CLAUDE.md - AI Assistant Guide

Last Updated: 2025-11-27 **Repository:** ISSB Membership Application **Purpose:** Comprehensive guide for AI assistants working on this codebase

---

Table of Contents

Project Overview

What This Is

The **ISSB Membership Application** is a comprehensive community engagement platform for the Islamic Society of the South Bay (ISSB). It provides:

- Membership management with tiered features
- AI-powered chatbot assistance
- Volunteer opportunity management
- Event management and gamification
- Donation portal

- Admin dashboard
- Communication portal

Current Status

- **Frontend:** Deployed at https://ngclt8fbwfb0.space.minimax.io
- **Backend:** Supabase (https://lsyimggqennkyxgajzvn.supabase.co)
- **Status:** ✅ Fully Operational
- **Phase:** Production (with ongoing enhancements)

Key Features

1. **Multi-tier Membership System** (Standard/Family/Volunteer)
2. **AI Help Assistant** (Google Gemini 2.0 Flash integration)
3. **Volunteer Management** (hour tracking, waivers, analytics)
4. **Event System** (with gamification and achievements)
5. **Donation Portal** (Stripe integration)
6. **Admin Dashboard** (comprehensive management interface)
7. **Knowledge Base** (searchable articles for AI integration)
8. **Role-Based Access Control** (user/admin/board roles)

Codebase Structure

Repository Layout

```
ISSB-Membership-App/
├── issb-portal/                # Frontend React application
│   ├── src/
│   │   ├── components/         # Reusable UI components
│   │   │   ├── ui/             # shadcn/ui components
│   │   │   ├── admin/          # Admin-specific components
│   │   │   ├── chat/           # AI chatbot components
│   │   │   ├── events/         # Event management
│   │   │   ├── volunteer/      # Volunteer portal
│   │   │   ├── announcements/  # Announcements system
│   │   │   ├── achievements/   # Gamification
│   │   │   ├── contests/       # Contest system
│   │   │   └── layout/         # Layout components
│   │   ├── pages/              # Page components
│   │   ├── features/           # Feature modules
│   │   ├── store/              # Redux state management
│   │   ├── contexts/           # React contexts
│   │   ├── hooks/              # Custom React hooks
│   │   ├── lib/                # Utilities & configs
│   │   │   └── supabase.ts    # Supabase client
│   │   ├── types/              # TypeScript type definitions
│   │   ├── styles/             # Global styles
│   │   └── tests/              # Test files
│   ├── public/                 # Static assets
│   ├── supabase/               # Legacy Supabase functions
│   ├── docs/                   # Documentation
│   ├── package.json
│   ├── vite.config.ts
│   ├── tsconfig.json
│   └── tailwind.config.js
│
├── supabase/                   # Supabase backend
│   └── functions/              # Edge Functions (Deno)
│       ├── _shared/            # Shared utilities
│       │   ├── rate-limiter.ts
│       │   └── security-headers.ts
│       ├── chat-message/       # AI chatbot endpoint
│       ├── chat-create-session/
│       ├── chat-history/
│       ├── chat-escalate/
│       ├── knowledge-base-search/
│       ├── admin-escalation-management/
│       ├── create-opportunity/
│       ├── list-opportunities/
│       ├── signup-for-opportunity/
│       ├── log-volunteer-hours/
│       ├── create-donation-payment/
│       ├── create-subscription/
│       ├── stripe-webhook/
│       └── [43 total edge functions]
│
├── docs/                       # Project documentation
│   ├── admin_dashboard/
│   ├── api_security/
│   ├── rls_enhancements/
│   └── frontend_enhancements/
```

```
│
├── package.json               # Root package (monorepo)
├── .gitignore
├── PROJECT_RECAP.md          # Project history & context
├── QUICK_REFERENCE.md        # Quick access to URLs & credentials
└── CLAUDE.md                 # This file
```

Key Directories

Frontend (`issb-portal/src/`)

- **components/ui/**: shadcn/ui components (Button, Card, Dialog, etc.)
- **components/admin/**: Admin dashboard components
- **components/chat/**: ChatWidget, ChatWindow for AI assistant
- **store/**: Redux slices for state management
- **lib/**: Utility functions and Supabase client configuration

Backend (`supabase/functions/`)

- **_shared/**: Common utilities (rate limiting, security headers)
- **chat-*/**: AI chatbot related functions
- **create-*/**: Resource creation endpoints
- **manage-*/**: Resource management endpoints

