Application Modules Documentation

Overview

The Acrux MVP is structured into logical modules that handle different aspects of the objective management and team collaboration platform. This document outlines each module, its responsibilities, and key components.

Module Architecture

```
Acrux MVP
├── Authentication Module
├── Landing Page Module
├── Dashboard Module
├── Objectives Module
├── Pulse Check Module
├── Team Management Module
├── Profile Module
└── API Module
```

---

1. Authentication Module

Purpose

Handles user registration, login, session management, and access control.

Location

- `app/auth/`
- `lib/auth.ts`
- `components/auth/`

Key Components

Frontend Components

- **LoginForm** (`components/auth/login-form.tsx`)
- Email/password authentication
- Form validation
- Error handling
- Redirect after successful login
- **SignupForm** (`components/auth/signup-form.tsx`)
- User registration
- Field validation (name, email, password)
- Automatic role assignment

- Post-registration redirect

Pages

- **Login Page** (`app/auth/login/page.tsx`)
- Public access
- Clean, modern UI
- Link to signup page
- **Signup Page** (`app/auth/signup/page.tsx`)
- Public access
- Registration form
- Link to login page

Backend Services

- **Auth Configuration** (`lib/auth.ts`)
- NextAuth.js configuration
- Credentials provider setup
- Session management
- JWT token handling

API Routes

- `POST /api/auth/signup` - User registration
- `POST /api/auth/login` - User login (handled by NextAuth)
- `POST /api/auth/signout` - User logout

Features

- Email/password authentication
- Secure password hashing (bcrypt)
- Session persistence
- Protected route middleware
- Role-based access control (admin/team_member)

Security

- Password hashing with bcrypt
- JWT session tokens
- CSRF protection via NextAuth.js
- Secure HTTP-only cookies

---

2. Landing Page Module

Purpose

Marketing website that introduces Acrux to potential users and drives sign-ups.

Location

- `app/page.tsx`
- `components/landing/`

- `components/layout/`

Key Components

Sections

1. **HeroSection** (`components/landing/hero-section.tsx`)
   - Main value proposition
   - Primary CTA buttons
   - Eye-catching headline

2. **SocialProofSection** (`components/landing/social-proof-section.tsx`)
   - Trust indicators
   - User testimonials preview
   - Company logos or statistics

3. **ProblemSolutionSection** (`components/landing/problem-solution-section.tsx`)
   - Pain points identification
   - Solution presentation
   - Benefits overview

4. **FeaturesSection** (`components/landing/features-section.tsx`)
   - Core feature highlights
   - Icon-based feature cards
   - Feature descriptions

5. **TestimonialSection** (`components/landing/testimonial-section.tsx`)
   - User success stories
   - Client testimonials
   - Social proof elements

6. **CTASection** (`components/landing/cta-section.tsx`)
   - Final call-to-action
   - Sign-up encouragement
   - Multiple entry points

7. **Footer** (`components/layout/footer.tsx`)
   - Company information
   - Quick links
   - Legal pages

Features

- Responsive design
- Modern, professional UI
- Smooth scrolling navigation
- Clear conversion paths
- SEO-optimized content

---

3. Dashboard Module

Purpose

Central hub for users to view objectives, health scores, and quick actions.

Location

- `app/dashboard/page.tsx`
- `components/dashboard/`

Key Components

Pages

- **Dashboard Home** (`app/dashboard/page.tsx`)
- Server-side rendered
- Fetches user objectives
- Displays pulse request count
- Protected route (requires authentication)

Components

1. **DashboardNavbar** (`components/dashboard/dashboard-navbar.tsx`)
   - Navigation menu
   - User profile dropdown
   - Quick action buttons
   - Logout functionality

2. **DashboardContent** (`components/dashboard/dashboard-content.tsx`)
   - Main dashboard layout
   - Objective grid display
   - Statistics overview
   - Quick actions panel

3. **ObjectiveGrid** (`components/dashboard/objective-grid.tsx`)
   - Grid layout of objectives
   - Filter and sort controls
   - Empty state handling

