Cursor Protection Guide - Rocky Web Studio

Created: January 26, 2025
Purpose: Protect working features from accidental breakage
Status: Active Protection Rules

🔴 CRITICAL: Files You MUST NOT BREAK

These files are **LIVE IN PRODUCTION** and breaking them = broken website.

1. `app/api/ai-assistant/route.ts`

Status: ✅ Working perfectly
Why Critical: This is the API endpoint for the chat widget. If this breaks, the chat widget stops working.

What NOT to Do:

- ❌ Don't change the model name (`claude-3-haiku-20240307`)
- ❌ Don't remove error handling
- ❌ Don't remove Supabase integration
- ❌ Don't remove rate limiting
- ❌ Don't remove streaming response logic

What You CAN Do:

- ✅ Update comments (e.g., "Claude 3.5 Sonnet" → "Claude 3 Haiku")
- ✅ Add logging (non-breaking additions)
- ✅ Refactor variable names (if you test thoroughly)

Before Modifying:

1. Create backup branch: `git checkout -b backup-ai-assistant-route`
2. Test locally: `npm run dev` → visit `/test-ai`
3. Send a test message in chat widget
4. Verify response comes back correctly
5. Check browser console for errors (F12)

If You Break It:

```
# Revert to last working version
git checkout app/api/ai-assistant/route.ts

# Or restore from backup branch
git checkout backup-ai-assistant-route -- app/api/ai-assistant/route.ts
```

---

2. `components/AIAssistantWidget.tsx`

Status: ✅ Live on all pages
Why Critical: This is the chat widget users see. If this breaks, the widget disappears or stops working.

What NOT to Do:

- ❌ Don't remove the widget rendering logic
- ❌ Don't change the API endpoint (`/api/ai-assistant`)
- ❌ Don't break keyboard navigation (WCAG compliance)
- ❌ Don't remove error handling
- ❌ Don't change the widget's position (fixed bottom-right)
- ❌ Don't remove accessibility attributes (aria-labels, roles)

What You CAN Do:

- ✅ Add lead capture functionality (Week 6.3)
- ✅ Improve styling (keep position same)
- ✅ Add features (non-breaking additions)

Before Modifying:

1. Create backup branch
2. Test locally: `npm run dev`
3. Visit any page (homepage works)
4. Click chat widget button
5. Send test message
6. Verify widget opens/closes correctly
7. Test keyboard navigation (Tab, Enter, Escape)
8. Check accessibility: F12 → Lighthouse → Accessibility

If You Break It:

```
# Revert to last working version
git checkout components/AIAssistantWidget.tsx

# Verify widget appears again
npm run dev
# Visit homepage, check bottom-right corner
```

---

3. `lib/claude.ts`

Status: ✅ Claude API integration working
Why Critical: This handles all Claude API calls. If this breaks, no AI responses work.

What NOT to Do:

- ❌ Don't change the model (`claude-3-haiku-20240307`)
- ❌ Don't remove error handling (especially 402, 403, 429 codes)
- ❌ Don't change the streaming logic
- ❌ Don't remove message validation

What You CAN Do:

- ✅ Update error messages (user-friendly improvements)
- ✅ Add logging
- ✅ Refactor internal functions (if tested)

Before Modifying:

1. Create backup branch
2. Test: `npm run dev` → `/test-ai`
3. Send message in chat widget
4. Verify Claude responds correctly
5. Test error scenarios (disconnect network, see if error message appears)

If You Break It:

```
git checkout lib/claude.ts
npm run dev
# Test chat widget again
```

---

4. `app/layout.tsx`

Status: ✅ Widget integrated, WCAG compliant
Why Critical: This imports the chat widget. If you remove the import, widget disappears. Also affects Lighthouse score.

What NOT to Do:

- ❌ Don't remove `<AIAssistantWidget />`
- ❌ Don't change the import statement
- ❌ Don't modify metadata (SEO impact)
- ❌ Don't break WCAG compliance (semantic HTML, lang attribute)

What You CAN Do:

- ✅ Add new components (non-breaking)
- ✅ Update metadata (SEO improvements)
- ✅ Modify styling (if you test Lighthouse)

Before Modifying:

1. Create backup branch
2. Test: `npm run dev`
3. Visit homepage
4. Verify chat widget appears (bottom-right)
5. Run Lighthouse: F12 → Lighthouse → Generate Report
6. Check Accessibility score (should be >90)
7. Check Performance score (should be >91)

If You Break It:

```
git checkout app/layout.tsx
npm run dev
# Verify widget appears again
# Run Lighthouse to verify score
```

---

5. Database Migrations

Location: `supabase/migrations/*.sql`
Status: ✅ Applied to production database
Why Critical: Never revert migrations. They've already run on production.

