CareLinkAI Reliability & Safety Checklist

This document provides a comprehensive checklist of all reliability and safety measures implemented in CareLinkAI.

✅ Error Handling

Backend Error Handling

- [x] Standardized error response format (`ApiErrorResponse`)
- [x] Error codes enum for consistent error identification
- [x] Centralized error handling utility (`handleApiError`)
- [x] Prisma error mapping (P2002, P2025, etc.)
- [x] Error logging with context
- [x] Safe error messages (no sensitive data leaked to client)

Frontend Error Handling

- [x] Toast notification system (react-hot-toast)
- [x] Custom `ToastProvider` with consistent styling
- [x] Client error utilities (`showErrorToast`, `showSuccessToast`)
- [x] Error message extraction from API responses
- [x] Loading state toasts with promises

Usage Example

```
import { handleApiError, createErrorResponse, ErrorCode } from '@/lib/errors/api-errors';
import { showErrorToast, showSuccessToast } from '@/lib/errors/client-errors';

// Backend
try {
  // ... your code
} catch (error) {
  return handleApiError(error, { userId, action: 'create' });
}

// Frontend
try {
  const response = await fetch('/api/...');
  const data = await handleApiResponse(response);
  showSuccessToast('Operation successful!');
} catch (error) {
  showErrorToast(error);
}
```

✅ Input Validation

Server-Side Validation

- [x] Zod schemas for all major forms
- [x] Email validation with normalization
- [x] Password strength requirements (8+ chars, upper, lower, number)
- [x] Phone number format validation
- [x] URL validation and sanitization
- [x] File type and size validation
- [x] Numeric ranges (age, rate, experience, etc.)
- [x] String length limits

Available Schemas

- `registerSchema` - User registration
- `loginSchema` - User login
- `updateProfileSchema` - Profile updates
- `familyProfileSchema` - Family-specific fields
- `caregiverProfileSchema` - Caregiver profiles
- `providerProfileSchema` - Provider profiles
- `credentialSchema` - Credential uploads
- `createLeadSchema` - Lead submissions
- `sendMessageSchema` - Messaging
- `fileUploadSchema` - File uploads
- `marketplaceSearchSchema` - Search/filters

Usage Example

```
import { registerSchema } from '@/lib/validation/schemas';

const result = registerSchema.safeParse(body);
if (!result.success) {
  return createErrorResponse(
    ErrorCode.VALIDATION_ERROR,
    'Invalid input',
    400,
    { errors: result.error.format() }
  );
}
```

✅ Security

XSS Protection

- [x] DOMPurify integration (isomorphic)
- [x] HTML sanitization for rich text fields
- [x] Plain text sanitization (strip all HTML)
- [x] URL scheme validation (block javascript:, data:, vbscript:)
- [x] Filename sanitization (prevent path traversal)

- [x] Object-level sanitization utility

Available Functions

- `sanitizeHtml()` - Allow limited HTML tags
- `sanitizePlainText()` - Strip all HTML
- `sanitizeUserInput()` - General user input
- `sanitizeRichText()` - Bio, notes, descriptions
- `sanitizeUrl()` - URL validation
- `sanitizeFilename()` - File upload safety
- `sanitizeObject()` - Sanitize all fields in an object

Usage Example

```
import { sanitizeObject } from '@/lib/validation/sanitize';

const sanitized = sanitizeObject(data, {
  richTextFields: ['bio', 'description'],
  urlFields: ['website', 'photoUrl'],
});
```

✅ External Interactions

S3 Upload Robustness

- [x] Retry logic (3 attempts with exponential backoff)
- [x] File type validation (whitelist: JPG, PNG, WEBP, PDF)
- [x] File size validation (5MB images, 10MB documents)
- [x] Presigned URL generation with retries
- [x] Client-side upload helper with retries
- [x] Progress callback support
- [x] Error logging for upload failures

Database Robustness

- [x] Transaction wrapper with retry logic
- [x] Retryable error detection (connection, timeout)
- [x] Exponential backoff for retries
- [x] Query execution with retry helper
- [x] Transaction timeout configuration

