Software Development Technical Manual

Acrux MVP - Research Data Management Platform

Version: 1.0
Last Updated: October 3, 2025
Document Owner: Engineering Team
Audience: Developers, DevOps, QA Engineers

Table of Contents

Introduction

Purpose

This manual provides comprehensive guidelines for developers working on the Acrux MVP platform. It covers setup procedures, coding standards, development workflows, and best practices to ensure consistent, high-quality code across the team.

Prerequisites

Required Knowledge:
- JavaScript/TypeScript fundamentals
- React and Next.js concepts
- REST APIs and databases
- Git version control

Required Software:
- Node.js v20.x or higher
- Yarn package manager
- PostgreSQL 15.x

- Git 2.x
- VS Code (recommended)

Development Environment Setup

Quick Start

```
# 1. Clone repository
git clone https://github.com/your-org/acrux_mvp.git
cd acrux_mvp/app

# 2. Install dependencies
yarn install

# 3. Configure environment
cp .env.example .env.local
# Edit .env.local with your credentials

# 4. Setup database
yarn prisma migrate dev
yarn prisma generate

# 5. Start development server
yarn dev
```

Visit http://localhost:3000

Environment Configuration

Create `.env.local` file:

```
# Database
DATABASE_URL="postgresql://user:password@localhost:5432/acrux_dev"

# Authentication
NEXTAUTH_URL="http://localhost:3000"
NEXTAUTH_SECRET="your-secret-key"  # Generate: openssl rand -base64 32

# AWS S3
AWS_BUCKET_NAME="acrux-dev-storage"
AWS_REGION="us-east-1"
AWS_ACCESS_KEY_ID="your-access-key"
AWS_SECRET_ACCESS_KEY="your-secret-key"
AWS_FOLDER_PREFIX="dev/"

# Application
NEXT_PUBLIC_APP_URL="http://localhost:3000"
```

VS Code Setup

Recommended extensions:
- ESLint
- Prettier
- Tailwind CSS IntelliSense

- Prisma
- TypeScript

Project Structure

```
acrux_mvp/
├── app/                          # Next.js application
│   ├── app/                      # App router (pages & APIs)
│   │   ├── (auth)/              # Authentication routes
│   │   ├── (dashboard)/         # Dashboard routes
│   │   ├── projects/            # Project pages
│   │   ├── api/                 # API endpoints
│   │   ├── layout.tsx           # Root layout
│   │   └── page.tsx             # Home page
│   │
│   ├── components/              # React components
│   │   ├── ui/                 # shadcn/ui components
│   │   ├── layout/             # Layout components
│   │   ├── features/           # Feature components
│   │   └── shared/             # Shared components
│   │
│   ├── lib/                     # Utilities and helpers
│   │   ├── db.ts               # Prisma client
│   │   ├── auth.ts             # Auth helpers
│   │   ├── s3.ts               # File storage
│   │   └── utils.ts            # Utility functions
│   │
│   ├── prisma/                  # Database
│   │   ├── schema.prisma       # Schema definition
│   │   └── migrations/         # Migration files
│   │
│   ├── hooks/                   # Custom React hooks
│   ├── public/                  # Static assets
│   └── __tests__/              # Test files
│
├── docs/                         # Documentation
│   ├── README.md
│   ├── PRD.md
│   ├── TSD.md
│   └── Technical_Manual.md (this file)
│
└── README.md                     # Project overview
```

Coding Standards

TypeScript Guidelines

Always Define Types

```
// ✅ Good
interface User {
  id: string
  email: string
  name: string | null
}

function getUser(id: string): Promise<User | null> {
  return prisma.user.findUnique({ where: { id } })
}

// ❌ Bad
function getUser(id: any): any {
  return prisma.user.findUnique({ where: { id } })
}
```

Use Optional Chaining

```
// ✅ Good
const userName = user?.name ?? 'Anonymous'
const projectCount = data?.projects?.length ?? 0

// ❌ Bad
const userName = user.name || 'Anonymous'
```

