
  cancelTrip(id: $id) {
    id
    tripStatus
  }
}
```

Start Trip

```
mutation StartTrip($id: ID!) {
  startTrip(id: $id) {
    id
    tripStatus
  }
}
```

Complete Trip

```
mutation CompleteTrip($id: ID!) {
  completeTrip(id: $id) {
    id
    tripStatus
  }
}
```

Booking Mutations

Create Booking

```
mutation CreateBooking($input: CreateBookingInput!) {
  createBooking(input: $input) {
    id
```

```
      seatsBooked
      totalAmount
      bookingStatus
      paymentStatus
    }
  }
```

Input:

```
  input CreateBookingInput {
    tripId: ID!
    seatsBooked: Int!
    pickupLocation: String
    pickupCoordinates: CoordinatesInput
    specialRequests: String
  }
```

Confirm Booking

```
  mutation ConfirmBooking($id: ID!) {
    confirmBooking(id: $id) {
      id
      bookingStatus
    }
  }
```

Cancel Booking

```
  mutation CancelBooking($id: ID!) {
    cancelBooking(id: $id) {
      id
      bookingStatus
    }
  }
```

Payment Mutations

Initialize Payment

```
  mutation InitializePayment($input: InitializePaymentInput!) {
    initializePayment(input: $input) {
      id
      amount
      paystackReference
      paymentStatus
    }
  }
```

Input:

```
input InitializePaymentInput {
  bookingId: ID!
  paymentMethod: PaymentMethod!
}
```

Verify Payment

```
mutation VerifyPayment($reference: String!) {
  verifyPayment(reference: $reference) {
    id
    paymentStatus
    paystackTransactionId
  }
}
```

Process Refund

```
mutation ProcessRefund($paymentId: ID!) {
  processRefund(paymentId: $paymentId) {
    id
    paymentStatus
  }
}
```

Review Mutations

Create Review

```
mutation CreateReview($bookingId: ID!, $rating: Int!, $comment: String) {
  createReview(bookingId: $bookingId, rating: $rating, comment: $comment) {
    id
    rating
    comment
    createdAt
  }
}
```

Notification Mutations

Mark Notification as Read

```
mutation MarkNotificationAsRead($id: ID!) {
  markNotificationAsRead(id: $id) {
    id
    isRead
  }
}
```

Mark All Notifications as Read

```
mutation MarkAllNotificationsAsRead {
  markAllNotificationsAsRead
}
```

🔄 Subscriptions

Real-time Updates

Trip Location Updates

```
subscription TripLocationUpdate($tripId: ID!) {
  tripLocationUpdate(tripId: $tripId) {
    tripId
    currentLocation {
      latitude
      longitude
    }
    estimatedArrival
  }
}
```

Trip Status Updates

```
subscription TripStatusUpdate($tripId: ID!) {
  tripStatusUpdate(tripId: $tripId) {
    tripId
    status
    timestamp
  }
}
```

Booking Status Updates

```
subscription BookingStatusUpdate($bookingId: ID!) {
  bookingStatusUpdate(bookingId: $bookingId) {
    bookingId
    status
    timestamp
  }
}
```

New Notifications

```
subscription NewNotification {
  newNotification {
    id
    title
    message
    notificationType
    createdAt
  }
}
```

🛠 Development

Project Structure

```
src/
├── config/
│   └── database.ts          # Database configuration
├── middleware/
│   ├── auth.ts              # Firebase authentication
│   ├── errorHandler.ts      # Error handling
│   └── rateLimiter.ts       # Rate limiting
├── resolvers/
│   ├── userResolvers.ts     # User operations
│   ├── tripResolvers.ts     # Trip operations
│   ├── bookingResolvers.ts  # Booking operations
│   ├── paymentResolvers.ts  # Payment operations
│   ├── reviewResolvers.ts   # Review operations
│   ├── notificationResolvers.ts # Notification operations
│   └── index.ts             # Resolver exports
├── schema/
│   └── index.ts             # GraphQL schema definitions
├── types/
│   └── index.ts             # TypeScript type definitions
├── utils/
```

