Mindful Message - Code Review Report

Executive Summary

This document provides a methodical analysis of the Mindful Message codebase, identifying bugs, code smells, and security concerns organized by workflow. The application is a mental health counselling platform with multiple user roles (Counsellor, Organisation, Admin, Super Admin) managing referrals, appointments, billing, and psychometric assessments.

Part 1: System Understanding

Core Entities & Relationships

Entity	Description
User	Base account with OAuth/credentials auth
Profile	Extended user info with role assignment
Counsellor	Professional details, specialities, verification status
Organisation	Companies that submit referrals
Referral	Client cases submitted for counselling
Appointment	Sessions between counsellor and referral
Invoice	Billing records for completed appointments
Contract	Agreements between organisations and platform

User Roles Hierarchy

```
PUBLIC (0) → CLIENT (1) → COUNSELLOR (2) → ORGANISATION (3) → ADMIN (4) → SUPER_ADMIN (5)
```

Key Workflows

1. **Invite → Signup → Onboarding** (Counsellor/Organisation)
2. **Referral Submission → Approval → Assignment** (Organisation/Admin)
3. **Appointment Scheduling → Completion → Billing** (Counsellor)
4. **Psychometric Assessment Distribution** (Counsellor)

5. **Super Admin Management** (Separate auth system)

Part 2: Information Flow Analysis

Authentication Flow

```
User Login (OAuth/Credentials)
    ↓
NextAuth JWT Callback
    ↓
Fetch Profile/Organisation/AdminUser from DB
    ↓
Session populated with role, organisationId, adminUser
    ↓
Middleware route protection
```

Referral Flow

```
Organisation creates Referral
    ↓
Admin reviews (PENDING → APPROVED/REJECTED)
    ↓
Admin assigns to Counsellor
    ↓
Counsellor schedules Appointments
    ↓
Appointment completed → Invoice created
```

Part 3: Bugs & Issues by Workflow

Workflow 1: Authentication & Authorization

BUG 1.1: Missing Authorization Check in Send Invite API

Severity: HIGH | **File:** `app/api/invites/send/route.ts`

typescript

```typescript
// Line 17-19: Only checks if user is authenticated, not their role
const session = await auth();
if (!session?.user?.id) {
  return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });
}
```

Issue: Any authenticated user can send invites for ANY role including SUPER_ADMIN. There's no permission check to verify the caller has `createInvite` permission.

Fix: Add role check:

```typescript
import { hasPermission } from '@/server/auth/permissions';
if (!hasPermission(session.user.role, 'createInvite')) {
  return NextResponse.json({ error: 'Forbidden' }, { status: 403 });
}
```

BUG 1.2: Inconsistent Role Check in Invite Action

Severity: MEDIUM | File: `server/action/invite-action.ts`

```typescript
// Line 229: Only creates profile for ADMIN role, not others
if (invite.role === Role.ADMIN) {
  // ... creates profile
}
```

Issue: Uses `==` instead of `===` for comparison (minor), but more importantly, the logic only handles ADMIN invites during acceptance, not COUNSELLOR or ORGANISATION invites.

BUG 1.3: Super Admin JWT Secret Fallback

Severity: HIGH | File: `middleware.ts`

```typescript
// Line 15-16: Hardcoded fallback secret
const JWT_SECRET = new TextEncoder().encode(
  process.env.SUPER_ADMIN_JWT_SECRET || 'your-super-secret-key-change-this-in-production'
);
```

Issue: If the environment variable is not set, the system uses a well-known default secret, making it trivial to forge super admin tokens.

Fix: Throw an error if the secret is not configured instead of using a fallback.

BUG 1.4: Organisation Lookup by Email Creates Security Hole

Severity: MEDIUM | **File:** `auth.ts`

```typescript
// Lines 104-114: Organisation fetched by contactEmail
const organisation = await prisma.organisation.findFirst({
  where: {
    contactEmail: userData?.email,
  },
  // ...
});
```

Issue: If two organisations have the same contact email, `findFirst` returns arbitrary results. Additionally, if a user changes their email to match an organisation's contactEmail, they could gain access to that organisation's data.

Fix: Use a proper foreign key relationship instead of email matching.

BUG 1.5: Missing Middleware Protection for Dashboard Root

Severity: LOW | **File:** `middleware.ts`

The middleware only matches `/dashboard/:path*` but `/dashboard` itself (root) isn't caught by the redirect logic since it checks specific paths like `/dashboard/counsellor`.

Workflow 2: Referral Management

BUG 2.1: Missing Organisation Scope Validation

Severity: HIGH | **File:** `server/action/referrals-action.ts`

```typescript
```

```typescript
// Line 173-176: getReferralByIdAction allows any ORGANISATION user to view any referral
export async function getReferralByIdAction(id: string) {
  // ...
  await requireAnyRole([Role.ADMIN, Role.SUPER_ADMIN, Role.ORGANISATION]);
  const referral = await getReferralById(id);
  // No check that Organisation user can only see their own referrals
}
```

Issue: An ORGANISATION user can view referrals from OTHER organisations by knowing/guessing the referral ID.

