IsoStack Glossary

 # IsoStack Glossary & Terminology

Quick Reference Guide for IsoStack Development

Last Updated: December 18, 2025

Table of Contents

Quick Reference Cheat Sheet

Users & Dashboards

```
P1 (Platform Admin) → PDash (Platform Dashboard)
C1 (Client Super Admin) → CDash (Client Admin Dashboard)
C2 (Client Admin) → CDash (Client Admin Dashboard)
C3 (Client User) → Udash (User Dashboard) or Mdash (Module Dashboard)

M1 (Single-Module Org) → Mdash only (default dashboard)
Mx (Multi-Module Org) → Udash + Mdash per module
```

Tenant Scoping (CRITICAL)

```
// ✅ ALWAYS scope by organizationId
```

```
where: { organizationId: user.organizationId }

// ❌ NEVER query without tenant scoping
where: {} // DATA LEAK!
```

Issue Creation Methods

```
Orange Dot → Location-aware (isocareComponentId set)
Blue Button → Manual (no location context)
Floating + → Manual (platform-wide)
```

Environment Pipeline

```
dev → techtest → staging → main → production
```

Abbreviations

Dashboards

Abbreviation	Full Name	Access Level
PDash	Platform Admin Dashboard	Platform Admins only (P1)
CDash	Client Admin Dashboard	Super Admins & Admins (C1, C2)
Udash	User Dashboard	Multi-module organization users (C3)
Mdash	Module Dashboard	Default for single-module orgs, module-specific view

Users & Roles

Abbreviation	Full Name	Database Role	Description

P1	Platform Admin	`platformAdmin: true`	Isoblue team, super-user access
C1	Client Super Admin	`role: 'OWNER'`	Org owner, billing contact, full control
C2	Client Admin	`role: 'ADMIN'`	User/settings management
C3	Client User	`role: 'MEMBER'`	Standard user access
Org	Organization	`Organization` model	Tenant/Client (interchangeable)

Modules & Features

Abbreviation	Full Name	Description
Mod	Module	Single module reference
Mods	Modules	Plural modules
M1	Single-Module Org	Organization with one module enabled
Mx	Multi-Module Org	Organization with multiple modules enabled
WL	White Label	Custom branding (ENTERPRISE feature)
FF	Feature Flag	Org-level feature toggle
IC	IsoCare	Issue tracking system
TTip / TT	Tooltip	Contextual help system

Technical

Abbreviation	Full Name	Description
DB	Database	PostgreSQL (Neon)

RLS	Row-Level Security	Database tenant isolation
authz	Authorization	Permission checks
authn	Authentication	Identity verification
env	Environment	dev/techtest/staging/prod
orgId	organizationId	Tenant scoping field
AR	Anticipated Resolution	Expected issue fix date
CR	Change Request	Grouped issue export

Architecture Terms

Multi-Tenancy

Tenant isolation at all layers - Every organization's data is strictly separated.

- **Tenant** - A single customer organization using the platform
- **Organization** - Database entity representing a tenant (`Organization` model)
- **organizationId** - Foreign key linking data to tenant (MANDATORY on queries)
- **Tenant Scoping** - Filtering all queries by `organizationId`
- **Cross-Tenant Data Leak** - Critical security vulnerability when data from one org appears in another

Multi-Schema Architecture

PostgreSQL schemas separate core from modules:

- **public schema** - Core platform tables (users, orgs, auth, settings, audit logs)
- **Module schemas** - Each module has its own schema (e.g., `bedrock`, `tailoraid`)
- **Schema isolation** - Modules cannot access each other's schemas
- **Core ↔ Module contract** - Modules depend on Core, never on each other

Cascading Configuration

Settings inherit with override capability:

```
Platform Defaults (P1 sets)
    ↓ (can override if allowed)
Organization Settings (C1 configures)
    ↓ (can override if policy permits)
Module Settings (per module)
```

Example: Platform sets default primary color (🔵#228be6) → Org overrides to brand color (🟠#ff6b35) → All modules inherit org color

---

User Roles & Permissions

Role Hierarchy

```
Platform Admin (P1)
    ↓ (super-user, cross-org access)
Organization Owner (C1)
    ↓ (full org control, billing)
Organization Admin (C2)
    ↓ (user/settings management)
Organization Member (C3)
    ↓ (standard access)
Anonymous User
    (no access, public pages only)
```

Permission Levels

Platform Admin (P1)

