Profile Creation Issue - Backend Fix Required

Priority: 🔴 High - Blocks user onboarding
Status: Frontend ready, awaiting backend fix
Date: Current
Affected Endpoint: `PUT /api/v1/profiles/me`

📋 Executive Summary

Users cannot complete their profile setup because the backend `PUT /api/v1/profiles/me` endpoint does not support creating new profiles. It only works for updating existing profiles, but new users don't have a profile record yet. This blocks the entire onboarding flow.

Current Behavior:

- Frontend calls `PUT /api/v1/profiles/me` with profile data
- Backend returns:
  `{ "success": false, "message": "An unexpected error occurred while updating the profile.", "error": "Profile not found" }`
- User sees error: "Profile not found"

Required Behavior:

- Frontend calls `PUT /api/v1/profiles/me` with profile data
- Backend should create profile if it doesn't exist (upsert behavior)
- Backend should return success with created/updated profile data

🔍 Technical Details

Current Backend Behavior

1. **PUT** `/api/v1/profiles/me` - Returns "Profile not found" if profile doesn't exist
   - Status: ❌ Only supports updates, not creation
   - Expected: ✅ Should support upsert (create if doesn't exist)
2. **POST** `/api/v1/profiles` - Does not exist (returns 404)
   - Status: ❌ Not implemented
   - Note: Frontend no longer attempts this endpoint

Frontend Implementation

Complete Profile Page (`/profile/complete-profile`):

- ✅ Does NOT fetch profile data (no GET requests) - profile doesn't exist yet
- ✅ Uses `PUT /api/v1/profiles/me` to create profile
- ✅ Handles errors gracefully with clear user messages
- ✅ Validates data before sending (username, displayName, bio, niche required)

Request Format:

```
PUT /api/v1/profiles/me
Authorization: Bearer <token>
Content-Type: application/json

{
  "username": "john_doe",
  "displayName": "John Doe",
  "bio": "Content creator and influencer",
  "niche": ["fitness", "lifestyle"],
  "location": {
    "city": "New York",
    "state": "NY",
    "country": "USA"
  },
  "socialLinks": {
    "instagram": "https://instagram.com/johndoe",
    "tiktok": "https://tiktok.com/@johndoe"
  },
  "avatar": "https://cdn.example.com/avatar.jpg"  // Optional
}
```

Expected Response (Success):

```
{
  "success": true,
  "profile": {
    "id": "profile_id",
    "userId": "user_id",
    "username": "john_doe",
    "displayName": "John Doe",
    "bio": "Content creator and influencer",
    "niche": ["fitness", "lifestyle"],
    "location": {
      "city": "New York",
      "state": "NY",
      "country": "USA"
    },
    "socialLinks": {
      "instagram": "https://instagram.com/johndoe",
      "tiktok": "https://tiktok.com/@johndoe"
    },
    "avatar": "https://cdn.example.com/avatar.jpg",
    "createdAt": "2024-01-01T00:00:00Z",
    "updatedAt": "2024-01-01T00:00:00Z"
  }
}
```

Current Error Response:

```
{
  "success": false,
  "message": "An unexpected error occurred while updating the profile.",
  "error": "Profile not found"
}
```

✅ Recommended Solution: Implement Upsert in PUT Endpoint

Why This Approach:

- ✅ Simplest for frontend (only one endpoint to call)
- ✅ Most RESTful approach (PUT is idempotent)
- ✅ Handles both create and update seamlessly
- ✅ No need to check if profile exists first
- ✅ Better user experience (no additional API calls)

Implementation Guide

Step 1: Update PUT Endpoint Handler

Modify `PUT /api/v1/profiles/me` to support upsert behavior:

Node.js/Express Example:

```
// PUT /api/v1/profiles/me
router.put('/me', authenticateUser, async (req, res) => {
  try {
    const userId = req.user.id;
    const profileData = req.body;

    // Validate required fields
    if (!profileData.username || !profileData.displayName || !profileData.bio || !profileData.niche) {
      return res.status(400).json({
        success: false,
        message: 'Missing required fields: username, displayName, bio, and niche are required'
      });
    }

    // Validate niche is an array with at least one item
    if (!Array.isArray(profileData.niche) || profileData.niche.length === 0) {
      return res.status(400).json({
        success: false,
        message: 'Niche must be an array with at least one item'
      });
    }

    // Check if username is already taken by another user
    const existingProfile = await Profile.findOne({
      username: profileData.username,
      userId: { $ne: userId }  // Exclude current user
    });

    if (existingProfile) {
      return res.status(409).json({
        success: false,
        message: 'Username is already taken',
        error: 'Username already exists'
      });
    }

    // Upsert: Find or create profile
    let profile = await Profile.findOne({ userId });

    if (!profile) {
      // Create new profile if doesn't exist
      profile = new Profile({
        userId,
        ...profileData,
        createdAt: new Date(),
        updatedAt: new Date()
      });
    } else {
      // Update existing profile
      Object.assign(profile, profileData);
      profile.updatedAt = new Date();
    }

    await profile.save();
```

```
    res.status(200).json({
      success: true,
      profile: profile.toObject()  // or profile.toJSON() depending on your ORM
    });

  } catch (error) {
    console.error('Profile save error:', error);

    // Handle duplicate key errors (e.g., unique username constraint)
    if (error.code === 11000 || error.name === 'MongoError') {
      return res.status(409).json({
        success: false,
        message: 'Username is already taken',
        error: 'Duplicate username'
      });
    }

    res.status(500).json({
      success: false,
      message: 'An unexpected error occurred while saving the profile',
      error: error.message
    });
  }
});
```