Technology Stack

Frontend

- **Framework:** React 18.3.1
- **Build Tool:** Vite 6.0.1
- **Language:** TypeScript 5.6.2
- **Package Manager:** pnpm
- **UI Framework:** Tailwind CSS 3.4.16
- **Component Library:** Radix UI (shadcn/ui)
- **State Management:**
- Redux Toolkit 2.9.2 (global state)
- Zustand 5.0.8 (lightweight state)
- React Query 5.64.0 (server state)
- **Routing:** React Router DOM v6
- **Form Handling:** React Hook Form 7.54.2 + Zod 3.24.1

- **Styling:**
- Tailwind CSS
- class-variance-authority
- clsx & tailwind-merge
- **Icons:** Lucide React 0.364.0
- **Payment:** Stripe React (@stripe/react-stripe-js)
- **Charts:** Recharts 2.12.4

Backend

- **Platform:** Supabase
- **Runtime:** Deno (Edge Functions)
- **Database:** PostgreSQL
- **Authentication:** Supabase Auth (JWT)
- **Storage:** Supabase Storage
- **AI Integration:** Google Gemini 2.0 Flash
- **Payment Processing:** Stripe

Development Tools

- **Linting:** ESLint 9.15.0
- **Type Checking:** TypeScript compiler
- **Testing:** (Framework to be determined)
- **Code Formatting:** Prettier 3.0.0
- **Git Hooks:** Husky 8.0.3 + lint-staged

Development Workflows

Getting Started

Prerequisites

- Node.js >= 18.0.0
- npm >= 8.0.0 or pnpm (preferred)
- Git

Initial Setup

```
# Clone repository
git clone <repository-url>
cd ISSB-Membership-App

# Install dependencies (frontend)
cd issb-portal
pnpm install

# Set up environment variables
cp .env.example .env
# Edit .env with your Supabase credentials
```

Environment Variables

Create `issb-portal/.env`:

```
VITE_SUPABASE_URL=https://lsyimggqennkyxgajzvn.supabase.co
VITE_SUPABASE_ANON_KEY=<your-anon-key>
```

Development Commands

Frontend Development

```
cd issb-portal

# Start development server (with auto-install)
pnpm dev

# Build for production
pnpm build

# Build for production (prod mode)
pnpm build:prod

# Type checking
pnpm run type-check

# Linting
pnpm lint

# Format code
pnpm run format

# Clean install
pnpm clean
```

Supabase Edge Functions

```
# Deploy a function
supabase functions deploy <function-name>

# Deploy with environment variables
supabase functions deploy <function-name> --no-verify-jwt

# Test function locally
supabase functions serve <function-name>

# View logs
supabase functions logs <function-name>
```

Git Workflow

Branch Naming

- Feature branches: `feature/description`
- Bug fixes: `fix/description`
- AI-generated branches: `claude/claude-md-*`

Commit Messages

Follow conventional commit format:

```
type(scope): description

- feat: New feature
- fix: Bug fix
- docs: Documentation changes
- style: Formatting, missing semicolons, etc.
- refactor: Code restructuring
- test: Adding tests
- chore: Maintenance tasks
```

Example:

```
git commit -m "feat(chat): add message history pagination"
git commit -m "fix(volunteer): resolve hour tracking calculation bug"
```

Pre-commit Hooks

The repository uses Husky for pre-commit hooks: - Automatic code formatting (Prettier) - Linting (ESLint) - Type checking (TypeScript)

Key Conventions

Code Style

TypeScript

- **Strict Mode:** Enabled
- **Path Aliases:** Use `@/` for `src/` imports `typescript import { supabase } from '@/lib/supabase' import { Button } from '@/components/ui/button'`
- **Type Safety:** Always define types, avoid `any`
- **Interfaces vs Types:** Prefer `interface` for objects, `type` for unions/intersections

React Components

```
// ✅ Preferred: Functional components with TypeScript
interface ButtonProps {
  variant?: 'primary' | 'secondary'
  onClick?: () => void
  children: React.ReactNode
}

export function Button({ variant = 'primary', onClick, children }: ButtonProps) {
  return (
    <button
      className={cn('base-styles', variant === 'primary' && 'primary-styles')}
      onClick={onClick}
    >
      {children}
    </button>
  )
}

// ❌ Avoid: Class components, missing types
export function Button(props: any) { ... }
```

