

INNOPAY ECOSYSTEM - PROJECT OVERVIEW

Last Updated: 2026-01-02 **Architecture:** Hub-and-Spokes Multi-Restaurant Payment System with Centralized Blockchain Polling

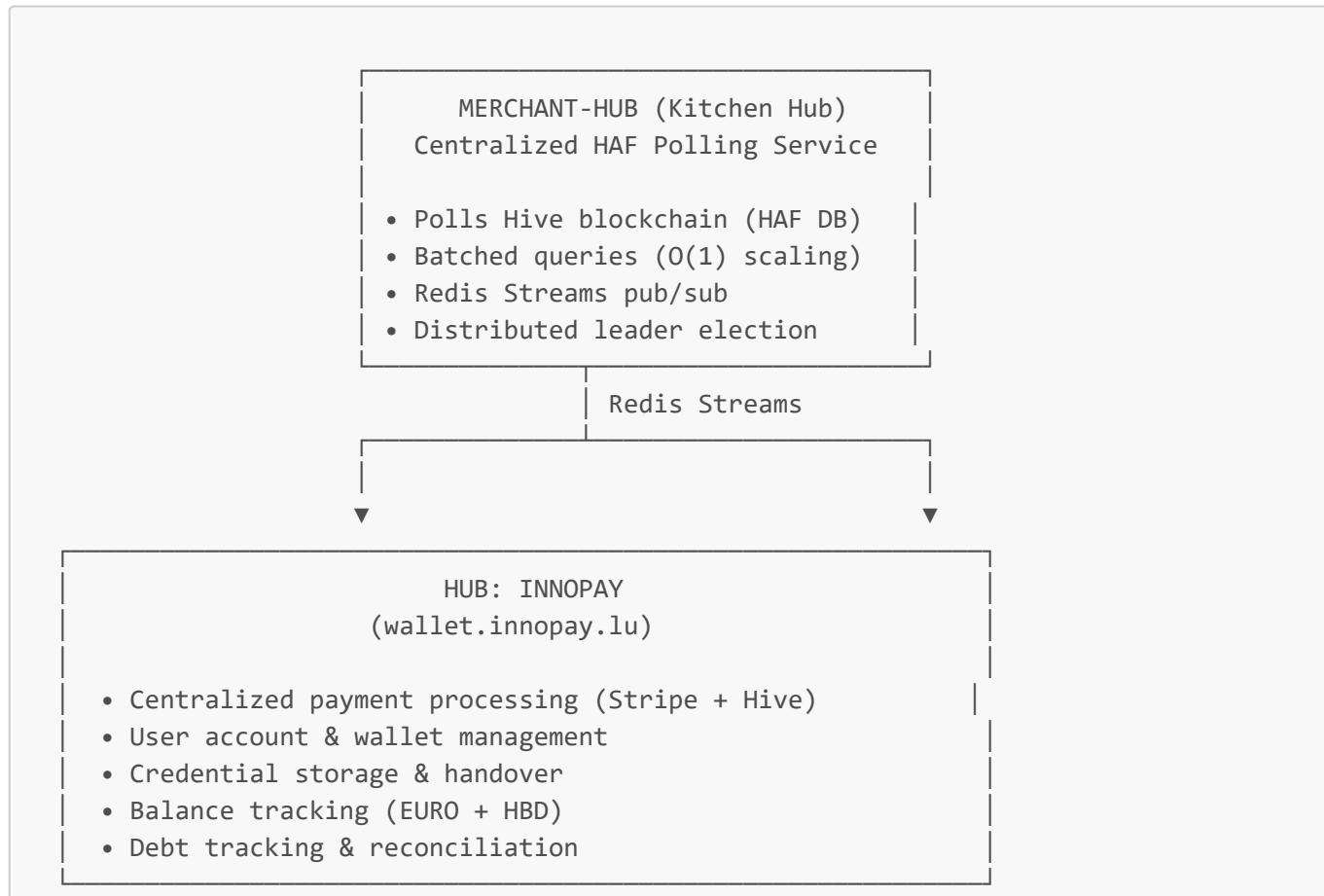
📋 TABLE OF CONTENTS

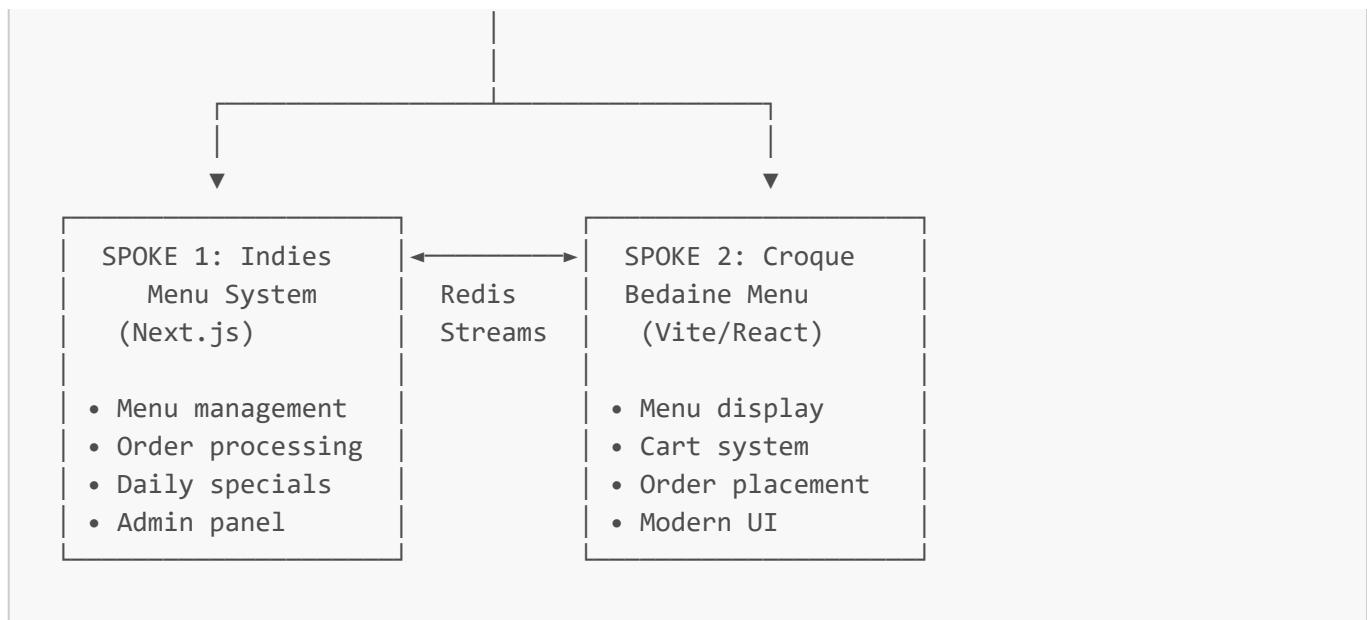
1. [Architecture Overview](#)
2. [Hub: Innopay](#)
3. [Merchant Hub: HAF Polling Infrastructure](#)
4. [Spoke 1: Indiesmenu](#)
5. [Spoke 2: Croque-Bedaine](#)
6. [Payment Flows](#)
7. [Technology Stack](#)
8. [Development Setup](#)
9. [Deployment](#)

🏗 ARCHITECTURE OVERVIEW

The Innopay ecosystem follows a **hub-and-spokes architecture** where:

- **Hub (innopay):** Centralized payment processor and wallet management system
- **Spokes:** Individual restaurant applications that integrate with the hub for payments





Key Design Principles

1. **Centralized Payment Processing:** All blockchain operations and Stripe payments happen in the hub
2. **Credential Security:** Hub manages sensitive account credentials, passes them securely to spokes
3. **Spoke Independence:** Each restaurant can have unique UI/UX and features
4. **Technology Flexibility:** Spokes can use different tech stacks (Next.js, Vite, etc.)
5. **Scalability:** Easy to add new restaurant spokes without modifying existing ones

HUB: INNOPAY

Repository: [../innopay](#) **Tech Stack:** Next.js 15 + TypeScript + Prisma + PostgreSQL **URL:** Production: [wallet.innopay.lu](#) | Dev: [localhost:3000](#)

Purpose

Innopay is the **central payment hub** that handles:

- User account creation and verification
- Wallet management (Hive blockchain)
- Payment processing (Stripe for EUR, Hive for HBD)
- Credential storage and secure handover to spokes
- Balance tracking and debt reconciliation

Key Features

1. Payment Processing

- **Stripe Integration:** EUR topups via credit/debit cards
- **Hive Blockchain:** HBD transfers and EURO token transfers
- **Dual-Currency Support:** Handles both EUR and HBD seamlessly
- **Debt Tracking:** Records outstanding debts when customer transfers fail