Python/FastAPI Example:

```python
@router.put("/profiles/me")
async def update_profile(
    profile_data: ProfileUpdate,
    current_user: User = Depends(get_current_user)
):
    try:
        # Validate required fields
        if not all([profile_data.username, profile_data.display_name,
                    profile_data.bio, profile_data.niche]):
            raise HTTPException(
                status_code=400,
                detail="Missing required fields: username, displayName, bio, and niche are required"
            )

        if not profile_data.niche or len(profile_data.niche) == 0:
            raise HTTPException(
                status_code=400,
                detail="Niche must be an array with at least one item"
            )

        # Check if username is taken by another user
        existing_profile = await Profile.find_one(
            Profile.username == profile_data.username,
            Profile.user_id != current_user.id
        )

        if existing_profile:
            raise HTTPException(
                status_code=409,
                detail="Username is already taken"
            )

        # Upsert: Find or create profile
        profile = await Profile.find_one(Profile.user_id == current_user.id)

        if not profile:
            # Create new profile
            profile = Profile(
                user_id=current_user.id,
                **profile_data.dict(),
                created_at=datetime.utcnow(),
                updated_at=datetime.utcnow()
            )
        else:
            # Update existing profile
            for key, value in profile_data.dict(exclude_unset=True).items():
                setattr(profile, key, value)
            profile.updated_at = datetime.utcnow()

        await profile.save()

        return {
            "success": True,
            "profile": profile.dict()
        }
```

```
    except HTTPException:
        raise
    except Exception as e:
        logger.error(f"Profile save error: {str(e)}")
        raise HTTPException(
            status_code=500,
            detail="An unexpected error occurred while saving the profile"
        )
```

Step 2: Field Validation Requirements

Required Fields:

- `username` (string, 3-30 chars, alphanumeric + underscore, unique)
- `displayName` (string, max 50 chars)
- `bio` (string, max 500 chars)
- `niche` (array of strings, 1-5 items)

Optional Fields:

- `location` (object: `{ city, state, country }`)
- `socialLinks` (object with platform URLs)
- `avatar` (string URL)

Validation Rules:

- Username must be unique across all users
- Niche array must contain 1-5 items
- Social links should be valid URLs (optional validation)
- Avatar URL should be valid (optional validation)

Step 3: Error Handling

Status Codes:

- `200` - Success (profile created or updated)
- `400` - Bad Request (validation errors, missing required fields)
- `401` - Unauthorized (invalid or missing token)
- `409` - Conflict (username already taken)
- `500` - Internal Server Error

Error Response Format:

```
{
  "success": false,
  "message": "Human-readable error message",
  "error": "Error code or technical details"
}
```

🔄 Alternative Solutions (If Upsert Not Possible)

Option 2: Create Profile During Registration

If you prefer to keep PUT for updates only, create an empty profile during registration:

```
// POST /api/v1/auth/register
router.post('/register', async (req, res) => {
  try {
    // ... existing user creation code ...

    // Create empty profile record immediately after user creation
    await Profile.create({
      userId: user.id,
      username: null,
      displayName: null,
      bio: null,
      niche: [],
      location: null,
      socialLinks: {},
      avatar: null,
      createdAt: new Date(),
      updatedAt: new Date()
    });

    // ... rest of registration logic ...
  } catch (error) {
    // Handle errors
  }
});
```

Pros:

- Keeps PUT endpoint simple (update only)
- Profile always exists for new users

Cons:

- Requires changes to registration flow
- Creates empty records that might never be used

Option 3: Add POST Endpoint (Not Recommended)

If you must keep PUT as update-only, implement `POST /api/v1/profiles`:

```
// POST /api/v1/profiles
router.post('/', authenticateUser, async (req, res) => {
  try {
    const userId = req.user.id;

    // Check if profile already exists
    const existingProfile = await Profile.findOne({ userId });
    if (existingProfile) {
      return res.status(409).json({
        success: false,
        message: 'Profile already exists. Use PUT /profiles/me to update.'
      });
    }

    // Create new profile
    const profile = await Profile.create({
      userId,
      ...req.body
    });

    res.status(201).json({ success: true, profile });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: 'Failed to create profile',
      error: error.message
    });
  }
});
```

Note: This approach requires frontend changes (not implemented, would need to add POST fallback).