Naming Conventions

- **Components:** PascalCase (`UserProfile.tsx` , `ChatWindow.tsx`)
- **Utilities:** camelCase (`formatDate.ts` , `validateEmail.ts`)
- **Constants:** UPPER_SNAKE_CASE (`API_ENDPOINTS` , `MAX_RETRY_COUNT`)
- **Types/Interfaces:** PascalCase (`User` , `ChatMessage` , `ApiResponse`)
- **CSS Classes:** kebab-case or Tailwind utilities

File Organization

```
ComponentName/
├── ComponentName.tsx          # Main component
├── ComponentName.test.tsx     # Tests
├── ComponentName.types.ts     # Type definitions
├── hooks/                     # Component-specific hooks
│   └── useComponentLogic.ts
└── index.ts                   # Exports
```

Or for simpler components:

```
ComponentName.tsx              # All-in-one file
```

State Management Guidelines

When to Use What

1. **Local State (useState):** Component-specific, non-shared state
2. **Context (React Context):** Theme, auth, app-wide settings
3. **Redux:** Complex global state, cross-feature data
4. **React Query:** Server state, API data caching
5. **Zustand:** Lightweight global state, simple stores

Redux Slice Pattern

```
// store/authSlice.ts
import { createSlice, PayloadAction } from '@reduxjs/toolkit'

interface AuthState {
  user: User | null
  isAuthenticated: boolean
}

const initialState: AuthState = {
  user: null,
  isAuthenticated: false,
}

export const authSlice = createSlice({
  name: 'auth',
  initialState,
  reducers: {
    setUser: (state, action: PayloadAction<User>) => {
      state.user = action.payload
      state.isAuthenticated = true
    },
    logout: (state) => {
      state.user = null
      state.isAuthenticated = false
    },
  },
})

export const { setUser, logout } = authSlice.actions
export default authSlice.reducer
```

Component Patterns

Composition Over Props Drilling

```
// ✅ Good: Use composition
<Card>
  <CardHeader>
    <CardTitle>Title</CardTitle>
  </CardHeader>
  <CardContent>Content here</CardContent>
</Card>

// ❌ Avoid: Too many props
<Card title="Title" content="Content here" showHeader={true} />
```

Custom Hooks

```
// hooks/useAuth.ts
export function useAuth() {
  const [user, setUser] = useState<User | null>(null)
  const [loading, setLoading] = useState(true)

  useEffect(() => {
    // Auth logic
  }, [])

  return { user, loading, login, logout }
}

// Usage in component
function Profile() {
  const { user, loading } = useAuth()

  if (loading) return <Loading />
  if (!user) return <Login />

  return <UserProfile user={user} />
}
```

Security Patterns

Edge Function Security

Rate Limiting

All edge functions should implement rate limiting:

```
import { checkRateLimit, createRateLimitResponse, RATE_LIMITS } from '../_shared/rate-limiter.ts'

const rateLimit = checkRateLimit(req, RATE_LIMITS.API_GENERAL)

if (!rateLimit.allowed) {
  return createRateLimitResponse(rateLimit.resetTime)
}
```

Available Rate Limit Configs: - `RATE_LIMITS.AUTH`: 5 requests/minute (login, signup) - `RATE_LIMITS.VOLUNTEER_SIGNUP`: 3 requests/hour - `RATE_LIMITS.FORM_SUBMISSION`: 10 requests/hour - `RATE_LIMITS.API_GENERAL`: 100 requests/minute

Security Headers

Always use security headers in edge functions:

```
import { getCompleteHeaders } from '../_shared/security-headers.ts'

return new Response(
  JSON.stringify({ data: result }),
  {
    status: 200,
    headers: getCompleteHeaders(),
  }
)
```

Headers Applied: - Content Security Policy (CSP) - X-Frame-Options (clickjacking protection) - X-Content-Type-Options (MIME sniffing protection) - HSTS (HTTP Strict Transport Security) - CORS headers - Cache control

Authentication

```
// Verify JWT token
const authHeader = req.headers.get('Authorization')
if (!authHeader) {
  return new Response('Unauthorized', { status: 401 })
}

const token = authHeader.replace('Bearer ', '')
const { data: { user }, error } = await supabase.auth.getUser(token)

if (error || !user) {
  return new Response('Unauthorized', { status: 401 })
}
```

Recent Authentication Enhancements (Nov 2025): - Email verification system implemented for new user registrations - Fixed orphaned auth user issues by implementing database triggers - Improved error messaging for unconfirmed email accounts - Added profile auto-creation triggers to prevent database inconsistencies