2. Account Management

- **BIP39 Seed Generation:** Secure wallet creation
- **Email Verification:** 6-digit code verification system
- **Multiple Account Support:** Users can have multiple Hive accounts
- **Credential Sessions:** Temporary secure sessions for credential handover

3. API Routes

Core Payment APIs:

- `/api/create-checkout-session` - Stripe checkout session creation
- `/api/wallet-payment` - Hive wallet payment execution
- `/api/sign-and-broadcast` - Blockchain transaction signing
- `/api/execute-order-payment` - Complete order payment flow
- `/api/webhooks` - Stripe webhook handler (unified architecture)

Account Management APIs:

- `/api/create-hive-account` - New Hive account creation
- `/api/account/retrieve` - Retrieve account info
- `/api/account/credentials` - Fetch account credentials
- `/api/account/create-credential-session` - Secure credential handover
- `/api/verify/*` - Email verification endpoints

Balance & Currency APIs:

- `/api/balance/euro` - Check EURO token balance
- `/api/currency` - EUR/USD exchange rate
- `/api/checkout/status` - Payment status checking

Database Schema (Prisma)

Core Models:

- `innouser` - User accounts with email verification
- `walletuser` - Hive wallet accounts
- `bip39seedandaccount` - Seed storage for account recovery
- `topup` - EUR topup transaction history
- `guestcheckout` - Guest checkout sessions
- `accountCredentialSession` - Temporary credential sessions (5min expiry)
- `outstanding_debt` - Tracks debts (EURO/HBD) when transfers fail
- `bonus` - Promotional bonus tracking
- `campaign` - Marketing campaign management
- `email_verification` - Email verification codes

Dependencies

Key Libraries:

- `@hiveio/dhive` - Hive blockchain integration
- `stripe` - Payment processing

- `@prisma/client` - Database ORM
- `bip39` - Wallet seed generation
- `@storacha/*` - Decentralized storage
- `resend` - Email service

Environment-Aware URL Resolution

The hub uses `getSpokeUrl(spoke: string)` function to resolve spoke URLs:

```
// Production
wallet.innopay.lu → menu.indies.lu

// Mobile Testing
192.168.x.x:3000 → 192.168.x.x:3001

// Localhost
localhost:3000 → localhost:3001
```

⌚ MERCHANT HUB: HAF POLLING INFRASTRUCTURE

Repository: [../merchant-hub](#) **Tech Stack:** Next.js 15 + TypeScript + PostgreSQL (HAF) + Upstash Redis

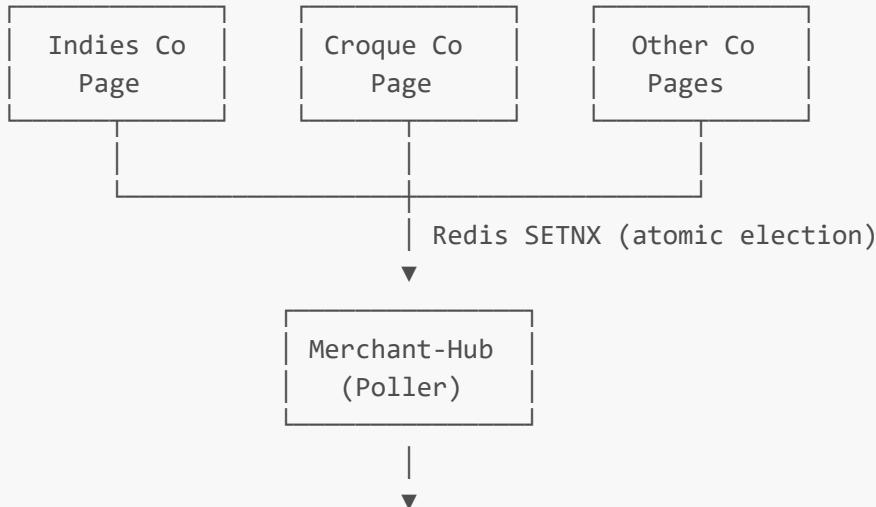
Deployment: Vercel (serverless with Cron)

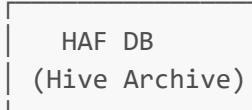
Purpose

The merchant-hub is a **centralized blockchain polling service** that solves the "diabolo topology" problem. Instead of each restaurant independently polling the Hive blockchain (which would create $N \times 3$ database queries for N restaurants), merchant-hub centralizes all polling into **3 total queries** (one per currency).