4. **ObjectiveCard** (`components/dashboard/objective-card.tsx`)
   - Individual objective display
   - Progress indicator
   - Health score badge
   - Quick action buttons
   - Blocker count indicator

5. **HealthScore** (`components/dashboard/health-score.tsx`)
   - Visual health score display
   - Color-coded indicators
   - Tooltip explanations

Features

- Real-time objective overview
- Health score visualization

- Progress tracking
- Quick access to pulse requests
- Blocker alerts
- Role-based view (admin vs team member)

Data Fetching

- Server-side data fetching
- Optimized database queries with relations
- Automatic re-validation on navigation

---

4. Objectives Module

Purpose

Complete CRUD operations for objectives, including metrics, pulse checks, and blockers.

Location

- `app/dashboard/objectives/`
- `components/objectives/`

Key Components

Pages

1. **Objectives List** (`app/dashboard/objectives/page.tsx`)
   - All objectives view
   - Filter by status
   - Sort by date/health
   - Create new objective button

2. **Create Objective** (`app/dashboard/objectives/new/page.tsx`)
   - Objective creation form
   - Metric definition
   - Team assignment
   - Target date setting

3. **Objective Detail** (`app/dashboard/objectives/[id]/page.tsx`)
   - Detailed objective view
   - Progress visualization
   - Metrics display
   - Pulse check history
   - Blocker management
   - Team member list

4. **Edit Objective** (`app/dashboard/objectives/[id]/edit/page.tsx`)
   - Update objective details
   - Modify metrics
   - Change assignments
   - Update progress

Components

1. **ObjectivesPageClient** (`components/objectives/objectives-page-client.tsx`)
   - Client-side objective list
   - Search functionality
   - Filter controls
   - Status tabs

2. **ObjectiveForm** (`components/objectives/objective-form.tsx`)
   - Objective creation/edit form
   - Field validation
   - Metric management
   - Team assignment selector

3. **ObjectiveDetailClient** (`components/objectives/objective-detail-client.tsx`)
   - Detailed objective view
   - Tab navigation (overview, pulse, blockers, metrics)
   - Interactive charts
   - Action buttons

4. **ObjectivesList** (`components/objectives/objectives-list.tsx`)
   - Objective list renderer
   - Card/table view toggle
   - Empty state handling

5. **EditObjectiveForm** (`components/objectives/edit-objective-form.tsx`)
   - Objective update form
   - Pre-populated fields
   - Validation handling

Features

- Create, read, update, delete objectives
- Progress tracking (0-100%)
- Health score calculation
- Target date management
- Multiple metrics per objective
- Team member assignments
- Status management (active, completed, paused)
- Anonymous pulse checks
- Blocker tracking

Business Logic

- Auto-calculation of progress based on metrics
- Health score influenced by pulse checks
- Blocker severity affects health score
- Assignment notifications

---

5. Pulse Check Module

Purpose

Enables team members to provide sentiment and confidence ratings for objectives, helping admins gauge team morale and project health.

Location

- `components/modals/pulse-modal.tsx`
- `components/modals/simple-pulse-modal.tsx`
- `app/dashboard/pulse-requests/page.tsx`
- `components/team/pulse-requests-inbox.tsx`
- `app/api/objectives/[id]/pulse/route.ts`
- `app/api/pulse-requests/`

Key Components

Frontend Components

1. **PulseModal** (`components/modals/pulse-modal.tsx`)
   - Comprehensive pulse check form
   - Sentiment selector (1-5 emoji scale)
   - Confidence slider (1-5)
   - Optional feedback text area
   - Anonymous submission option

2. **SimplePulseModal** (`components/modals/simple-pulse-modal.tsx`)
   - Quick pulse check
   - Minimal UI for rapid feedback
   - Used in dashboard quick actions

3. **PulseRequestsInbox** (`components/team/pulse-requests-inbox.tsx`)
   - List of pending pulse requests
   - Request details display
   - Quick response action
   - Due date indicators
   - Status badges