Fix: Add scope validation:

```typescript
if (session.user.role === Role.ORGANISATION) {
  if (referral.organisationId !== session.user.organisationId) {
    return { success: false, error: 'Forbidden' };
  }
}
```

BUG 2.2: Race Condition in Referral Assignment

Severity: MEDIUM | File: `server/data/referrals-data.ts`

```typescript
// assignReferralToCounsellor has no transaction or optimistic locking
export async function assignReferralToCounsellor(
  referralId: string,
  counsellorId: string,
  assignedByUserId: string
) {
  const referral = await prisma.referral.update({
    where: { id: referralId },
    // No check for current assignStatus
```

Issue: Two admins could simultaneously assign the same referral to different counsellors. The last one wins without any conflict detection.

Fix: Use optimistic locking or a transaction with a WHERE clause checking current status.

BUG 2.3: No Validation of Counsellor Assignment Eligibility

Severity: MEDIUM | File: `server/action/referrals-action.ts`

```typescript
// assignReferralAction doesn't verify counsellor is verified
export async function assignReferralAction(referralId: string, counsellorId: string) {
  // ... just calls assignReferralToCounsellor directly
```

Issue: A referral could be assigned to an unverified or rejected counsellor. While `getAvailableCounsellors` filters for verified counsellors, the assignment action doesn't re-verify.

Workflow 3: Appointment & Scheduling

BUG 3.1: No Ownership Verification on Appointment Operations

Severity: HIGH | File: `server/action/appointments-action.ts`

```typescript
// Lines 244-276: updateAppointmentAction has no ownership check
export async function updateAppointmentAction(id: string, data: {...}) {
  const session = await auth();
  if (!session || !session.user) {
    return { success: false, error: 'Unauthorized' };
  }
  // Immediately updates without checking if user owns this appointment
  const appointment = await updateAppointment(id, data);
}
```

Issue: Any authenticated user can update ANY appointment by knowing/guessing the appointment ID. Same issue exists for `deleteAppointmentAction` and `updateAppointmentStatusAction`.

Fix: Add ownership verification:

```typescript
const appointment = await getAppointmentById(id);
const counsellor = await getCounsellorByUserId(session.user.id);
if (appointment.counsellorId !== counsellor?.id) {
  return { success: false, error: 'Forbidden' };
}
```

BUG 3.2: Appointment Double-Booking Not Prevented

Severity: MEDIUM | **File:** `server/data/appointments-data.ts`

The `createAppointment` function doesn't check for overlapping appointments for the same counsellor or referral.

Fix: Add overlap check before creation:

```typescript
const overlapping = await prisma.appointment.findFirst({
  where: {
    counsellorId: data.counsellorId,
    status: { notIn: ['CANCELLED'] },
    OR: [
      { startTime: { gte: data.startTime, lt: data.endTime } },
      { endTime: { gt: data.startTime, lte: data.endTime } },
    ],
  },
});
```

BUG 3.3: No Session Funding Limit Enforcement

Severity: MEDIUM | **File:** `server/action/appointments-action.ts`

Referrals have a `sessionFunding` field (default 12 sessions), but there's no check when creating appointments to ensure the limit isn't exceeded.

Workflow 4: Billing & Invoicing

BUG 4.1: Invoice Number Race Condition

Severity: HIGH | **File:** `server/action/billing-action.ts`

```typescript
// Lines 235-236: Non-atomic invoice number generation
const invoiceCount = await prisma.invoice.count();
const invoiceNumber = `INV-${new Date().getFullYear()}-${String(invoiceCount + 1).padStart(4, '0')}`;
```

Issue: Concurrent requests could generate duplicate invoice numbers because `count()` and `create()` aren't atomic.

Fix: Use a database sequence or atomic counter, or catch unique constraint violations and retry.

BUG 4.2: No Prevention of Double-Invoicing

Severity: MEDIUM | **File:** `server/action/billing-action.ts`

While `createInvoiceFromAppointmentAction` checks `if (appointment.invoice)`, the bulk action `bulkCreateInvoicesAction` doesn't handle partial failures gracefully - it may create invoices for some appointments but not others.

BUG 4.3: Invoice Deletion Without Cascade Handling

Severity: LOW | **File:** `server/action/billing-action.ts`

```
typescript

// deleteInvoiceAction deletes invoice but InvoiceItems are cascade-deleted
// However, if invoice is marked as PAID, deletion should probably be prevented
```

Workflow 5: Psychometric Assessments

BUG 5.1: Token Expiry Not Checked for Feedback Submission

Severity: MEDIUM | **File:** Form submission routes

The token generation includes expiry, but the verification logic should ensure forms can't be submitted after token expiration. Need to verify the form submission endpoints properly validate token expiry.

BUG 5.2: No Rate Limiting on Assessment Form Submissions

Severity: LOW | **Files:** `app/submit/*/route.ts`

Assessment forms can be submitted multiple times with the same token or by the same referral. While FormRequest tracks pending status, there's no prevention of spam submissions.