Input Validation

```
// Validate and sanitize input
function sanitizeInput(input: string): string {
  return input
    .trim()
    .replace(/[<>]/g, '') // Remove potential HTML tags
    .substring(0, 1000) // Limit length
}

// Validate with Zod
import { z } from 'zod'

const schema = z.object({
  email: z.string().email(),
  name: z.string().min(2).max(100),
})

const result = schema.safeParse(requestData)
if (!result.success) {
  return new Response(
    JSON.stringify({ error: 'Invalid input', details: result.error }),
    { status: 400 }
  )
}
```

Frontend Security

Environment Variables

- **Never commit** `.env` files
- Use `VITE_` prefix for public variables
- Keep sensitive keys server-side only

XSS Prevention

- React automatically escapes JSX
- Be careful with `dangerouslySetInnerHTML`
- Sanitize user input before rendering

CSRF Protection

- Supabase Auth handles CSRF tokens
- Use `sameSite` cookies

Database Architecture

Core Tables

Authentication & Users

- **auth.users**: Supabase auth users (managed by Supabase)
- **public.profiles**: Extended user profiles
- `user_id` (FK to auth.users)
- `full_name`, `email`, `phone`
- `membership_tier` (standard/family/volunteer)
- `role` (user/admin/board)

AI Help Assistant

- **chat_sessions**: User chat sessions
- **chat_messages**: All chat messages
- **knowledge_base_articles**: Searchable content for AI
- **escalation_requests**: Human agent escalation

Volunteer System

- **volunteer_opportunities**: Available volunteer positions
- **volunteer_signups**: User registrations
- **volunteer_hours**: Hour tracking
- **volunteer_waivers**: Waiver records

Events & Gamification

- **events**: Community events
- **event_registrations**: User event signups
- **badges**: Achievement definitions
- **user_badges**: User badge awards
- **contests**: Competition definitions
- **contest_submissions**: User contest entries

Communication

- **announcements**: System-wide announcements
- **family_members**: Family membership tracking

Row Level Security (RLS)

All tables have RLS enabled. Key policies:

```
-- Example: Users can only read their own profile
CREATE POLICY "Users can view own profile"
  ON profiles FOR SELECT
  USING (auth.uid() = user_id);

-- Admins can view all profiles
CREATE POLICY "Admins can view all profiles"
  ON profiles FOR SELECT
  USING (
    EXISTS (
      SELECT 1 FROM profiles
      WHERE user_id = auth.uid() AND role IN ('admin', 'board')
    )
  );
```

Database Conventions

- **Primary Keys:** `id` (UUID)
- **Foreign Keys:** `{table}_id` (e.g., `user_id`, `session_id`)
- **Timestamps:** `created_at`, `updated_at` (automatically managed)
- **Soft Deletes:** Use `deleted_at` timestamp when needed
- **Enums:** Use PostgreSQL ENUMs or CHECK constraints

API & Edge Functions

Edge Function Structure

```
// Standard edge function template
import { serve } from 'https://deno.land/std@0.168.0/http/server.ts'
import { createClient } from 'https://esm.sh/@supabase/supabase-js@2'
import { checkRateLimit, RATE_LIMITS, createRateLimitResponse } from '../_shared/rate-limiter.ts'
import { getCompleteHeaders } from '../_shared/security-headers.ts'

serve(async (req) => {
  // Handle CORS preflight
  if (req.method === 'OPTIONS') {
    return new Response('ok', { headers: getCompleteHeaders() })
  }

  try {
    // Rate limiting
    const rateLimit = checkRateLimit(req, RATE_LIMITS.API_GENERAL)
    if (!rateLimit.allowed) {
      return createRateLimitResponse(rateLimit.resetTime)
    }

    // Authentication
    const authHeader = req.headers.get('Authorization')
    if (!authHeader) {
      return new Response('Unauthorized', { status: 401 })
    }

    // Initialize Supabase client
    const supabaseUrl = Deno.env.get('SUPABASE_URL') ?? ''
    const supabaseKey = Deno.env.get('SUPABASE_SERVICE_ROLE_KEY') ?? ''
    const supabase = createClient(supabaseUrl, supabaseKey)

    // Get user from token
    const token = authHeader.replace('Bearer ', '')
    const { data: { user }, error: authError } = await supabase.auth.getUser(token)

    if (authError || !user) {
      return new Response('Unauthorized', { status: 401 })
    }

    // Parse request body
    const body = await req.json()

    // Business logic here
    const result = await processRequest(body, user)

    // Return response
    return new Response(
      JSON.stringify({ success: true, data: result }),
      {
        status: 200,
        headers: getCompleteHeaders(),
      }
    )

  } catch (error) {
    console.error('Error:', error)
    return new Response(
```

```
      JSON.stringify({
        error: 'Internal server error',
        message: error.message
      }),
      {
        status: 500,
        headers: getCompleteHeaders(),
      }
    )
  }
})
```