Pages

- **Pulse Requests Page** (`app/dashboard/pulse-requests/page.tsx`)
- Team member view only
- Lists all pulse requests
- Pending vs completed tabs
- Direct response capability

API Routes

- `POST /api/objectives/[id]/pulse` - Submit pulse check
- `GET /api/pulse-requests` - Fetch user's pulse requests
- `POST /api/pulse-requests/[id]/respond` - Respond to pulse request
- `POST /api/pulse-requests` - Create pulse request (admin only)

Features

- 1-5 emoji sentiment scale (😞 😕 😐 🙂 😊)
- 1-5 confidence rating slider
- Optional text feedback
- Anonymous submission option
- Pulse check history
- Admin-initiated pulse requests
- Due date tracking
- Email notifications (optional)

Workflow

1. Admin creates pulse request → sends to specific team member
2. Team member receives notification
3. Team member submits pulse check
4. Admin views aggregated pulse data
5. Health score auto-updates based on pulse

---

6. Blocker Module

Purpose

Track and manage obstacles preventing objective progress.

Location

- `components/modals/blocker-modal.tsx`
- `components/modals/simple-blocker-modal.tsx`
- `app/api/objectives/[id]/blockers/route.ts`

Key Components

Frontend Components

1. **BlockerModal** (`components/modals/blocker-modal.tsx`)
   - Blocker creation/edit form
   - Title and description fields
   - Severity selector (low, medium, high, critical)
   - Status management (open, in-progress, resolved)
   - Anonymous submission option

2. **SimpleBlockerModal** (`components/modals/simple-blocker-modal.tsx`)
   - Quick blocker reporting
   - Simplified interface
   - Essential fields only

API Routes

- `POST /api/objectives/[id]/blockers` - Create blocker
- `PUT /api/objectives/[id]/blockers/[blockerId]` - Update blocker
- `DELETE /api/objectives/[id]/blockers/[blockerId]` - Delete blocker
- `GET /api/objectives/[id]/blockers` - List blockers

Features

- Severity levels: low, medium, high, critical
- Status tracking: open, in-progress, resolved
- Anonymous reporting option
- Rich text descriptions
- Blocker history
- Impact on health score
- Assigned owner (optional)

Business Logic

- Critical blockers significantly impact health score
- Resolved blockers don’t affect health
- Open blocker count displayed on objective cards
- Notifications for high/critical blockers

---

7. Team Management Module

Purpose

Admin-only module for managing team members, sending invitations, and controlling access.

Location

- `app/dashboard/team/page.tsx`
- `components/admin/team-management.tsx`
- `app/api/team/`

Key Components

Pages

- **Team Management Page** (`app/dashboard/team/page.tsx`)
- Admin-only access (role check)
- Team member list
- Invitation management
- Role assignment

Components

1. **TeamManagement** (`components/admin/team-management.tsx`)
   - Team member table
   - Invite new member form
   - Pending invitations list
   - Member deactivation
   - Role modification

API Routes

- `GET /api/team/members` - List team members
- `POST /api/team/invite` - Send team invitation
- `DELETE /api/team/members/[id]` - Remove team member
- `PUT /api/team/members/[id]/role` - Update member role

- `GET /api/team/invitations` - List invitations
- `POST /api/team/invitations/[token]/accept` - Accept invitation

Features

- Email-based invitations
- Role assignment (admin, team_member)
- Member activation/deactivation
- Invitation expiration (7 days)
- Pending invitation tracking
- Team member search
- Bulk actions

Security

- Admin-only access
- Token-based invitation acceptance
- Email verification
- Role-based permissions

---

8. Profile Module

Purpose

Allow users to view and update their personal information and account settings.

Location

- `app/dashboard/profile/page.tsx`
- `components/profile/profile-form.tsx`

Key Components

Pages

- **Profile Page** (`app/dashboard/profile/page.tsx`)
- User information display
- Edit profile form
- Account settings

Components

1. **ProfileForm** (`components/profile/profile-form.tsx`)
   - Name fields (firstName, lastName)
   - Email field (display only)
   - Password change option
   - Save/cancel actions
   - Form validation

API Routes

- `GET /api/profile` - Fetch user profile
- `PUT /api/profile` - Update user profile
- `POST /api/profile/change-password` - Change password

Features

- Update personal information
- Password change
- Email display (not editable)
- Avatar/profile picture (optional)
- Account creation date
- Last login information

---

9. API Module

Purpose

Backend API routes for all data operations, following RESTful conventions.