React Component Structure

```
'use client' // Only if client component needed

import { useState } from 'react'
import { Button } from '@/components/ui/button'

// 1. Props interface
interface ProjectCardProps {
  project: Project
  onDelete?: (id: string) => void
}

// 2. Component
export function ProjectCard({ project, onDelete }: ProjectCardProps) {
  // 3. Hooks
  const [isDeleting, setIsDeleting] = useState(false)

  // 4. Event handlers
  const handleDelete = async () => {
    setIsDeleting(true)
    await onDelete?.(project.id)
    setIsDeleting(false)
  }

  // 5. Render
  return (
    <div>
      <h3>{project.name}</h3>
      <Button onClick={handleDelete} disabled={isDeleting}>
        Delete
      </Button>
    </div>
  )
}
```

Server vs Client Components

```
// SERVER COMPONENT (default)
// No 'use client' directive
export default async function ProjectsPage() {
  const projects = await prisma.project.findMany()
  return <ProjectList projects={projects} />
}

// CLIENT COMPONENT
// Requires 'use client' for interactivity
'use client'

export function ProjectForm() {
  const [name, setName] = useState('')
  return <input value={name} onChange={(e) => setName(e.target.value)} />
}
```

API Route Pattern

```
import { NextRequest, NextResponse } from 'next/server'
import { requireAuth } from '@/lib/auth'
import { handleError } from '@/lib/error-handler'

export async function GET(req: NextRequest) {
  try {
    // 1. Authentication
    const user = await requireAuth()

    // 2. Validation
    const { searchParams } = new URL(req.url)
    const page = parseInt(searchParams.get('page') || '1')

    // 3. Business logic
    const data = await fetchData(user.id, page)

    // 4. Response
    return NextResponse.json({ success: true, data })
  } catch (error) {
    return handleError(error)
  }
}
```

Naming Conventions

Type	Convention	Example
Components	PascalCase	`UserProfile`, `DataTable`
Functions	camelCase	`getUserById`, `formatDate`
Hooks	camelCase + `use`	`useAuth`, `useProject`
Constants	UPPER_SNAKE_CASE	`MAX_FILE_SIZE`, `API_URL`
Files	kebab-case	`user-profile.tsx`

Development Workflow

Git Workflow

Branch Strategy

```
main                     # Production
  └─ develop            # Integration
      ├─ feature/xxx    # New features
      ├─ bugfix/xxx     # Bug fixes
      └─ hotfix/xxx     # Critical fixes
```

Commit Messages

Use conventional commits format:

```
<type>(<scope>): <description>

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

Examples:
feat(projects): add project creation form
fix(upload): resolve file size validation
docs(readme): update setup instructions
```

Pull Request Checklist

Before Creating PR:
- [ ] Tests pass: `yarn test`
- [ ] Linter passes: `yarn lint`
- [ ] Types check: `yarn tsc --noEmit`
- [ ] Manually tested
- [ ] Code follows style guide

PR Description:
- Describe changes
- Link related issues
- Add screenshots (for UI changes)
- Request reviewers

Testing Guidelines

Test Structure

```
__tests__/
├── unit/              # Individual functions/components
├── integration/       # API routes, database
└── e2e/              # Complete user flows
```

Unit Tests

```
// lib/__tests__/utils.test.ts
import { formatFileSize } from '../utils'

describe('formatFileSize', () => {
  it('formats correctly', () => {
    expect(formatFileSize(1024)).toBe('1 KB')
    expect(formatFileSize(1048576)).toBe('1 MB')
  })

  it('handles edge cases', () => {
    expect(formatFileSize(0)).toBe('0 Bytes')
  })
})
```

Integration Tests

```
// app/api/__tests__/projects.test.ts
describe('POST /api/projects', () => {
  it('creates project successfully', async () => {
    const response = await POST(mockRequest)
    expect(response.status).toBe(201)
  })
})
```

E2E Tests

```
// __tests__/e2e/project.spec.ts
import { test, expect } from '@playwright/test'

test('create project flow', async ({ page }) => {
  await page.goto('/login')
  await page.fill('[name="email"]', 'test@example.com')
  await page.click('button[type="submit"]')
  await page.click('text=New Project')
  await expect(page).toHaveURL(/\/projects\/new/)
})
```

Running Tests

```
yarn test              # All tests
yarn test:watch        # Watch mode
yarn test:coverage     # With coverage
yarn test:e2e          # E2E tests only
```

Common Tasks

Add New Page

```
// app/app/my-page/page.tsx
export default function MyPage() {
  return <div>My Page Content</div>
}

// Accessible at: http://localhost:3000/my-page
```