Key Edge Functions

AI Chatbot Functions

- **chat-message** (`/functions/v1/chat-message`)
- Processes user messages
- Integrates with Google Gemini 2.0 Flash
- Searches knowledge base
- Detects escalation needs
- Returns AI-generated responses
- **chat-create-session** (`/functions/v1/chat-create-session`)
- Creates new chat sessions
- Associates with user
- **chat-history** (`/functions/v1/chat-history`)
- Retrieves conversation history
- Supports pagination
- **chat-escalate** (`/functions/v1/chat-escalate`)
- Creates escalation requests
- Notifies admins

Volunteer Functions

- **create-opportunity**: Create volunteer positions
- **list-opportunities**: Get available opportunities
- **signup-for-opportunity**: Register for volunteer work
- **log-volunteer-hours**: Track hours worked
- **approve-volunteer-hours**: Admin approval

Donation Functions

- **create-donation-payment**: Process one-time donations
- **create-subscription**: Set up recurring donations
- **stripe-webhook**: Handle Stripe events

Complete Edge Functions List (43 Total)

1. admin-approve-volunteer-hours
2. admin-escalation-management
3. admin-update-user-role
4. approve-volunteer-hours
5. calculate-volunteer-waiver
6. chat-create-session
7. chat-escalate
8. chat-history
9. chat-message
10. create-admin-user
11. create-announcement
12. create-bucket-badge-icons-temp
13. create-bucket-contest-submissions-temp
14. create-bucket-event-images-temp
15. create-donation-payment
16. create-opportunity
17. create-payment-intent
18. create-student-subscription
19. create-subscription
20. create-volunteer-subscription
21. delete-announcement
22. delete-opportunity
23. get-community-metrics
24. get-member-assignments
25. get-membership-analytics
26. get-subscription-status
27. get-volunteer-progress
28. knowledge-base-search
29. list-announcements
30. list-opportunities

31. log-volunteer-hours
32. manage-family-members
33. manage-opportunity-capacity
34. manage-subscription
35. process-application
36. signup-for-opportunity
37. stripe-webhook
38. submit-application
39. test-gemini-api
40. update-announcement
41. update-opportunity
42. volunteer-analytics
43. withdraw-from-opportunity

API Response Format

Success Response:

```
{
  "success": true,
  "data": {
    // Response data
  }
}
```

Error Response:

```
{
  "error": {
    "code": "ERROR_CODE",
    "message": "Human-readable error message",
    "details": {} // Optional additional details
  }
}
```

Frontend Architecture

Component Hierarchy

```
App
├── AuthProvider (Context)
├── ThemeProvider (Context)
├── Router
│   ├── Public Routes
│   │   ├── Login
│   │   ├── Register
│   │   └── Home
│   └── Protected Routes
│       ├── Dashboard
│       ├── Profile
│       ├── Volunteer Portal
│       ├── Events
│       ├── Donations
│       └── Admin Dashboard (admin only)
└── ChatWidget (Global, floating)
```

Routing Structure

```
// Main routes
/                       # Home/Landing page
/login                  # Login page
/register               # Registration
/dashboard              # User dashboard
/profile                # User profile
/volunteer              # Volunteer portal
/events                 # Events listing
/donations              # Donation portal
/admin                  # Admin dashboard (protected)
/admin/users            # User management
/admin/volunteers       # Volunteer management
/admin/events           # Event management
/admin/analytics        # Analytics dashboard
```

UI Component Library (shadcn/ui)

Located in `src/components/ui/`. Key components:

- **button**: Buttons with variants
- **card**: Card containers
- **dialog**: Modal dialogs
- **dropdown-menu**: Dropdown menus
- **form**: Form components with validation
- **input**: Text inputs

- **select**: Select dropdowns
- **table**: Data tables
- **toast**: Toast notifications
- **tabs**: Tabbed interfaces
- **badge**: Status badges
- **avatar**: User avatars
- **accordion**: Collapsible sections

All components follow the shadcn/ui pattern and are fully customizable.