What NOT to Do:

- ❌ Don't delete migration files
- ❌ Don't modify existing migrations (they've already run)
- ❌ Don't create conflicting migrations
- ❌ Don't rename existing migration files

What You CAN Do:

- ✅ Create new migration files (with new timestamps)
- ✅ Add new tables/columns
- ✅ Create indexes (non-breaking additions)

Before Creating New Migration:

1. Check existing migrations: `ls supabase/migrations/`
2. Use timestamp format: `YYYYMMDD_description.sql`
3. Test locally first (if you have local Supabase)
4. Always use `IF NOT EXISTS` for safety

5. Use transactions: `BEGIN; ... COMMIT;`

Example Safe Migration:

```sql
-- supabase/migrations/20250127_create_case_studies_table.sql
BEGIN;

CREATE TABLE IF NOT EXISTS case_studies (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  -- ... columns
);

CREATE INDEX IF NOT EXISTS idx_case_studies_slug ON case_studies(slug);

COMMIT;
```

If You Need to Fix a Migration:

- Don't modify the old migration file
- Create a NEW migration file that fixes it
- Example: `20250127_fix_case_studies_column.sql`

---

6. Environment Variables

Location: `.env.local` (local), Vercel dashboard (production)
Status: ✅ Configured and working
Why Critical: These contain secrets and configuration. Breaking them = broken features.

What NOT to Do:

- ❌ Don't commit `.env.local` to git (should be in `.gitignore`)
- ❌ Don't share API keys publicly
- ❌ Don't remove required variables
- ❌ Don't change variable names (code expects these names)

Required Variables:

- `ANTHROPIC_API_KEY` - Claude API (chat widget needs this)
- `NEXT_PUBLIC_SUPABASE_URL` - Supabase connection
- `NEXT_PUBLIC_SUPABASE_ANON_KEY` - Supabase public key
- `SUPABASE_SERVICE_ROLE_KEY` - Supabase admin key (server-side only)

If Environment Variables Break:

1. Check `.env.local` exists locally
2. Check Vercel dashboard for production variables
3. Verify variable names match code expectations
4. Restart dev server: `npm run dev`

🟡 CAUTION: Files to Modify Carefully

These files can be modified, but test thoroughly.

`types/supabase.ts`

Why Caution: TypeScript types must match database schema. If types are wrong, TypeScript errors occur.

Before Modifying:

1. Check if you need to regenerate types instead of editing manually
2. Test: `npm run build` (should pass)
3. Verify no TypeScript errors in files using these types

How to Regenerate Types:

```
# If you have Supabase CLI configured
supabase gen types typescript --project-id YOUR_PROJECT_ID > types/supabase.ts
```

`middleware.ts`

Why Caution: Protects routes. If you break it, authentication might fail.

Current Status:

- ✅ Protects `/admin/*` routes (checks if logged in)
- ⚠️ Doesn't check admin role (any logged-in user can access)

Before Modifying:

1. Test admin routes still work
2. Test non-authenticated users get redirected to `/login`
3. Test authenticated users can access admin routes

Recommended Enhancement (Week 4): Add role-based access control:

```
// Check if user is admin
if (isAdmin && session?.user?.role !== 'admin') {
  return NextResponse.json({ error: 'Unauthorized' }, { status: 403 });
}
```

Any Component Using Chat Widget

Files: Any page/component that might conflict with the widget
Why Caution: Z-index conflicts, positioning issues, CSS conflicts

Before Modifying:

1. Check if component has `z-50` or higher (widget uses `z-50`)
2. Test: Does widget still appear and work?
3. Test: Can you click widget button? (not blocked by other elements)

✅ SAFE: Files You Can Modify Freely

These are safe to modify (they don't affect critical features).

- `app/page.tsx` (homepage - but test after changes)
- New files you're creating
- `components/` (except `AIAssistantWidget.tsx`)
- `lib/` (except `claude.ts`)
- `docs/` (documentation files)
- `case-studies/` (new case study content)

Still Test After Changes:

- Run `npm run build`
- Test locally: `npm run dev`

- Check for TypeScript errors
- Check browser console (F12) for runtime errors

💉 Testing Checklist (Before Every Commit)

Use this checklist before committing ANY changes.

1. TypeScript Check

```
npm run build
```

Expected: No errors
If errors: Fix TypeScript errors before committing

2. Local Development Test

```
npm run dev
```

Expected: Server starts without errors
If errors: Fix startup errors before committing

3. Feature Test

- Visit the page/feature you modified
- Test the functionality you changed
- Verify it works as expected

4. Widget Test (if you modified anything global)

- Visit homepage
- Click chat widget button
- Send a test message
- Verify response comes back
- Verify widget can close/open

5. Console Check

- Open browser: F12 → Console
- Look for red errors
- **Expected:** No errors (warnings are usually okay)

6. Lighthouse Check (if you modified layout/styles)

- F12 → Lighthouse
- Run audit
- **Expected:** Accessibility >90, Performance >91
- **If dropped:** Investigate what changed

7. Accessibility Check (if you modified UI)

- Tab through page (keyboard navigation)
- Check focus indicators visible
- Verify screen reader labels present
- **Expected:** All interactive elements keyboard accessible

8. Git Status Check

```
git status
```

Review: What files are changing?
If protected files: Double-check you meant to modify them

9. Commit Message

Write clear commit message:

```
git commit -m "feat(case-studies): add database schema migration"
```

Format: `type(scope): description`
Types: `feat` , `fix` , `docs` , `style` , `refactor` , `test` , `a11y`