Add API Endpoint

```
// app/api/my-endpoint/route.ts
import { NextRequest, NextResponse } from 'next/server'

export async function GET(req: NextRequest) {
  return NextResponse.json({ message: 'Hello' })
}

export async function POST(req: NextRequest) {
  const body = await req.json()
  return NextResponse.json({ success: true })
}
```

Add Database Model

```
// prisma/schema.prisma
model NewModel {
  id        String   @id @default(cuid())
  name      String
  createdAt DateTime @default(now())
}
```

```
# Apply changes
yarn prisma migrate dev --name add_new_model
yarn prisma generate
```

Add UI Component

```
# Add from shadcn/ui library
npx shadcn-ui@latest add button
npx shadcn-ui@latest add dialog
npx shadcn-ui@latest add form
```

File Upload Implementation

```
// app/api/upload/route.ts
export async function POST(req: NextRequest) {
  const formData = await req.formData()
  const file = formData.get('file') as File

  const buffer = Buffer.from(await file.arrayBuffer())
  const key = await uploadToS3(buffer, file.name)

  return NextResponse.json({ fileKey: key })
}
```

Troubleshooting

Common Issues

Issue	Solution
Module not found	Check path, restart dev server
Prisma client error	Run `yarn prisma generate`
DB connection failed	Verify PostgreSQL is running, check DATABASE_URL
Hydration mismatch	Don't use `Date.now()` or `Math.random()` in render
TypeScript errors	Run `yarn tsc --noEmit`, fix type issues

Debugging Tools

Browser:
- Console: Check errors and warnings
- Network: Monitor API requests
- React DevTools: Inspect components

VS Code:
- Set breakpoints (F9)
- Start debugging (F5)
- Step through code (F10/F11)

Database:

```
yarn prisma studio  # Visual database browser
```

Quick Fixes

```
# Clear Next.js cache
rm -rf .next

# Reinstall dependencies
rm -rf node_modules && yarn install

# Reset database (dev only!)
yarn prisma migrate reset

# Regenerate Prisma client
yarn prisma generate
```

Best Practices

Performance

1. **Use Server Components** - Default to server, use client only when needed
2. **Optimize Images** - Use Next.js Image component
3. **Pagination** - Don't load all data at once
4. **Caching** - Cache expensive operations

```
// Optimize images
import Image from 'next/image'
<Image src="/img.jpg" alt="..." width={500} height={300} />

// Pagination
const items = await prisma.item.findMany({
  skip: (page - 1) * pageSize,
  take: pageSize
})
```

Security

1. **Validate Inputs** - Use Zod schemas
2. **Check Auth** - Protect all routes
3. **Never Expose Secrets** - Use environment variables

```
// Validate
import { z } from 'zod'
const schema = z.object({
  email: z.string().email(),
  password: z.string().min(8)
})

// Auth check
const user = await requireAuth()
```

Code Quality

1. **Self-Documenting Code** - Use clear, descriptive names
2. **Small Functions** - Single responsibility, < 50 lines

3. **TypeScript** - Define all types, avoid `any`

```
// ✅ Good
const MILLISECONDS_PER_DAY = 86400000
const tomorrow = Date.now() + MILLISECONDS_PER_DAY

// ❌ Bad
const x = Date.now() + 86400000
```

Resources

Official Documentation

- Next.js (https://nextjs.org/docs) - Framework documentation
- React (https://react.dev) - React fundamentals
- TypeScript (https://www.typescriptlang.org/docs) - Language reference
- Prisma (https://www.prisma.io/docs) - Database ORM
- Tailwind CSS (https://tailwindcss.com/docs) - Styling framework
- shadcn/ui (https://ui.shadcn.com) - Component library

Learning Resources

- Next.js Learn (https://nextjs.org/learn) - Interactive tutorial
- React Foundations (https://react.dev/learn) - Core concepts
- Prisma Quickstart (https://www.prisma.io/docs/getting-started) - Database setup

Team Communication

- **Questions**: Ask in team chat or GitHub discussions
- **Bugs**: Create GitHub issue with reproduction steps
- **Features**: Discuss in PRs or team meetings

Document History

Version	Date	Author	Changes
1.0	Oct 3, 2025	Engineering Team	Initial version

Document Status: ✅ Active
Next Review: January 2026
Maintained By: Acrux Engineering Team

For questions or suggestions, contact the engineering team or create an issue in the repository.