Architecture: Distributed Leader Election

The merchant-hub uses a **distributed leader election** pattern where multiple restaurant co (customer-facing) pages coordinate to elect a single poller:





Election Process:

1. First co page to open calls [/api/wake-up](#)
2. Attempts Redis `SETNX` with random collision-avoidance delay (Ethernet-like)
3. Winner becomes "poller", polls every 6 seconds
4. Losers subscribe to Redis Streams for transfer notifications
5. Poller maintains heartbeat (30s TTL)
6. If poller dies, another co page takes over

Polling Modes

Active Mode (6-second polling):

- Triggered when at least one restaurant co page is open
- Poller queries HAF database every 6 seconds
- Publishes transfers to Redis Streams
- Maintains heartbeat lock

Sleeping Mode (1-minute polling):

- Vercel Cron fallback when all shops closed
- Polls every 1 minute via [/api/cron-poll](#)
- Only runs if no active 6-second poller detected
- Ensures transfers aren't missed overnight

Batched Query Architecture (O(1) Scaling)

The Scaling Problem (Old Approach):

```
2 restaurants × 3 currencies = 6 queries
400 restaurants × 3 currencies = 1,200 queries
Execution time: ~24 seconds at scale
```

The Solution (Batched Queries):

```
// ONE query for ALL restaurants per currency
const hbdResult = await pool.query(
  `SELECT id, to_account, from_account, amount, symbol, memo
   FROM hafsql.operation_transfer_table
  WHERE to_account = ANY($1) -- ['indies.cafe', 'croque.bedaine', ...]
        AND symbol = 'HBD'
        AND id > $2
  ORDER BY id DESC
```

```

    LIMIT 100` ,
    [allAccounts, minLastId]
);

```

Performance Comparison:

Restaurants	Old Queries	New Queries	Improvement	Est. Time
2	6	3	2x	~60ms
4	12	3	4x	~65ms
400	1,200	3	400x	~65ms

Key Insight: Network latency (10-50ms per round-trip) dominates query execution time. By reducing from N queries to 3 queries, we eliminate the dominant cost factor.

Multi-Environment Polling

Since batched queries scale at O(1), merchant-hub queries **ALL accounts** (prod + dev) simultaneously:

```

const accounts = [
  'indies.cafe',      // Production
  'indies-test',      // Development
  'croque.bedaine',   // Production
  'croque-test'       // Development
];
// Still just 3 queries total!

```

Each transfer includes the **account** field so restaurant co pages can filter by environment if needed.

Currency Support

Three currencies polled:

1. **HBD** (Hive-Backed Dollars) - Native Hive token

- Query: [operation_transfer_table](#)
- Fast native transfers
- Base layer blockchain

2. **EURO** - Hive-Engine token

- Query: [operation_custom_json_view](#)
- Pegged to EUR (1:1)
- Layer 2 smart contract

3. **OCLT** - Hive-Engine loyalty token

- Query: [operation_custom_json_view](#)

- Community token
- Layer 2 smart contract

Redis Streams Integration

Stream Architecture:

transfers:indies	→ Indies restaurant transfers
transfers:croque-bedaine	→ Croque restaurant transfers
system:broadcasts	→ System coordination messages
polling:poller	→ Current poller identity
polling:heartbeat	→ Poller liveness
polling:mode	→ active-6s sleeping-1min
lastId:indies:HBD	→ Last processed HBD transfer ID
lastId:indies:EURO	→ Last processed EURO transfer ID

Transfer Object Structure:

```
interface Transfer {
  id: string; // HAF operation ID
  restaurant_id: string; // 'indies' | 'croque-bedaine'
  account: string; // 'indies.cafe' | 'indies-test'
  from_account: string; // Customer Hive account
  amount: string; // Transfer amount
  symbol: 'HBD' | 'EURO' | 'OCLT';
  memo: string; // Order details + table info
  parsed_memo?: string; // Decoded memo
  received_at: string; // ISO timestamp
  block_num?: number; // Blockchain block number
}
```

API Routes

Coordination APIs:

- [/api/wake-up](#) - Co page initialization, attempt leader election
- [/api/heartbeat](#) - Poller heartbeat maintenance (every 5s)
- [/api/poll](#) - Active polling endpoint (every 6s)
- [/api/cron-poll](#) - Vercel Cron fallback (every 1min)

Monitoring APIs (future):

- [/api/status](#) - System health check
- [/api/metrics](#) - Polling statistics

Key Features

1. **Zero Missed Transfers:** Dual-mode (active + sleeping) ensures 24/7 coverage