🚨 Emergency Recovery

Chat Widget Disappeared

Symptom: No chat button on pages
Likely Cause: Removed `<AIAssistantWidget />` from `app/layout.tsx`

Fix:

```
# Check if import is missing
grep -n "AIAssistantWidget" app/layout.tsx

# If missing, restore from git
git checkout app/layout.tsx
```

Verify:

```
npm run dev
# Visit homepage, check bottom-right corner
```

Chat Widget Not Responding

Symptom: Widget opens but messages don't send/get response
Likely Cause: API route broken or environment variable missing

Fix:

```
# Check API route exists
ls app/api/ai-assistant/route.ts

# Check environment variable
echo $ANTHROPIC_API_KEY  # Should show key (locally)
# Or check Vercel dashboard for production

# Revert API route if needed
git checkout app/api/ai-assistant/route.ts
```

Verify:

```
npm run dev
# Visit /test-ai page
# Send message, check response
```

TypeScript Build Errors

Symptom: `npm run build` fails with type errors
Likely Cause: Removed import, changed type, or schema mismatch

Fix:

1. Read error message carefully
2. Check if you removed an import
3. Check if `types/supabase.ts` is outdated
4. If schema changed, regenerate types:

```
supabase gen types typescript --project-id YOUR_PROJECT_ID > types/supabase.ts
```

Verify:

```
npm run build
# Should pass without errors
```

Lighthouse Score Dropped

Symptom: Accessibility or Performance score below target
Likely Cause: Changed colors (contrast), added heavy images, or removed semantic HTML

Fix:

1. Run Lighthouse: F12 → Lighthouse → Generate Report
2. Check which metric dropped
3. Review "Opportunities" section for fixes
4. If contrast issue: Check color contrast (should be 4.5:1 minimum)
5. If performance: Optimize images, remove unused code

Common Fixes:

- Color contrast: Use darker/lighter colors
- Performance: Use Next.js Image component, lazy load images
- Accessibility: Add semantic HTML, aria-labels, alt text

Database Migration Failed

Symptom: Migration doesn't apply or breaks database
Likely Cause: Syntax error, conflict with existing schema, or migration already ran

Fix:

1. Check migration syntax (run locally if possible)
2. Check if migration already ran (check Supabase dashboard)
3. If migration partially applied, create new migration to fix it
4. **Never:** Delete or modify existing migration files

Prevention:

- Always use `IF NOT EXISTS` for safety
- Always use transactions (`BEGIN; ... COMMIT;`)
- Test migrations locally first (if possible)

📋 Quick Command Reference

Safe Commands (Information Only)

```
# Check what files changed
git status

# See what changed in a file
git diff filename

# Check if file exists
ls filename

# Check TypeScript types
npm run build
```

Recovery Commands (Use Carefully)

```
# Revert one file to last committed version
git checkout filename

# Revert all changes (destructive!)
git reset --hard HEAD

# Undo last commit (keeps changes)
git reset HEAD~1

# Create backup branch
git checkout -b backup-branch-name

# Switch back to main branch
git checkout main
```

🎯 Best Practices

Before Modifying Protected Files

1. ✅ Read this guide (you're doing this now!)
2. ✅ Create backup branch
3. ✅ Understand what the file does
4. ✅ Make smallest change possible
5. ✅ Test immediately
6. ✅ Only then continue

When Creating New Features

1. ✅ Create new files (safer than modifying existing)
2. ✅ Follow existing patterns in codebase
3. ✅ Use TypeScript types
4. ✅ Test thoroughly

5. ✅ Follow accessibility guidelines (WCAG 2.1 AA)

Before Every Commit

1. ✅ Run testing checklist (above)
2. ✅ Write clear commit message
3. ✅ Review what's changing (`git diff`)
4. ✅ Verify no protected files broken
5. ✅ Then commit and push

📞 Quick Help Decision Tree

"I'm about to modify file X, is it safe?"

- → Check this guide → Search for filename
- → If marked 🔴 = VERY CAREFUL, read section first
- → If marked 🟡 = Test thoroughly after changes
- → If not listed = Probably safe, but still test

"I got an error, what do I do?"

- → Check "Emergency Recovery" section above
- → Match error to symptom
- → Follow fix instructions
- → Verify fix worked

"Something broke after my changes!"

- → Check git status: `git status`
- → Check what changed: `git diff`
- → Revert if needed: `git checkout filename`
- → Test: `npm run dev`
- → If still broken, check recovery section

✅ Status Checklist

Before starting work each day:

- ☐ Read this guide (refresher)
- ☐ Check which files you'll modify today
- ☐ Create backup branch if modifying protected files
- ☐ Have testing checklist ready

Before committing:

- ☐ Ran all tests in checklist
- ☐ No TypeScript errors
- ☐ No console errors
- ☐ Widget still works (if modified anything global)
- ☐ Lighthouse score maintained (if modified layout/styles)
- ☐ Clear commit message written

Status: Active Protection Guide
Last Updated: January 26, 2025
Next Review: After Week 2 completion

Remember: When in doubt, test first, commit second. When really in doubt, create a backup branch first! 🛡️