```
│   └── logger.ts            # Logging utility
└── index.ts                 # Server entry point
```

Environment Variables

```
# Server Configuration
NODE_ENV=development
PORT=4000
LOG_LEVEL=debug
CORS_ORIGINS=https://studio.apollographql.com,http://localhost:3000

# Database Configuration
DATABASE_URL=postgresql://username:password@host:port/database

# Firebase Configuration
FIREBASE_PROJECT_ID=your-project-id
FIREBASE_CLIENT_EMAIL=your-service-account-email
FIREBASE_PRIVATE_KEY=your-private-key
FIREBASE_DATABASE_URL=your-database-url
```

Database Schema

The API expects the following PostgreSQL tables:

- `users` - User information
- `driver_profiles` - Driver-specific data
- `vehicles` - Vehicle information
- `trips` - Trip details
- `bookings` - Booking records
- `payments` - Payment transactions
- `reviews` - User reviews
- `notifications` - User notifications

Error Handling

The API implements comprehensive error handling:

- Authentication errors (401)
- Authorization errors (403)
- Validation errors (400)
- Not found errors (404)
- Internal server errors (500)

Rate Limiting

Rate limiting is applied to the GraphQL endpoint:

- 100 requests per 15 minutes per IP

- Configurable through environment variables

Logging

Comprehensive logging with different levels:

- `error` - Error messages
- `warn` - Warning messages
- `info` - Informational messages
- `debug` - Debug information

🚀 Deployment

Production Checklist

1. Set `NODE_ENV=production`
2. Configure production database
3. Set up proper CORS origins
4. Configure Firebase production credentials
5. Set up monitoring and logging
6. Configure rate limiting
7. Set up SSL/TLS certificates

Health Check

The API provides a health check endpoint:

```
GET /health
```

Response:

```json
{
  "status": "OK",
  "timestamp": "2025-01-27T19:00:00.000Z",
  "uptime": 3600
}
```

📝 Examples

Complete User Flow Example

1. **Create User Account**

```
mutation {
  createUser(input: {
    email: "john@example.com"
    firstName: "John"
    lastName: "Doe"
    userType: DRIVER
  }) {
    id
    email
    userType
  }
}
```

2. **Create Driver Profile**

```
mutation {
  createDriverProfile(input: {
    licenseNumber: "DL123456789"
    licenseExpiry: "2026-12-31"
    drivingExperienceYears: 5
  }) {
    id
    licenseNumber
    backgroundCheckStatus
  }
}
```

3. **Create a Trip**

```
mutation {
  createTrip(input: {
    originCity: "Lagos"
    destinationCity: "Abuja"
    originCoordinates: { latitude: 6.5244, longitude: 3.3792 }
    destinationCoordinates: { latitude: 9.0765, longitude: 7.3986 }
    departureTime: "2025-01-28T08:00:00Z"
    availableSeats: 3
    pricePerSeat: 15000.00
    tripType: ONE_WAY
  }) {
    id
    originCity
    destinationCity
    tripStatus
  }
}
```

4. **Book a Trip (as Rider)**

```
mutation {
  createBooking(input: {
    tripId: "trip-id-here"
    seatsBooked: 2
    pickupLocation: "Victoria Island"
  }) {
    id
    totalAmount
    bookingStatus
  }
}
```

5. **Initialize Payment**

```
mutation {
  initializePayment(input: {
    bookingId: "booking-id-here"
    paymentMethod: PAYSTACK
  }) {
    id
    paystackReference
    amount
  }
}
```

🤝 Support

For API support and questions:

- Check the GraphQL Playground for interactive documentation
- Review error messages for detailed information
- Ensure proper authentication headers are included
- Verify database connectivity for full functionality

📄 License

This API is part of the Y3nko ride-sharing platform.