ENRICHIFY Data Platform API Documentation

Overview

The ENRICHIFY Data Platform API is a Node.js Express server that handles lead generation data processing, user management, and automated email notifications. The system manages demographic targeting data, processes array fields, and sends detailed HTML email notifications for leads and user registrations.

Table of Contents

Setup & Configuration

Dependencies

```javascript
const express = require('express');
const cors = require('cors');
const mongoose = require('mongoose');
const nodemailer = require('nodemailer');
const dataModel = require('./data');
const userModel = require('./user');
```

Database Connection

```javascript
```

```
mongoose.connect('mongodb+srv://user:user@cluster0.pfn059x.mongodb.net/Cluster0?retryWrites=true&w=majority
  serverSelectionTimeoutMS: 5000,
  socketTimeoutMS: 45000,
  family: 4
});
```

Email Configuration

```javascript
const transporter = nodemailer.createTransporter({
  service: 'gmail',
  auth: {
    user: 'leads@enrichifydata.com',
    pass: 'obeahkmflwnesojn'
  }
});
```

Database Models

User Model (`userModel`)

Expected fields:

- `name`: String - User's full name
- `email`: String - User's email address (unique)
- `phone`: String - User's phone number
- `address`: String - User's address
- `password`: String - User's password

Data Model (`dataModel`)

Expected fields include all lead generation parameters (see Data Processing section for complete list).

API Endpoints

1. Submit Lead Data

POST `/sendData`

Processes lead generation data and sends email notifications.

Request Body

```json
{
  "user": {
    "name": "John Doe",
    "email": "john@example.com"
  },
  "campaign_type": "Standard Lead Generation",
  "phone_options": "Include mobile",
  "dedup_option": "Standard dedup",
  "supression_option": "Email suppression",
  "zip_codes": ["12345", "67890"],
  "states": ["CA", "NY"],
  "cities": ["Los Angeles", "New York"],
  "dwelling_type": ["Single Family"],
  "home_owner": ["Owner"],
  "hh_income": ["$50K-$75K"],
  "age_group": ["25-35"],
  "credit_rating": ["Excellent"],
  "ethnic_code": ["Caucasian"],
  "propensity_to_give": ["High"],
  "donor_affinity_range": ["$100-$500"],
  "occupation": "Software Engineer"
}
```

Response

Success (200)

```json
{
  "message": "Success"
}
```

Error (400)

```json
```

```
{
  "error": "Error message"
}
```

2. User Registration

POST `/register`

Creates a new user account and sends welcome email.

Request Body

```
json

{
  "name": "John Doe",
  "email": "john@example.com",
  "phone": "+1234567890",
  "address": "123 Main St, City, State"
}
```

Response

Success (200)

```
json

{
  "message": "User created successfully"
}
```

Error (400)

```
json

{
  "error": "User already exists"
}
```

3. User Login

POST `/login`

Authenticates user credentials.

Request Body

```json
{
  "email": "john@example.com",
  "password": "userpassword"
}
```

Response

Success (200)

```json
{
  "user": {
    "_id": "userId",
    "name": "John Doe",
    "email": "john@example.com",
    // ... other user fields
  }
}
```

Error (400)

```json
{
  "error": "Email not found" // or "Invalid password"
}
```

4. Password Reset Request

POST `/forgetpassword`

Sends password reset email to user.

Request Body

```json
```

```
{
  "email": "john@example.com"
}
```

Response

Success (200)

```
json

{
  "message": "Password reset link successfully sent"
}
```

Error (400)

```
json

{
  "error": "Invalid email"
}
```

5. Reset Password

POST `/resetpassword`

Updates user password using reset token.

Request Body

```
json

{
  "password": "newpassword",
  "id": "userId"
}
```

Response

Success (200)

```
json
```

```
{
  "message": "Successfully updated password"
}
```

Error (400)

```
json

{
  "error": "Something went wrong please try again"
}
```

Data Processing

Array Field Processing

The system processes various demographic and targeting fields that can be in different formats:

Supported Array Fields

- `zip_codes`: Geographic zip code targeting
- `states`: State-level targeting
- `cities`: City-level targeting
- `dwelling_type`: Housing type preferences
- `home_owner`: Homeowner status
- `hh_income`: Household income ranges
- `individuals`: Individual targeting data
- `martial_status`: Marital status options
- `age_group`: Age demographic ranges
- `credit_rating`: Credit score categories
- `ethnic_code`: Ethnicity targeting
- `propensity_to_give`: Donation likelihood
- `donor_affinity_range`: Donation amount ranges
- `turning_65`: Age-specific targeting
- `pet_range`: Pet ownership data
- `propensity_range`: General propensity scores