2. **Automatic Failover:** Poller death triggers automatic re-election
3. **Scalable:** O(1) query complexity regardless of restaurant count
4. **Environment-Agnostic:** Works in Vercel prod/preview/dev environments
5. **Memo Filtering:** Per-restaurant memo patterns (e.g., "TABLE" keyword)
6. **LastId Tracking:** Per-restaurant, per-currency cursor for deduplication

Technology Stack

Category	Technology
Framework	Next.js 15 (serverless)
Language	TypeScript 5
Database	PostgreSQL (HAF - Hive Application Framework)
Cache/Pub-Sub	Upstash Redis (serverless)
Deployment	Vercel Pro (10s timeout)
Cron	Vercel Cron (1-minute)
DB Driver	pg (node-postgres)

Configuration

Restaurant Config ([lib/config.ts](#)):

```
export const RESTAURANTS: RestaurantConfig[] = [
  {
    id: 'indies',
    name: 'Indies Restaurant',
    accounts: {
      prod: 'indies.cafe',
      dev: 'indies-test'
    },
    currencies: ['HBD', 'EURO', 'OCLT'],
    memoFilters: {
      HBD: '%TABLE %',
      EURO: '%TABLE %',
      OCLT: '%TABLE %'
    }
  },
  // ... more restaurants
];
```

Vercel Deployment Constraints

Serverless Limitations:

- Max execution time: 10s (Vercel Pro)
- No long-running processes

- Stateless functions
- Cold starts possible

Solutions:

- External triggers (co page wake-up calls)
- Redis-based coordination
- Vercel Cron for fallback
- Fast queries (<1s execution)

Future Enhancements

Planned:

- /api/status endpoint for monitoring
- Prometheus metrics export
- Transfer confirmation/acknowledgment
- Historical transfer queries
- WebSocket streaming for real-time updates

Scaling Beyond 400 Restaurants:

- Current LIMIT 100 (HBD) and 1000 (tokens) handles expected volume
- If spike exceeds LIMIT, next poll (6s later) catches remainder
- Transfers delayed by seconds, not lost
- Midnight-based bounds (discussed but not implemented) as future optimization

SPOKE 1: INDIESMENU

Repository: [./indiesmenu](#) (current) **Tech Stack:** Next.js 15 + TypeScript + Prisma + PostgreSQL **URL:**

Production: menu.indies.lu | Dev: localhost:3001

Purpose

Indiesmenu is a **full-featured restaurant menu and ordering system** for Indies restaurant, with:

- Digital menu display
- Shopping cart and checkout
- Daily specials management
- Order history tracking
- Admin panel for menu management
- Multi-language support (FR)

Key Features

1. Menu System

- **Dynamic Menu:** Fetches menu from database with 7-day cache
- **Daily Specials:** Separate management for rotating daily dishes
- **Categories:** Soups, salads, main dishes, desserts, drinks

- **Allergen Information:** Track and display allergen info
- **Image Optimization:** Automated WebP conversion and optimization
- **Print-Friendly Display:** A3 landscape printout page ([/display/printout](#))

2. Payment Integration with Hub

Payment Flows:

- **Flow 4:** Create account only (no order) - Returns credentials to spoke
- **Flow 5:** Create account + order - Returns credentials + processes payment
- **Flow 6:** Pay with existing account (two-leg dual-currency)
- **Flow 7:** Pay with topup (unified webhook architecture)

Integration Pattern:

```
// 1. Redirect to hub with order context
window.location.href = `${hubUrl}/?
restaurant=indies&amount=${total}&table=${table}&...`;

// 2. Hub processes payment and redirects back
// Return URL: menu.indies.lu?/
order_success=true&session_id=...&credential_token=...

// 3. Spoke receives credentials and updates balance
const response = await fetch(`${hubUrl}/api/account/credentials` , {
  method: 'POST',
  body: JSON.stringify({ credentialToken })
});
```

3. State Management

- **CartContext:** Shopping cart with localStorage persistence
- **React Query:** Balance fetching with automatic caching and refetching
- **MiniWallet:** Display EURO balance, account name, quick topup

4. Admin Panel

- **Menu Management:** CRUD operations for dishes
- **Daily Specials:** Manage rotating daily menu
- **Image Management:** Upload, match, and optimize images
- **Order Fulfillment:** Mark orders as prepared/delivered
- **Cache Control:** Manual menu cache invalidation

API Routes

Menu APIs:

- [/api/menu](#) - Full menu with caching
- [/api/dishes](#) - Dish CRUD operations

- `/api/daily-specials` - Daily specials management
- `/api/admin/*` - Admin panel APIs

Integration APIs:

- `/api/balance/euro` - Fetch balance from Hive-Engine
- `/api/currency` - Exchange rate proxy
- `/api/fulfill` - Order fulfillment
- `/api/orders/history` - Order history

Database Schema

Core Models:

- `Category` - Menu categories
- `Dish` - Menu items with pricing, allergens
- `Order` - Customer orders with items
- `DailySpecial` - Rotating daily menu items

Key Components

- `app/menu/page.tsx` - Main menu page with cart and checkout (1600+ lines)
- `app/context/CartContext.tsx` - Shopping cart state management
- `hooks/useBalance.ts` - React Query balance hook
- `app/display/printout/page.tsx` - Printer-optimized daily specials

Features Unique to Indiesmenu

1. **Call Waiter Button:** Uses FLOW 6 architecture to notify staff
2. **Table-Based Ordering:** URL parameter `?table=X` for table tracking
3. **Daily Specials Display:** Separate page optimized for TV/print display
4. **Menu Cache Invalidation:** Auto-invalidates on dish CRUD operations
5. **Image Optimization Scripts:** Batch processing for menu images

SPOKE 2: CROQUE-BEDAINE

Repository: [..../croque-bedaine](#) **Tech Stack:** Vite + React 18 + TypeScript + Supabase **URL:** Dev: localhost:8080

Purpose

Croque-Bedaine is a **modern Vite-based restaurant menu application** built with:

- Vite for fast development and building
- React 18 with TypeScript
- shadcn/ui component library
- Supabase for backend (database + auth)
- React Query for data fetching

Key Differences from Indiesmenu

Feature	Indiesmenu	Croque-Bedaine
Framework	Next.js 15	Vite 5
Backend	Self-hosted API routes	Supabase
Database	PostgreSQL + Prisma	Supabase (PostgreSQL)
Rendering	Server + Client	Client-side (SPA)
Build Time	Slower (Next.js)	Faster (Vite)
UI Library	Custom + Tailwind	shadcn/ui
Routing	Next.js file-based	React Router

Tech Stack

Core Dependencies:

- `vite` - Build tool and dev server
- `react` + `react-dom` - UI framework
- `@supabase/supabase-js` - Backend integration
- `@tanstack/react-query` - Data fetching and caching
- `react-router-dom` - Client-side routing
- `shadcn/ui` - Component library (40+ Radix UI components)
- `tailwindcss` - Styling
- `zod` - Schema validation
- `react-hook-form` - Form management

Project Structure

```

croque-bedaine/
├── src/
│   ├── components/
│   │   ├── ui/          # shadcn/ui components (40+ files)
│   │   ├── CartSheet.tsx # Shopping cart UI
│   │   ├── DrinksSection.tsx
│   │   ├── Header.tsx
│   │   ├── MenuSection.tsx
│   │   └── ...
│   ├── App.tsx          # Main app component
│   └── main.tsx         # Entry point
├── public/             # Static assets
└── vite.config.ts     # Vite configuration
└── package.json

```

Configuration

Vite Config:

- Dev server on port **8080**
- Fast refresh with SWC compiler
- Path alias: `@/ → ./src/`
- Component tagging for development

Integration with Hub (Status Unknown)

Note: The integration pattern with the innopay hub for this spoke is not yet evident from the codebase. This may be:

1. Not yet implemented
2. Implemented differently than indiesmenu
3. Using Supabase edge functions for hub communication

To be documented: How croque-bedaine integrates with innopay for payments.

PAYMENT FLOWS

The hub-and-spokes architecture supports multiple payment flows:

Flow 4: Create Account Only

Trigger: User clicks "Create Wallet" without placing an order **Process:**

1. Spoke redirects to hub: `?restaurant=indies&amount=0`
2. Hub creates Hive account
3. Hub creates credential session and returns token
4. Spoke receives credentials and stores in localStorage
5. MiniWallet appears with account name

Files:

- Hub: `innopay/app/user/page.tsx` (credential handover)
- Spoke: `indiesmenu/app/menu/page.tsx:502-529` (Flow 4 detection)

Flow 5: Create Account + Order

Trigger: User with no account places order **Process:**

1. Spoke redirects to hub with order details
2. Hub creates account + processes Stripe payment
3. Hub executes blockchain transfer to restaurant
4. Hub returns credentials to spoke
5. Spoke shows success banner