- Access all organizations
- Impersonate any user (except other platform admins)
- Manage platform-wide settings
- View/edit all modules and data
- Create/delete organizations
- Manage feature flags globally

Client Super Admin (C1 - OWNER)

- Full control over their organization
- Billing and subscription management
- Create/delete users
- Enable/disable modules
- Configure white label branding (ENTERPRISE)
- Delete organization (danger zone)

Client Admin (C2 - ADMIN)

- User management (invite, edit, suspend)
- Organization settings (excluding billing)
- Module configuration
- Cannot delete organization
- Cannot manage billing

Client User (C3 - MEMBER)

- Use enabled modules
- View own data
- Edit own profile
- No admin capabilities

UI Components & Patterns

AppShell

Platform-wide layout wrapper providing consistent navigation and structure.

Components:

- **Navbar** - Left sidebar with module navigation
- **Header** - Top bar with user menu, impersonation banner
- **Main** - Content area (all pages render here)
- **Footer** - Optional footer content

TooltipAnchor

Wrapper component for discoverable help tooltips:

```
<TooltipAnchor componentId="dashboard.welcome">
  <Title>Welcome to IsoStack</Title>
</TooltipAnchor>
```

Issue Indicators (Orange Dots)

Location-aware issue markers displayed on UI elements:

- **Orange dot** - Unresolved issue exists at this location
- **Red dot** - Critical priority issue
- **Green dot** - Issue recently resolved
- **Amber dot** - Issue in progress

Keyboard shortcut: `Ctrl+Shift+I` (toggle visibility)

Tooltip Help Indicators (Blue Icons)

Contextual help system with rich content:

- **Blue "?" icon** - Help available for this element
- **Tooltip content** - Can include text, images, videos
- **Three-tier inheritance** - Global → App Owner → Tenant

Keyboard shortcuts:

- `Shift+?` - Toggle help visibility
- `Ctrl+Shift+?` - Enter Edit Mode (admins only)

Database & Data Model

Core Models (public schema)

Organization

```
model Organization {
  id        String   @id @default(uuid())
```

```
  name      String
  slug      String   @unique

  // Multi-tenancy
  users     User[]

  // White label branding
  primaryColor   String  @default("#228be6")
  lightLogoUrl   String?
  darkLogoUrl    String?

  // Module access
  featureFlags   FeatureFlag?
}
```

User

```
model User {
  id             String   @id @default(cuid())
  email          String   @unique
  emailEncrypted String?  // AES-256 encrypted
  name           String?
  role           Role     // OWNER, ADMIN, MEMBER

  // Multi-tenancy (CRITICAL)
  organizationId String
  organization   Organization @relation(...)

  // Platform admin
  platformAdmin  PlatformAdmin?
}
```

PlatformAdmin

```
model PlatformAdmin {
  id     String @id @default(cuid())
  userId String @unique
  user   User   @relation(...)
}
```

AuditLog

```
model AuditLog {
  id             String   @id @default(cuid())
  action         String   // USER_CREATED, ISSUE_UPDATED, etc.
  entityType     String   // User, Issue, Tooltip, etc.
  entityId       String?
  metadata       Json?    // Additional context

  // Multi-tenancy
  userId         String
  organizationId String

  // Compliance
  ipAddress      String?
  userAgent      String?
  createdAt      DateTime @default(now())
}
```

Critical Database Patterns

Always scope by organizationId:

```
// Reading data
const items = await prisma.item.findMany({
  where: { organizationId: user.organizationId }
});

// Creating data
const item = await prisma.item.create({
  data: {
    ...data,
    organizationId: user.organizationId,
    createdById: user.id,
  }
});

// Updating data
const item = await prisma.item.update({
  where: {
```

```
    id: itemId,
    organizationId: user.organizationId, // Verify ownership!
  },
  data: { ...updates },
});
```

Field-level encryption:

```
import { encrypt, decrypt } from '@/lib/encryption';

// Store encrypted
const encrypted = encrypt(sensitiveData);
await prisma.user.create({
  data: { emailEncrypted: encrypted }
});

// Read decrypted
const user = await prisma.user.findUnique({...});
const email = decrypt(user.emailEncrypted);
```

Features & Modules

Core Platform Features

Authentication Methods:

1. **Magic Link** (Primary) - Passwordless email login
2. **WebAuthn/FIDO2** - Touch ID, Face ID, hardware keys
3. **Email/Password** (Legacy) - Traditional credentials
4. **OAuth** (Optional) - Google, Microsoft (planned)