Styling Patterns

Tailwind CSS Utilities

```
// Use cn() helper to merge class names
import { cn } from '@/lib/utils'

<div className={cn(
  "base-styles",
  variant === "primary" && "primary-styles",
  isActive && "active-styles",
  className // Allow prop override
)} />
```

Component Variants (CVA)

```
import { cva, type VariantProps } from "class-variance-authority"

const buttonVariants = cva(
  "inline-flex items-center justify-center rounded-md font-medium transition-colors",
  {
    variants: {
      variant: {
        default: "bg-primary text-primary-foreground hover:bg-primary/90",
        destructive: "bg-destructive text-destructive-foreground hover:bg-destructive/90",
        outline: "border border-input hover:bg-accent hover:text-accent-foreground",
      },
      size: {
        default: "h-10 px-4 py-2",
        sm: "h-9 rounded-md px-3",
        lg: "h-11 rounded-md px-8",
      },
    },
    defaultVariants: {
      variant: "default",
      size: "default",
    },
  }
)
```

Data Fetching Patterns

Using Supabase Client

```
import { supabase } from '@/lib/supabase'

// Fetch data
const { data, error } = await supabase
  .from('profiles')
  .select('*')
  .eq('user_id', userId)
  .single()

// Insert data
const { data, error } = await supabase
  .from('volunteer_signups')
  .insert({ user_id: userId, opportunity_id: oppId })

// Update data
const { data, error } = await supabase
  .from('profiles')
  .update({ full_name: 'New Name' })
  .eq('user_id', userId)

// Real-time subscription
const subscription = supabase
  .channel('announcements')
  .on('postgres_changes',
    { event: 'INSERT', schema: 'public', table: 'announcements' },
    (payload) => console.log('New announcement:', payload)
  )
  .subscribe()
```

Using React Query

```
import { useQuery, useMutation } from '@tanstack/react-query'

// Fetch data
const { data, isLoading, error } = useQuery({
  queryKey: ['opportunities'],
  queryFn: async () => {
    const { data } = await supabase.from('volunteer_opportunities').select('*')
    return data
  },
})

// Mutate data
const mutation = useMutation({
  mutationFn: async (newOpportunity) => {
    const { data } = await supabase
      .from('volunteer_opportunities')
      .insert(newOpportunity)
    return data
  },
  onSuccess: () => {
    queryClient.invalidateQueries(['opportunities'])
  },
})
```

Testing Guidelines

Testing Philosophy

- Write tests for critical business logic
- Test user interactions, not implementation details
- Prioritize integration tests over unit tests
- Use TypeScript for type safety in tests

Test File Naming

```
ComponentName.test.tsx        # Component tests
utilityFunction.test.ts       # Utility tests
integration.test.tsx          # Integration tests
```

Testing Tools (Recommended)

- **Unit Testing:** Vitest (Vite-native)
- **Component Testing:** React Testing Library

- **E2E Testing:** Playwright
- **API Testing:** Postman/Insomnia

Example Test Structure

```
import { describe, it, expect } from 'vitest'
import { render, screen, fireEvent } from '@testing-library/react'
import { Button } from './Button'

describe('Button', () => {
  it('renders with correct text', () => {
    render(<Button>Click me</Button>)
    expect(screen.getByText('Click me')).toBeInTheDocument()
  })

  it('calls onClick when clicked', () => {
    const handleClick = vi.fn()
    render(<Button onClick={handleClick}>Click me</Button>)

    fireEvent.click(screen.getByText('Click me'))
    expect(handleClick).toHaveBeenCalledTimes(1)
  })
})
```

Deployment

Frontend Deployment

Current Platform: Minimax.io (https://ngclt8fbwfb0.space.minimax.io)

Build Process:

```
cd issb-portal
pnpm build:prod
```

Output: `issb-portal/dist/`

Environment Variables Required: - `VITE_SUPABASE_URL` - `VITE_SUPABASE_ANON_KEY`

Backend Deployment (Supabase)

Edge Functions:

```
# Deploy single function
supabase functions deploy <function-name>

# Deploy all functions
supabase functions deploy

# Set environment variables
supabase secrets set GOOGLE_GEMINI_API_KEY=<key>
supabase secrets set STRIPE_SECRET_KEY=<key>
```