Usage Example

```
import { executeTransaction } from '@/lib/db/transactions';
import { generatePresignedUrl, uploadToS3 } from '@/lib/s3/upload';

// Database transaction
const result = await executeTransaction(async (tx) => {
  const user = await tx.user.create({ data: {...} });
  const profile = await tx.family.create({ data: {...} });
  return { user, profile };
});

// S3 upload
const urlResult = await generatePresignedUrl(key, contentType);
if (urlResult.success) {
  const uploadResult = await uploadToS3(urlResult.url, file, contentType);
}
```

✅ RBAC (Role-Based Access Control)

Server-Side RBAC

- [x] Session authentication helper
- [x] Role checking utilities
- [x] Resource ownership validation
- [x] Permission definitions for all resources
- [x] RBAC middleware wrapper (`withAuth`)
- [x] Granular permission checks

Available Functions

- `requireAuth()` - Ensure authenticated
- `requireRole(role)` - Require specific role
- `requireAnyRole(roles)` - Require one of multiple roles
- `requireAdmin()` - Admin only
- `requireOperator()` - Operator only
- `requireStaff()` - Admin or Operator
- `requireOwnership()` - Owner or Admin
- `requireOwnershipOrStaff()` - Owner, Admin, or Operator

Client-Side RBAC

- [x] React hooks for role checking
- [x] Role-based component rendering
- [x] Conditional UI elements

Available Hooks

- `useUserRole()` - Get current user's role
- `useHasRole(role)` - Check specific role
- `useHasAnyRole(roles)` - Check multiple roles
- `useIsAdmin()` - Is admin?

- `useIsOperator()` - Is operator?
- `useIsStaff()` - Is staff (admin/operator)?
- `useIsFamily()` - Is family?
- `useIsAide()` - Is aide/caregiver?
- `useIsProvider()` - Is provider?

Usage Example

```
// Backend
import { requireRole, requireOwnership } from '@/lib/auth/rbac';

const session = await requireRole(UserRole.FAMILY);
requireOwnership(session, resourceUserId);

// Frontend
import { useIsAdmin, RoleGate, AdminOnly } from '@/lib/auth/client-rbac';

const isAdmin = useIsAdmin();

<RoleGate roles={[UserRole.ADMIN, UserRole.OPERATOR]}>
  <StaffOnlyContent />
</RoleGate>

<AdminOnly>
  <AdminPanel />
</AdminOnly>
```

✅ Logging & Monitoring

Structured Logging

- [x] JSON-formatted logs
- [x] Log levels (ERROR, WARN, INFO, DEBUG)
- [x] Context-aware logging (userId, role, path, etc.)
- [x] Sensitive data sanitization
- [x] Audit logging for critical events
- [x] Environment-based log levels

Logged Events

- Authentication attempts (success/failure)
- User registration
- Profile updates
- Lead submissions
- Credential uploads
- Resource access (ownership checks)
- API errors
- S3 upload failures
- Database transaction failures

Usage Example

```
import { logger } from '@/lib/logger';

logger.info('User registered', {
  userId: user.id,
  role: user.role,
  email: user.email,
});

logger.error('API error', {
  userId,
  path: req.url,
  error: error.message,
});

logger.audit('Lead created', {
  userId: session.user.id,
  leadId: lead.id,
  targetType: lead.targetType,
});
```

✅ API Route Audit Status

Fully Hardened (Example)

- [x] `/api/example-hardened/*` - Reference implementation

Requires Hardening

The following routes should be updated to use the new utilities:

Authentication

- [ ] `/api/auth/register`
- [ ] `/api/auth/[...nextauth]`

Profile Management

- [ ] `/api/profile` (GET, PATCH)
- [ ] `/api/profile/photo`
- [ ] `/api/family/profile`
- [ ] `/api/aide/profile`
- [ ] `/api/provider/profile`

Credentials

- [ ] `/api/caregiver/credentials`
- [ ] `/api/caregiver/credentials/[id]`
- [ ] `/api/caregiver/credentials/upload-url`
- [ ] `/api/provider/credentials`
- [ ] `/api/provider/credentials/[id]`
- [ ] `/api/admin/provider-credentials/[id]`

Leads

- [ ] `/api/leads` (POST)