🧪 Testing Checklist

After implementing the fix, please test:

1. New User Profile Creation

- ☐ Register a new user
- ☐ Navigate to `/profile/complete-profile`
- ☐ Fill in all required fields (username, displayName, bio, niche)
- ☐ Submit the form
- ☐ Verify profile is created successfully
- ☐ Verify response includes complete profile data
- ☐ Verify profile appears in database

2. Profile Update (Existing User)

- ☐ Login with existing user who has a profile
- ☐ Navigate to `/profile/edit` or update profile
- ☐ Modify profile fields

- [ ] Submit the form
- [ ] Verify profile is updated successfully
- [ ] Verify existing profile data is preserved where not updated

3. Validation Tests

- [ ] Try to create profile with missing required fields → Should return 400
- [ ] Try to use duplicate username → Should return 409
- [ ] Try to submit with invalid niche (empty array) → Should return 400
- [ ] Try without authentication token → Should return 401

4. Edge Cases

- [ ] Create profile, then immediately update it → Should work
- [ ] Create profile with optional fields (location, socialLinks, avatar) → Should save correctly
- [ ] Update profile with partial data → Should merge with existing data

5. Network Verification

- [ ] Check Network tab in browser DevTools
- [ ] Verify `PUT /api/v1/profiles/me` returns status 200
- [ ] Verify response includes `success: true` and profile data
- [ ] Verify no 404 or “Profile not found” errors

📊 Database Schema Reference

Expected Profile Model:

```
{
  userId: ObjectId,          // Required, unique, references User
  username: String,          // Required, unique, 3-30 chars
  displayName: String,       // Required, max 50 chars
  bio: String,               // Required, max 500 chars
  niche: [String],           // Required, array of 1-5 strings
  location: {
    city: String,            // Optional
    state: String,           // Optional
    country: String          // Optional
  },
  socialLinks: {
    instagram: String,       // Optional URL
    tiktok: String,          // Optional URL
    youtube: String,         // Optional URL
    twitch: String,          // Optional URL
    twitter: String,         // Optional URL
    linkedin: String         // Optional URL
  },
  avatar: String,            // Optional URL
  createdAt: Date,           // Auto-generated
  updatedAt: Date            // Auto-updated
}
```

🔗 Related Endpoints

Current Implementation:

- `GET /api/v1/profiles/me` - Fetch current user's profile (used in edit profile page)
- `PUT /api/v1/profiles/me` - Update profile (needs upsert support)

Other Profile Endpoints:

- `POST /api/v1/profiles/upload-media` - Upload avatar/image (already working)
- `GET /api/v1/users/me` - Get user data with profile (used in Dashboard)

📝 Implementation Notes

Frontend Behavior

- **Complete Profile Page:** Only uses PUT, no GET requests (profile doesn't exist yet)
- **Edit Profile Page:** Uses GET to fetch existing data, then PUT to update
- **Dashboard:** Uses GET `/users/me` for verification and profile completeness check

Authentication

- All requests require `Authorization: Bearer <token>` header
- Token is obtained from `localStorage.getItem('accessToken')`

- User ID is extracted from the authenticated token

Data Cleaning

Frontend sends cleaned data:

- All string fields are trimmed
- Empty strings are removed
- Undefined values are removed
- Empty arrays/objects are removed

🚨 Current Error Messages

What Users See:

```
"Profile not found. Please contact support - your profile needs to be initialized.
The backend should automatically create a profile record when you register or
support creating profiles via PUT request."
```

What Backend Returns:

```
{
  "success": false,
  "message": "An unexpected error occurred while updating the profile.",
  "error": "Profile not found"
}
```

✅ Success Criteria

The fix is complete when:

1. ✅ New users can successfully create their profile via `PUT /api/v1/profiles/me`
2. ✅ Existing users can still update their profile via `PUT /api/v1/profiles/me`
3. ✅ Response returns status 200 with `success: true` and profile data
4. ✅ Username uniqueness is enforced
5. ✅ All validation rules are enforced
6. ✅ Error messages are clear and helpful

📞 Questions or Issues?

If you have questions about:

- Frontend expectations or request format
- Field validation requirements
- Error handling requirements

- Testing scenarios

Please contact the frontend team or refer to this document.

Document Version: 2.0
Last Updated: Current
Frontend Status: ✅ Ready and waiting for backend fix
Backend Status: ⏳ Pending implementation