Architecture: Original flow, still supported

Flow 6: Pay with Existing Account (Two-Leg Dual-Currency)

Trigger: User with existing account places order (sufficient balance) **Process:**

1. Check if sufficient EURO balance available
2. **Leg 1:** Transfer EURO tokens to restaurant (Hive-Engine)
3. **Leg 2:** Transfer HBD to innopay (Hive native)
4. Both legs signed locally, broadcast via hub
5. Restaurant receives payment immediately

Architecture: November 2025 - Two-leg dual-currency **Status:** STABLE - DO NOT BREAK **Files:** [indiesmenu/app/menu/page.tsx:1498-1670](#)

Flow 7: Pay with Topup (Unified Webhook)

Trigger: User with account but insufficient EURO balance **Process:**

1. Redirect to hub for Stripe checkout
2. User completes EUR topup
3. **Unified webhook** processes both:
 - Topup account balance
 - Execute pending order payment
4. Return to spoke with success parameters
5. Cart clears, balance updates

Architecture: December 2025 - Unified webhook (single webhook handles topup + payment) **Status:**
PRODUCTION READY Files:

- Hub: [innopay/app/api/webhooks/route.ts](#) (unified webhook)
- Spoke: [indiesmenu/app/menu/page.tsx:1452-1496](#) (Flow 7 checkout)

Guest Checkout

Trigger: User without account places order **Process:**

1. Redirect to hub for guest checkout
2. Stripe payment processed
3. Hub executes blockchain transfer to restaurant
4. No account created, one-time payment

Files: [innopay/app/api/checkout/guest/route.ts](#)

Call Waiter

Purpose: Notify restaurant staff without payment **Process:**

1. User clicks "Call Waiter" button
2. Uses FLOW 6 architecture (sign-and-broadcast)
3. Sends tiny HBD transfer (0.001) with memo
4. Blue notification banner appears (15 seconds)

Files: [indiesmenu/app/menu/page.tsx:1100-1280](#)



Hub (innopay)

Category	Technology
Framework	Next.js 15.5
Language	TypeScript 5
Database	PostgreSQL + Prisma 6.11
Payment	Stripe 18.3
Blockchain	@hiveio/dhive 1.3
Storage	Storacha (decentralized) + Bunny CDN
Email	Resend
Styling	Tailwind CSS 4

Spoke 1 (indiesmenu)

Category	Technology
Framework	Next.js 15.5
Language	TypeScript 5
Database	PostgreSQL + Prisma 6.11
State	React Query 5.90
Blockchain	@hiveio/dhive 1.3
Image Processing	Sharp 0.34
Testing	Jest 30
Styling	Tailwind CSS 4

Spoke 2 (croque-bedaine)

Category	Technology
Build Tool	Vite 5.4
Framework	React 18.3
Language	TypeScript 5
Backend	Supabase
State	React Query 5.83
UI Components	shadcn/ui (Radix UI)
Routing	React Router 6.30

Category	Technology
Forms	React Hook Form 7.61 + Zod 3.25
Styling	Tailwind CSS 3.4

Common Dependencies

All Projects Share:

- TypeScript 5.x
- Tailwind CSS
- React Query (TanStack Query)
- Modern React (18+)

Key Differences:

- **Build:** Next.js (innopay, indiesmenu) vs Vite (croque-bedaine)
- **Backend:** Self-hosted API routes vs Supabase
- **Components:** Custom vs shadcn/ui library

🚀 DEVELOPMENT SETUP

Prerequisites

- Node.js 20+ (recommended: use nvm)
- PostgreSQL (for innopay and indiesmenu)
- Supabase account (for croque-bedaine)
- npm or pnpm

Hub Setup (innopay)

```
cd innopay

# Install dependencies
npm install

# Setup environment variables
cp .env.example .env
# Edit .env with your credentials:
# - POSTGRES_URL
# - STRIPE_SECRET_KEY
# - STRIPE_WEBHOOK_SECRET
# - RESEND_API_KEY
# - DATABASE_URL (Prisma)

# Run database migrations
npm run migrate:dev

# Start dev server
```

```
npm run dev  
# → http://localhost:3000
```

Spoke 1 Setup (indiesmenu)

```
cd indiesmenu  
  
# Install dependencies  
npm install  
  
# Setup environment variables  
cp .env.example .env  
# Edit .env with your credentials:  
# - POSTGRES_URL  
# - NEXT_PUBLIC_HUB_URL=http://localhost:3000  
  
# Run database migrations  
npm run migrate:dev  
  
# Start dev server  
npm run dev  
# → http://localhost:3001
```