Location

- `app/api/`

API Structure

```
/api
├── auth/
│   ├── signup/route.ts
│   └── [...nextauth]/route.ts
├── objectives/
│   ├── route.ts (GET, POST)
│   └── [id]/
│       ├── route.ts (GET, PUT, DELETE)
│       ├── pulse/route.ts (POST)
│       ├── blockers/route.ts (GET, POST)
│       └── metrics/route.ts (GET, POST, PUT)
├── team/
│   ├── members/route.ts
│   ├── invite/route.ts
│   └── invitations/
│       └── [token]/accept/route.ts
├── pulse-requests/
│   ├── route.ts (GET, POST)
│   └── [id]/
│       ├── respond/route.ts (POST)
│       └── route.ts (GET, DELETE)
└── profile/
    ├── route.ts (GET, PUT)
    └── change-password/route.ts (POST)
```

Authentication Middleware

All API routes (except auth endpoints) require valid session:

```
const session = await getAuthSession()
if (!session?.user) {
  return NextResponse.json({ error: 'Unauthorized' }, { status: 401 })
}
```

Role-Based Authorization

Admin-only endpoints check user role:

```
if (session.user.role !== 'admin') {
  return NextResponse.json({ error: 'Forbidden' }, { status: 403 })
}
```

---

10. Shared UI Module

Purpose

Reusable UI components built on Radix UI and styled with Tailwind CSS.

Location

- `components/ui/`

Key Components

- **Badge** - Status and label indicators
- **Button** - Primary action buttons
- **Card** - Content containers
- **Checkbox** - Selection controls
- **Dialog/Modal** - Overlay dialogs
- **Dropdown Menu** - Context menus
- **Input** - Form input fields
- **Label** - Form field labels
- **Select** - Dropdown selectors
- **Slider** - Range inputs
- **Textarea** - Multi-line text input
- **Toast** - Notification system
- **Tooltip** - Hover information
- **Popover** - Contextual overlays
- **DateRangePicker** - Date selection
- **Breadcrumb** - Navigation trail

Design System

- **Colors**: Primary (blue), secondary (gray), success, warning, danger
- **Typography**: Inter font family
- **Spacing**: Tailwind CSS spacing scale
- **Shadows**: Subtle elevation system
- **Animations**: Smooth transitions and micro-interactions

Module Dependencies

```
Landing Page (Public)
    ↓
Authentication Module
    ↓
Dashboard Module (Protected)
    ├─> Objectives Module
    │   ├─> Pulse Check Module
    │   └─> Blocker Module
    ├─> Team Management Module (Admin only)
    ├─> Pulse Requests Module
    └─> Profile Module
```

Data Flow

Client-Side Flow

1. User interacts with component
2. Component calls API route
3. API route validates session
4. API route performs database operation
5. API route returns response
6. Component updates UI

Server-Side Flow

1. Page component fetches data
2. Database query executes
3. Data serialized and passed to client component
4. Client component renders with data
5. User interactions trigger client-side API calls

Module Communication

Inter-Module Communication Patterns

1. **Props Passing**: Parent → Child component data flow
2. **API Routes**: Client ↔ Server communication
3. **Context Providers**: Theme, Auth state
4. **URL Parameters**: Page-to-page navigation state
5. **Local Storage**: Client-side persistence (minimal)

Module Testing Strategy

Unit Tests

- Component rendering
- Form validation
- Utility functions

Integration Tests

- API route handlers
- Database operations
- Authentication flows

E2E Tests

- Complete user workflows
- Multi-module interactions
- Critical paths (signup, create objective, submit pulse)

Last Updated: 2025-10-03
Application Version: 1.0 (MVP)