Workflow 6: Super Admin Management

BUG 6.1: Verification Code Reuse Not Prevented

Severity: MEDIUM | **File:** `lib/verification-code.ts`

```typescript
// verifyCode marks code as verified but doesn't delete it
await prisma.verificationCode.update({
  where: { id: verificationCode.id },
  data: { verified: true },
});
```

Issue: While the query checks `verified: false`, there's a window where the same code could be verified multiple times in concurrent requests.

Fix: Use a transaction or delete the code after verification instead of marking it.

BUG 6.2: Duplicate Super Admin Sessions Possible

Severity: LOW | **File:** Super admin auth routes

The super admin uses a separate JWT-based auth (cookie: `super-admin-token`) that coexists with NextAuth sessions. A user could potentially have both a regular session and a super admin session active.

Workflow 7: Onboarding

BUG 7.1: Organisation Details Can Be Created Without Invite Verification

Severity: MEDIUM | **File:** `server/action/orgs-action.ts`

```typescript
// saveOrganisationDetailsAction only checks role, not invite acceptance
const userProfile = await prisma.profile.findUnique({
  where: { userId },
  select: { role: true },
});
if (userProfile.role !== Role.ORGANISATION) {
  throw new Error('Access denied...');
}
```

Issue: If a user's role is set to ORGANISATION through any means (not just invite), they can create organisation records.

BUG 7.2: Profile Role Can Be Overwritten

Severity: MEDIUM | **File:** server/data/profile-data.ts

The profile update function should not allow role changes through normal profile updates - role should only be set through invite acceptance or admin action.

Part 4: Code Smells & Maintenance Issues

CS-1: Inconsistent Error Handling Patterns

Some actions throw errors:

```typescript
throw new Error('Unauthorized - Please sign in');
```

Others return error objects:

```typescript
return { success: false, error: 'Unauthorized' };
```

Recommendation: Standardize on return objects for server actions to avoid unhandled exceptions.

CS-2: Dynamic Imports Used Unnecessarily

```typescript
// In multiple files:
const { prisma } = await import('@/lib/prisma');
const { getOrganisationByEmail } = await import('@/server/data/orgs-data');
```

These dynamic imports add complexity without clear benefit. Static imports would be cleaner.

CS-3: Repeated Session/Role Checking Logic

The pattern of checking session, then role, then extracting user ID is repeated in almost every action. This should be abstracted into a higher-order function or decorator pattern.

CS-4: Missing Type Safety on Prisma Queries

```typescript
const where: any = {}; // Multiple files
```

Using `any` for Prisma where clauses loses type safety. Use Prisma's generated types.

CS-5: No Input Sanitization

Email and text inputs are used directly without sanitization:

```typescript
email: email.toLowerCase() // Only lowercased, not sanitized
```

Should validate email format, sanitize text inputs for XSS prevention.

CS-6: Missing Database Indexes

The Prisma schema has some indexes but is missing several that would improve performance:

- `Referral.organisationId` for filtering referrals by org
- `Appointment.status` for filtering appointments
- `Invoice.status + organisationId` for billing queries

CS-7: No Soft Delete Implementation

Most entities are hard-deleted. For audit compliance and data recovery, soft deletes (with `deletedAt` timestamp) would be preferable.

Part 5: Security Recommendations

SEC-1: Add Rate Limiting

- Login endpoints
- Verification code requests

- Invite sending
- Assessment form submissions

SEC-2: Add Audit Logging

Track all sensitive operations:

- User authentication events
- Role changes
- Referral status changes
- Invoice creation/modification
- Counsellor verification

SEC-3: Implement CSRF Protection

While NextAuth provides some protection, ensure all server actions validate origin.

SEC-4: Add Data Encryption

Sensitive data like phone numbers and medical notes should be encrypted at rest.

SEC-5: Implement Proper Session Invalidation

When user role changes or user is deactivated, invalidate all their sessions.

Summary of Critical Issues

Priority	Bug ID	Description
CRITICAL	1.1	Missing auth check in invite API allows privilege escalation
CRITICAL	1.3	Hardcoded JWT secret fallback
CRITICAL	3.1	No ownership check on appointment operations
HIGH	2.1	Organisation can view other orgs' referrals
HIGH	4.1	Race condition in invoice number generation
MEDIUM	1.4	Organisation lookup by email is insecure
MEDIUM	2.2	Race condition in referral assignment
MEDIUM	3.2	No double-booking prevention

Priority	Bug ID	Description
MEDIUM	6.1	Verification code reuse possible

Recommendations for Immediate Action

1. **Add authorization checks to ALL API routes** - Don't rely on frontend to restrict access
2. **Implement ownership verification** before allowing operations on resources
3. **Remove hardcoded secrets** and fail fast if configuration is missing
4. **Add database transactions** for multi-step operations
5. **Implement comprehensive audit logging** for compliance
6. **Add rate limiting** to prevent abuse
7. **Review and standardize error handling** across all actions