Database Migrations:

```
# Create migration
supabase migration new <migration-name>

# Apply migrations
supabase db push
```

Production Checklist

- [ ] Environment variables configured
- [ ] Database migrations applied
- [ ] RLS policies verified
- [ ] Edge functions deployed
- [ ] API keys secured
- [ ] CORS settings correct
- [ ] Error monitoring enabled
- [ ] Performance monitoring enabled
- [ ] Backup strategy in place

Recent Bug Fixes & Improvements

Authentication System (Nov 2025)

Issue: Users unable to login after registration due to email verification requirements

Resolution: - Implemented email verification flow with proper user messaging - Added database triggers to auto-create user profiles - Fixed orphaned auth user issues - Improved error handling for unconfirmed accounts

Files Modified: - `issb-portal/src/contexts/AuthContext.tsx` - Enhanced signup and login flows - Database triggers added for automatic profile creation - RLS policies updated for better security

Payment Calculation (Nov 2025)

Issue: Incorrect next payment date calculation for past membership start dates

Resolution: - Fixed date calculation logic to handle edge cases - Improved payment scheduling for recurring memberships - Added validation for past dates

Impact: Ensures accurate billing cycles for all membership types

Known Issues

Currently, there are no critical known issues. The system is stable and fully operational.

For the latest status, check: - Recent commits: `git log --oneline -10` - Open issues in documentation files ending with `*_FIX*.md`

Common Tasks

Adding a New Component

1. **Create component file:** ```bash # For UI components touch issb-portal/src/components/ui/NewComponent.tsx

For feature components touch issb-portal/src/components/feature/NewComponent.tsx ```

1. **Component template:** ```typescript import { cn } from '@/lib/utils'

interface NewComponentProps { className?: string children?: React.ReactNode }

export function NewComponent({ className, children }: NewComponentProps) { return (

{children}
) } ```

1. **Export from index (if using directory structure):** `typescript // index.ts export { NewComponent } from './NewComponent'`

Adding a New Edge Function

1. **Create function directory:** `bash mkdir supabase/functions/new-function cd supabase/functions/new-function`
2. **Create index.ts:** `bash touch index.ts`
3. **Use the edge function template** (see API & Edge Functions)
4. **Deploy:** `bash supabase functions deploy new-function`

Adding a New Database Table

1. **Create migration:** `bash supabase migration new add_new_table`

2. **Write migration SQL:** ```sql -- Create table CREATE TABLE public.new_table (id UUID DEFAULT gen_random_uuid() PRIMARY KEY, user_id UUID REFERENCES auth.users(id) ON DELETE CASCADE, name TEXT NOT NULL, created_at TIMESTAMPTZ DEFAULT NOW(), updated_at TIMESTAMPTZ DEFAULT NOW());

-- Enable RLS ALTER TABLE public.new_table ENABLE ROW LEVEL SECURITY;

-- Add policies CREATE POLICY "Users can view own records" ON public.new_table FOR SELECT USING (auth.uid() = user_id);

CREATE POLICY "Users can insert own records" ON public.new_table FOR INSERT WITH CHECK (auth.uid() = user_id); ```

1. **Apply migration:** `bash supabase db push`

Adding Environment Variables

Frontend (.env):

```
echo "VITE_NEW_VARIABLE=value" >> issb-portal/.env
```

Backend (Supabase Secrets):

```
supabase secrets set NEW_VARIABLE=value
```

Debugging Issues

Frontend Issues

```
# Check for TypeScript errors
cd issb-portal && pnpm run type-check

# Check for linting errors
pnpm lint

# Clear cache and rebuild
rm -rf node_modules/.vite && pnpm dev
```

Backend Issues

```
# View edge function logs
supabase functions logs <function-name> --follow

# Test function locally
supabase functions serve <function-name>
```

Database Issues

```
# Check database status
supabase db status

# View database logs
supabase db logs

# Reset local database (⚠️ destructive)
supabase db reset
```

Important Files Reference

Configuration Files

File	Purpose	Location
`package.json`	Root dependencies & scripts	Root
`issb-portal/package.json`	Frontend dependencies	Frontend
`tsconfig.json`	TypeScript configuration	Multiple
`vite.config.ts`	Vite build configuration	Frontend
`tailwind.config.js`	Tailwind CSS config	Frontend
`.env`	Environment variables	Frontend (gitignored)
`.gitignore`	Git ignore patterns	Root