Impersonation

Platform admins can view system as any user:

- Orange banner displays: "⚠️ Impersonating {email}"
- All actions logged to audit trail
- 4-hour session timeout

- Cannot impersonate other platform admins

White Label Branding (ENTERPRISE)

Organizations can customize:

- Logo (light/dark mode)
- Primary/secondary/accent colors
- Typography color
- Favicon
- Custom auth slug (custom login URL)

Tooltip System

Three-tier content inheritance:

1. **Global** - Platform-wide defaults
2. **App Owner** - Isoblue's recommended content
3. **Tenant** - Organization-specific overrides

IsoCare Issue Tracking

Location-aware issue management:

- **Orange dots** - Mark specific UI elements
- **Issue fields** - isocareComponentId, isocareModule, isocarePageUrl
- **Status tracking** - Draft → Pending → In Progress → Testing → Complete
- **Audit trail** - Status changes logged and emailed to platform admin

Module Types

Analytics Modules:

- **Bedrock** - Data visualization, Google Sheets integration, virtual fields

Vertical Modules:

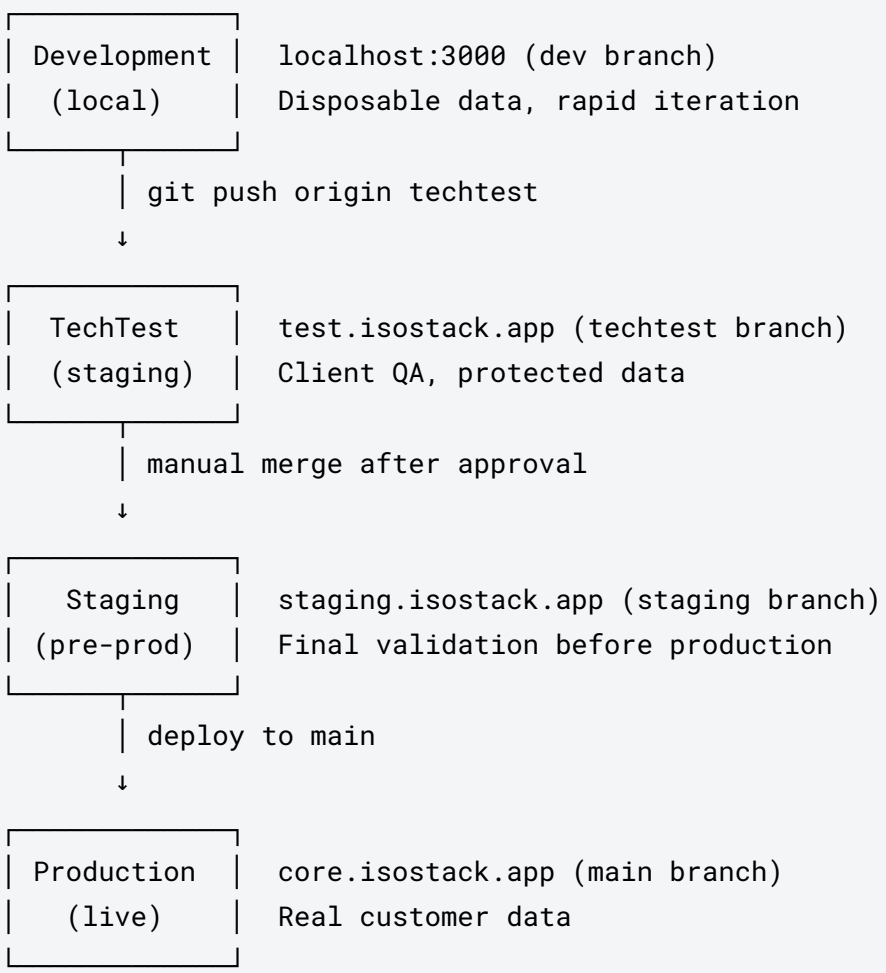
- **TailorAid** - Elderly care management
- **EmberBox** - Cannabis industry compliance
- **APIKeyChain** - API key management
- **LMSPro** - Learning management

Future Modules:

- Industry-specific applications built on same foundation

Development & Deployment

Environment Pipeline



Git Workflow

Standard Development Flow:

```
# 1. Work on dev branch
git checkout dev

# 2. Commit changes
git add -A
git commit -m "feat: Add user export feature"
```

```
# 3. Auto-merge to techtest (AI does this)
git checkout techtest
git merge dev -m "Merge dev: Add user export"
git push origin techtest
git checkout dev
```