- `outdoor_range`: Outdoor activity interests
- `sports_and_fitness_range`: Sports/fitness interests
- `travel_and_hobbies_range`: Travel/hobby interests
- `genre_range`: Entertainment genre preferences

Processing Functions

```javascript
function processArrayField(field) {
  // Handles string JSON, objects with value properties, and arrays
  // Returns clean array of string values
}

function processDataArrays(data) {
  // Processes all array fields in the data object
  // Returns cleaned data object
}
```

Additional Configuration Fields

- **Operators**: `donor_affinity_op`, `pet_op`, `propensity_op`, `outdoor_op`, `sports_and_fitness_op`, `travel_and_hobbies_op`, `genre_op`
- **Campaign Settings**: `campaign_type`, `phone_options`, `dedup_option`, `supression_option`
- **Personal Data**: `occupation`

Email Templates

Lead Notification Email

Recipient: `shipmate2134@gmail.com`
Sender: `leads@enrichifydata.com`
Subject: "New lead from Datazapp tool"

Features:

- Modern dark theme with purple gradient design
- Responsive grid layout
- Organized sections for different data categories
- Interactive hover effects
- ENRICHIFY branding with logo

- Timestamp and status indicators
- Next steps checklist

User Registration Email

Recipient: User's email address
Sender: `leads@enrichifydata.com`
Subject: "New user registered"

Features:

- Welcome message with user details
- Account information display
- Call-to-action button for login
- Next steps guidance
- Professional branding

Password Reset Email

Recipient: User's email address
Sender: `leads@enrichifydata.com`
Subject: "Password reset link"

Features:

- Secure reset link with user ID
- 24-hour expiration notice
- Security warnings and best practices
- Alternative link copying option
- Professional security-focused design

Error Handling

Common Error Responses

- **400 Bad Request**: Invalid input data, user already exists, authentication failures
- **Database Errors**: MongoDB connection issues, validation errors
- **Email Errors**: SMTP configuration issues, sending failures

Error Format

```json
{
  "error": "Descriptive error message"
}
```

Security Considerations

Current Implementation Issues

⚠️ **Security Warnings**:

1. **Exposed Credentials**: Database connection string and email password are hardcoded
2. **No Password Hashing**: Passwords stored in plain text
3. **No Input Validation**: Missing data sanitization and validation
4. **No Rate Limiting**: API endpoints are not protected against abuse
5. **No Authentication Middleware**: Endpoints lack proper authentication
6. **CORS**: Currently allows all origins

Recommended Security Improvements

```javascript
// Use environment variables
const dbUri = process.env.MONGODB_URI;
const emailPassword = process.env.EMAIL_PASSWORD;

// Hash passwords
const bcrypt = require('bcrypt');
const hashedPassword = await bcrypt.hash(password, 10);

// Add input validation
const { body, validationResult } = require('express-validator');

// Add rate limiting
const rateLimit = require('express-rate-limit');

// Add authentication middleware
const jwt = require('jsonwebtoken');
```

Server Configuration

Port: 5000

Environment: Development

CORS: Enabled for all origins

Frontend Integration

The API is designed to work with a React frontend hosted at:

- Production: `https://enrichify-data-request.vercel.app`
- Login: `https://enrichify-data-request.vercel.app/login`
- Password Reset: `https://enrichify-data-request.vercel.app/resetpassword`

Monitoring & Logging

Currently implements basic console logging:

```javascript
console.log(data); // Lead data logging
console.log(info); // Email sending confirmation
```

For production, consider implementing:

- Structured logging (Winston)
- Error tracking (Sentry)
- Performance monitoring
- Request logging middleware

API Testing

Sample cURL Commands

```bash
```

```
# Register User
curl -X POST http://localhost:5000/register \
  -H "Content-Type: application/json" \
  -d '{
    "name": "John Doe",
    "email": "john@example.com",
    "phone": "+1234567890",
    "address": "123 Main St"
  }'

# Submit Lead Data
curl -X POST http://localhost:5000/sendData \
  -H "Content-Type: application/json" \
  -d '{
    "user": {"name": "John Doe", "email": "john@example.com"},
    "zip_codes": ["12345"],
    "states": ["CA"]
  }'
```

Version Information

- **Node.js**: Compatible with Node.js 14+
- **Express**: 4.x
- **MongoDB**: 4.x+
- **Mongoose**: 6.x+