Documentation Files

File	Purpose
`PROJECT_RECAP.md`	Complete project history & status
`QUICK_REFERENCE.md`	URLs, credentials, quick commands
`CLAUDE.md`	This file - AI assistant guide
`docs/`	Various project documentation

Key Source Files

File	Purpose
`issb-portal/src/lib/supabase.ts`	Supabase client initialization
`issb-portal/src/App.tsx`	Main application component
`issb-portal/src/main.tsx`	Application entry point
`supabase/functions/_shared/`	Shared edge function utilities

Working with This Codebase - Guidelines for AI Assistants

Understanding Context

1. **Always read** `PROJECT_RECAP.md` first to understand current state
2. **Check** `QUICK_REFERENCE.md` for URLs and credentials
3. **Review** recent commits to understand recent changes
4. Use `git log --oneline -10` to see latest work
5. Check `*_FIX*.md` and `*_DEBUG*.md` files for recent bug fixes
6. Review `REGISTRATION_LOGIN_DEBUG_REPORT.md` for auth system details
7. **Examine** existing implementations before creating new ones

Best Practices

Before Making Changes

- ✅ Read existing code in the area you're modifying

- ✅ Understand the current patterns and conventions
- ✅ Check for similar implementations elsewhere
- ✅ Review related documentation

When Writing Code

- ✅ Follow existing naming conventions
- ✅ Use TypeScript with proper types
- ✅ Implement security patterns (rate limiting, headers, auth)
- ✅ Add error handling and validation
- ✅ Write clear, self-documenting code
- ✅ Add comments for complex logic only

When Adding Features

- ✅ Update relevant documentation
- ✅ Follow the established architecture
- ✅ Consider mobile responsiveness
- ✅ Implement proper loading states
- ✅ Add error boundaries where appropriate
- ✅ Test edge cases

What to Avoid

- ❌ Changing conventions without discussion
- ❌ Skipping security patterns
- ❌ Using `any` type in TypeScript
- ❌ Hardcoding values (use env variables)
- ❌ Ignoring existing patterns
- ❌ Creating duplicate implementations
- ❌ Committing sensitive data

Common Patterns to Recognize

Supabase Client Pattern

```
import { supabase } from '@/lib/supabase'
// Always use this singleton instance
```

Auth Check Pattern

```
const { data: { user } } = await supabase.auth.getUser()
if (!user) {
  // Handle unauthorized
}
```

Error Handling Pattern

```
try {
  const { data, error } = await supabase.from('table').select()
  if (error) throw error
  return data
} catch (error) {
  console.error('Operation failed:', error)
  // Show user-friendly error
}
```

Questions to Ask Before Making Changes

1. **Does this follow existing patterns?**
2. **Is this secure?** (Auth, validation, sanitization)
3. **Is this accessible?** (ARIA labels, keyboard navigation)
4. **Is this responsive?** (Mobile, tablet, desktop)
5. **Does this need error handling?**
6. **Should this be reusable?**
7. **Are there performance implications?**
8. **Do I need to update documentation?**

Additional Resources

External Documentation

- **React:** https://react.dev/
- **TypeScript:** https://www.typescriptlang.org/docs/
- **Tailwind CSS:** https://tailwindcss.com/docs
- **Supabase:** https://supabase.com/docs
- **Radix UI:** https://www.radix-ui.com/
- **shadcn/ui:** https://ui.shadcn.com/
- **Redux Toolkit:** https://redux-toolkit.js.org/

- **React Query:** https://tanstack.com/query/latest
- **React Hook Form:** https://react-hook-form.com/

Project Documentation

- `/docs/admin_dashboard/` - Admin dashboard documentation
- `/docs/api_security/` - API security patterns
- `/docs/rls_enhancements/` - RLS policies and enhancements
- `/docs/frontend_enhancements/` - Frontend architecture docs

Getting Help

- Review existing implementation examples
- Check the docs/ directory for detailed guides
- Look at test files for usage examples
- Examine similar features in the codebase

Changelog

2025-11-27

- Updated edge function count (43 total functions)
- Added authentication enhancements documentation
- Documented email verification implementation
- Added recent bug fixes and stability improvements
- Updated deployment and production status

2025-11-26

- Initial creation of CLAUDE.md
- Documented complete codebase structure
- Added comprehensive development guidelines
- Included security patterns and best practices
- Added common tasks and troubleshooting guides

Document Status: ✅ Complete and Current **Maintainer:** AI Assistants working on this project **Review Frequency:** Update with significant changes to architecture or workflows