# Coding Best Practices for the Project:

When completing the refactor, ensure you are complying with each of the following best practices:

#### 1. Type Definitions:

- Let Wasp handle its own types: Instead of writing `type WaspContext = {...}`, let Wasp's type system work by using its built-in types.
  - Example: Use `context: any` in operations rather than defining custom context types.
- Define custom types only for our interfaces: Create interfaces for our specific needs like `SaveOnboardingInput`, but don't try to override Wasp's internal types.
  - Example: `export interface SaveOnboardingInput { webAppUrl?: string; ... }`
- Keep types aligned with schema: Every type should reflect exactly what's in schema.prisma.
  - Example: If schema has `sredClaimName: String`, our type should be `sredClaimName: string`, not `name: string`.

## 2. Database Operations:

- Use schema as source of truth: Always check schema.prisma before creating or updating database operations.
  - Example: Using `documentStatusExperiment: 'pending'` because that's the exact field name in schema, not `status` or `experimentStatus`.
- Verify fields exist: Don't assume fields exist just because they make logical sense.
  - Example: We removed `lastUpdated` because it wasn't in the schema, even though it seemed logical to have.
- Respect Prisma's automatic fields: Some fields like `updatedAt` are handled by Prisma automatically.
  - Example: Don't try to set `updatedAt` manually in updates or creates.

## 3. Context Handling:

- Use Wasp's context: Let Wasp manage how context is structured and accessed.
  - Example: Access database through `context.entities.SREDClaim` rather than creating custom access patterns.
- Don't redefine delegates: Accept Wasp's delegate types rather than creating our own.

- Example: Don't define `type WaspContext = { entities: { ... } }`.
- Handle optional properties: Be explicit about which properties might be undefined.
  - Example: Always check `if (!context.user)` before accessing user properties.

#### 4. Error Prevention:

- Check schema first: Before writing any database operations, verify field names and types in schema.
  - Example: Check if a field is `String?` (optional) or `String` (required) in schema before setting defaults.
- Match schema types: Provide values that match the schema's type definitions.
  - Example: Use `[]` for fields defined as arrays like `additionalTechnologies String[]`.
- Handle null cases: Explicitly handle cases where values might be null.
  - Example: `data.companyAnalysis?.businessProblem ?? "` to handle optional values.

## 5. Code Organization:

- Separate types from logic: Keep type definitions at the top of the file, separate from implementation.
  - Example: Put all interfaces at the top of the file, before any function implementations.
- Use interfaces for inputs: Create clear interfaces for complex input objects.
  - Example: `SaveOnboardingInput` interface for the saveOnboardingData function's arguments.
- Document dependencies: Make it clear what types depend on what other types.
  - Example: Import statements showing where types come from: `import { SREDClaim } from '@prisma/client'`.

## 6. Testing:

- Incremental testing: Make small changes and test each change before moving on.
  - Example: Fix one type error at a time rather than trying to fix everything at once.
  - Type checking first: Use TypeScript errors as the first line of defense.
    - Example: Fix all TypeScript errors before testing runtime behavior.
- Edge case testing: Test with optional fields both present and absent.
  - Example: Test with both complete and partial `companyAnalysis` objects.