- [ ] `/api/operator/leads/*`

Messages

- [ ] `/api/messages`
- [ ] `/api/messages/*`

Marketplace

- [ ] `/api/marketplace/caregivers`
- [ ] `/api/marketplace/providers`
- [ ] `/api/marketplace/*/[id]`

Favorites

- [ ] `/api/favorites`
- [ ] `/api/favorites/all`

Admin

- [ ] `/api/admin/*`

✅ UI Pages Audit Status

Requires RBAC Guards

- [ ] `/settings/*` - Add role-based access
- [ ] `/operator/*` - Require operator/admin
- [ ] `/admin/*` - Require admin
- [ ] `/marketplace/*` - Public with auth for actions
- [ ] `/messages` - Require auth
- [ ] `/favorites` - Require auth

🔄 Migration Path

To harden an existing API route:

1. **Import utilities**
   `typescript`
   `import { handleApiError, createErrorResponse, ErrorCode } from '@/lib/errors/api-errors';`
   `import { requireAuth, requireRole } from '@/lib/auth/rbac';`
   `import { logger } from '@/lib/logger';`
   `import { sanitizeObject } from '@/lib/validation/sanitize';`
   `import { yourSchema } from '@/lib/validation/schemas';`

2. **Wrap handler in try-catch**
   `typescript`
   `try {`
   `// ... your code`
   `} catch (error) {`

```
    return handleApiError(error, { path, method });
  }
```

3. **Add authentication**

```
typescript
  const session = await requireAuth();
  // or
  const session = await requireRole(UserRole.FAMILY);
```

4. **Add validation**

```
typescript
  const result = yourSchema.safeParse(body);
  if (!result.success) {
    return createErrorResponse(
      ErrorCode.VALIDATION_ERROR,
      'Invalid input',
      400,
      { errors: result.error.format() }
    );
  }
```

5. **Sanitize input**

```
typescript
  const sanitized = sanitizeObject(result.data, {
    richTextFields: ['bio'],
    urlFields: ['website'],
  });
```

6. **Check ownership**

```
typescript
  requireOwnership(session, resource.userId);
```

7. **Add logging**

```
typescript
  logger.audit('Action performed', {
    userId: session.user.id,
    resourceId,
  });
```

📊 Testing Checklist

Error Handling

- [ ] Test invalid input (malformed JSON, wrong types)
- [ ] Test missing required fields
- [ ] Test validation errors display properly
- [ ] Test unauthorized access returns 401
- [ ] Test forbidden access returns 403
- [ ] Test not found returns 404
- [ ] Test duplicate resource returns 409

- [ ] Test toast notifications appear

RBAC

- [ ] Test role-based route access
- [ ] Test ownership validation
- [ ] Test admin-only features
- [ ] Test staff-only features
- [ ] Test cross-role access attempts fail
- [ ] Test UI elements hide/show based on role

File Uploads

- [ ] Test file type validation (try .exe, .js)
- [ ] Test file size limits (upload 20MB file)
- [ ] Test S3 upload retries (simulate failure)
- [ ] Test filename sanitization (try ../../../etc/passwd)

Input Sanitization

- [ ] Test XSS prevention (submit `<script>alert('xss')</script>`)
- [ ] Test HTML injection
- [ ] Test SQL injection (though Prisma protects)
- [ ] Test javascript: URLs

Database

- [ ] Test transaction rollback on error
- [ ] Test concurrent updates
- [ ] Test connection failures

🚀 Deployment Considerations

- [ ] Ensure environment variables are set (AWS_*, DATABASE_URL, etc.)
- [ ] Review and update rate limiting configuration
- [ ] Set up log aggregation (CloudWatch, Datadog, etc.)
- [ ] Configure error monitoring (Sentry, Rollbar, etc.)
- [ ] Set up uptime monitoring
- [ ] Configure CORS properly
- [ ] Review CSP headers
- [ ] Enable HTTPS only
- [ ] Set secure cookie flags
- [ ] Configure session timeout

📚 Additional Resources

- API Error Codes (./API_ERROR_CODES.md)
- RBAC Matrix (./RBAC_MATRIX.md)

- Example Hardened Route (/src/app/api/example-hardened/route.ts)