Branch Protection:

- `dev` - Active development, frequent commits
- `techtest` - Staging for client review, auto-deploys to test.isostack.app
- `staging` - Pre-production validation
- `main` - Production, requires approval

Database Workflow

Schema Changes:

```
# 1. Update schema.prisma
# 2. Push to database
npm run db:push

# 3. Generate Prisma Client
npm run db:generate

# 4. Test changes locally

# 5. Create migration for production
npx prisma migrate dev --name add_new_field
```

Neon Branching:

- Each environment has separate Neon database branch
- TechTest = child of Production (data protection)
- Same database = same encryption key (MANDATORY)

Testing Commands

```
# Type check (do before deploying!)
```

```
npm run type-check

# Build check
npm run build

# Run dev server
npm run dev

# Open Prisma Studio
npm run db:studio

# Run migrations
npm run db:push
```

Common Usage Examples

Checking User Permissions

```
// Is platform admin?
const isPlatformAdmin = session.user.platformAdmin === true;

// Is organization owner?
const isOwner = session.user.role === 'OWNER';

// Is admin-level user?
const isAdmin = ['OWNER', 'ADMIN'].includes(session.user.role);

// Check specific permission
const canManageUsers = isAdmin || isPlatformAdmin;
```

Creating Tenant-Scoped Data

```
export async function createItem(data: ItemInput) {
  const session = await auth();
  if (!session?.user) throw new Error('Unauthorized');

  const item = await prisma.item.create({
```

```
    data: {
      ...data,
      organizationId: session.user.organizationId, // CRITICAL
      createdById: session.user.id,
    },
  });

  // Audit log
  await prisma.auditLog.create({
    data: {
      action: 'ITEM_CREATED',
      entityType: 'Item',
      entityId: item.id,
      userId: session.user.id,
      organizationId: session.user.organizationId,
    },
  });

  return item;
}
```

tRPC Protected Procedure

```
export const itemRouter = router({
  list: protectedProcedure
    .input(z.object({ status: z.string().optional() }))
    .query(async ({ ctx, input }) => {
      return await ctx.prisma.item.findMany({
        where: {
          organizationId: ctx.session.user.organizationId, // ALWAYS
          status: input.status,
        },
      });
    }),
});
```

Checking Feature Flags

```
const { data: features } = trpc.features.get.useQuery();
```

```
if (features?.billing) {
  // Show billing UI
}

if (features?.whiteLabel) {
  // Show branding customization
}
```

Quick Communication Examples

Using abbreviations in conversation:

> "C1 on CDash enabled bedrock mod via FF toggle, but C3 users on Mdash don't see it yet."

Translation: Client Super Admin on the admin dashboard enabled the Bedrock module via feature flag, but standard users on the module dashboard don't see it yet.

> "P1 needs to add AR date to issue via orange dot on PDash."

Translation: Platform Admin needs to set the Anticipated Resolution date on an issue created via location-aware indicator on the Platform Dashboard.

> "DB query missing orgId - potential data leak!"

Translation: Database query is missing organizationId tenant scoping - this is a critical security issue that could expose data across tenants.

> "WL branding not applying on Mdash after C1 updated on CDash."

Translation: White Label branding customization isn't showing on the Module Dashboard after the Client Super Admin updated it on the Admin Dashboard.

> "M1 org sees Mdash as default, but Mx org needs Udash to switch between mods."

Translation: Single-module organizations see the Module Dashboard as their default landing page, but

multi-module organizations need the User Dashboard to switch between different modules.

> "C3 user in Mx org can't access bedrock mod - check FF on CDash."

Translation: A standard user in a multi-module organization can't access the Bedrock module - verify the feature flag is enabled on the Client Admin Dashboard.

Related Documentation

- **ISOSTACK_CORE_SPECIFICATION.md** - Complete platform architecture
- **Work_Method.md** - AI-Chris collaboration workflow
- **CRITICAL_SECURITY_WORK_BEFORE_GOLIVE.md** - Pre-launch security requirements
- **Environment-Variables-Reference.md** - All environment configurations
- **Impersonation/IMPERSONATION_COMPLETE.md** - Impersonation feature spec

---

Last Updated: December 18, 2025
Maintained By: Chris (isocb) & GitHub Copilot
Version: 1.0.0