Contents

Contributing to RADIANT

Thank you for your interest in contributing to RADIANT! This document provides guidelines and instructions for contributing.

Code of Conduct

Please read and follow our Code of Conduct.

Getting Started

Prerequisites

- Node.js 20.0+
- pnpm 8.0+
- Docker (for local development)
- AWS CLI (for deployment)

Setup

1. Fork the repository
2. Clone your fork:

   ```
   git clone https://github.com/YOUR_USERNAME/radiant.git
   cd radiant
   ```

3. Run the setup script:

   ```
   ./scripts/setup-dev.sh
   ```

4. Start local services:

   ```
   docker-compose up -d
   ```

5. Run migrations:

   ```
   ./scripts/run-migrations.sh --seed
   ```

Development Workflow

Branch Naming

- `feature/` - New features
- `fix/` - Bug fixes
- `docs/` - Documentation changes
- `refactor/` - Code refactoring
- `test/` - Test additions/changes

Example: `feature/add-model-analytics`

Commit Messages

Follow Conventional Commits:

```
type(scope): description

[optional body]

[optional footer]
```

Types: - `feat` - New feature - `fix` - Bug fix - `docs` - Documentation - `style` - Formatting - `refactor` - Code restructuring - `test` - Tests - `chore` - Maintenance

Example:

```
feat(billing): add volume discount calculation

Implements tiered volume discounts for credit purchases.
Discounts range from 5% to 20% based on purchase amount.

Closes #123
```

Pull Requests

1. Create a feature branch from `develop`
2. Make your changes
3. Run tests: `pnpm test`
4. Run linting: `pnpm lint`
5. Push your branch
6. Open a PR against `develop`

PR titles should follow commit message format.

Code Style

TypeScript

- Use strict TypeScript (`strict: true`)
- Prefer `interface` over `type` for objects
- Use explicit return types for functions
- Avoid `any` - use `unknown` if type is truly unknown

React/Next.js

- Use functional components with hooks
- Use `'use client'` directive only when necessary
- Prefer server components when possible
- Use Tailwind CSS for styling

Database

- Use parameterized queries (never string interpolation)
- Always use RLS with `app.current_tenant_id`
- Name migrations: `NNN_description.sql`
- Include rollback considerations in comments

Lambda Handlers

- Use the shared `handleError` utility
- Extract auth context with `extractAuthContext`
- Return proper HTTP status codes
- Log errors but don't expose internals
- Use standardized error codes from `@radiant/shared/errors`

```
import { ErrorCodes, RadiantError } from '@radiant/shared';

// Throw standardized errors
throw new RadiantError(ErrorCodes.RESOURCE_NOT_FOUND, 'User not found');
```

Testing

Unit Tests

```
# Run all tests
pnpm test

# Run with coverage
pnpm test:coverage

# Run specific test file
pnpm test -- services.test.ts
```

Lambda Handler Tests

Tests are located in `__tests__/` directories within each handler:

```
cd packages/infrastructure

# Run all Lambda tests
pnpm test

# Run specific handler tests
pnpm test -- admin
pnpm test -- billing
```

E2E Tests

```
cd apps/admin-dashboard

# Run Playwright tests
pnpm test:e2e

# Run with UI
pnpm test:e2e:ui
```

Swift Tests

```
cd apps/swift-deployer

# Run all Swift tests
swift test

# Run specific test class
swift test --filter LocalStorageManagerTests
```

Test Guidelines

- Test business logic, not implementation details
- Use descriptive test names
- Mock external dependencies

- Aim for >80% coverage on new code
- Use the test utilities in `@radiant/shared/testing`

Database Migrations

Creating Migrations

1. Create a new file in `packages/infrastructure/migrations/`
2. Name it with the next sequential number: `037_your_migration.sql`
3. Include:
   - Table creation with proper types
   - RLS policies
   - Indexes for common queries
   - Rollback comments

Migration Requirements

- Must be idempotent (use `IF NOT EXISTS`)
- Must include RLS policies for tenant isolation
- Must be tested locally before PR

Error Handling

Use standardized error codes from `packages/shared/src/errors/`:

```
import { ErrorCodes, RadiantError, createNotFoundError } from '@radiant/shared';

// Using factory functions
throw createNotFoundError('Tenant', tenantId);

// Direct construction
throw new RadiantError(ErrorCodes.VALIDATION_REQUIRED_FIELD, 'Email is required', {
  details: { field: 'email' },
});
```

See Error Codes Reference for the full list.

Documentation

- Update README.md for user-facing changes
- Update OpenAPI spec for API changes
- Add JSDoc comments for public functions
- Update CHANGELOG.md
- Update docs/ERROR_CODES.md for new error codes
- Update docs/TESTING.md for new test patterns

Releasing

Releases are managed by maintainers:

1. Merge PRs to `develop`

2. Create release PR from `develop` to `main`
3. Update CHANGELOG.md
4. Merge and tag release
5. CI/CD handles deployment

Questions?

- Open a Discussion
- Check existing Issues

License

By contributing, you agree that your contributions will be licensed under the MIT License.