Spoke 2 Setup (croque-bedaine)

```
cd croque-bedaine  
  
# Install dependencies  
npm install  
  
# Setup environment variables  
cp .env.example .env  
# Edit .env with Supabase credentials:  
# - VITE_SUPABASE_URL  
# - VITE_SUPABASE_ANON_KEY  
  
# Start dev server  
npm run dev  
# → http://localhost:8080
```

Development Workflow

1. **Start Hub First:** Always run innopay before spokes
2. **Environment URLs:** Spokes will automatically detect hub URL based on environment
3. **Database Migrations:** Run migrations when switching branches or after pull
4. **Testing Payments:** Use Stripe test mode with test cards
5. **Blockchain Testing:** Set `RECIPIENT_OVERRIDE` in innopay to test without real transfers

Testing the System

Test Flow 4 (Create Account):

1. Visit <http://localhost:3001/menu?table=1>
2. Click "Create Wallet" (no order)
3. Complete account creation on hub
4. Verify credentials returned to spoke
5. Check MiniWallet displays account name

Test Flow 6 (Pay with Account):

1. Ensure you have an account with EURO balance
2. Add items to cart (ensure total < balance)
3. Click "Pay with Account"
4. Verify both EURO and HBD transfers execute
5. Check balance updates correctly

Test Flow 7 (Topup + Pay):

1. Add items to cart (ensure total > balance)
2. Click checkout
3. Complete Stripe payment on hub
4. Verify redirect back with success
5. Check cart cleared and balance updated

📦 DEPLOYMENT

Vercel Deployment (Recommended)

All three projects are configured for Vercel deployment.

Hub Deployment

```
cd innopay

# Build command (in Vercel settings)
npm run vercel-build
# → npx prisma migrate deploy && npx prisma generate && next build

# Environment variables needed:
# - POSTGRES_URL
# - STRIPE_SECRET_KEY
# - STRIPE_WEBHOOK_SECRET
# - RESEND_API_KEY
# - DATABASE_URL
# - RECIPIENT_OVERRIDE (optional, for testing)
```

Production URL: wallet.innopay.lu

Spoke 1 Deployment

```
cd indiesmenu

# Build command (in Vercel settings)
npm run vercel-build

# Environment variables needed:
# - POSTGRES_URL
# - NEXT_PUBLIC_HUB_URL=https://wallet.innopay.lu
```

Production URL: menu.indies.lu

Spoke 2 Deployment

```
cd croque-bedaine

# Build command
npm run build
# → vite build

# Environment variables needed:
# - VITE_SUPABASE_URL
# - VITE_SUPABASE_ANON_KEY
# - VITE_HUB_URL (for innopay integration, TBD)
```

Note: Croque-bedaine can be deployed to Vercel, Netlify, or any static hosting.

Deployment Checklist

Before Deploying:

- ☐ Run `npm run build` locally to check for errors
- ☐ Verify all environment variables are set
- ☐ Test database migrations with `npm run migrate:deploy`
- ☐ Update Stripe webhook URLs to production
- ☐ Set `RECIPIENT_OVERRIDE` appropriately (remove for production)
- ☐ Test all payment flows in staging environment

After Deploying:

- ☐ Verify Stripe webhooks are receiving events
- ☐ Test end-to-end payment flows
- ☐ Check database migrations applied successfully
- ☐ Monitor error logs for issues

- Test mobile responsiveness

Database Migrations

Automatic Migration on Deploy: Both innopay and indiesmenu use `vercel-build` script that runs:

```
npx prisma migrate deploy && npx prisma generate && next build
```

This ensures database schema is updated before the app starts.

Manual Migration (if needed):

```
npm run migrate:deploy
```

📘 ADDITIONAL DOCUMENTATION

Innopay Documentation

- `../innopay/PROJECT_STATUS.md` - Detailed session notes (575+ lines, historical)
- `../innopay/FLOWS.md` - Payment flow documentation (reference)

Indiesmenu Documentation

- `./RESUME-TOMORROW.md` - Current status and next steps
- `./MIGRATION-SUMMARY.md` - Complete system status

Code Documentation

- Both Next.js projects have extensive inline comments with architectural decision dates
- Flow implementations include visual diagrams in comments
- API routes have JSDoc comments

🔑 KEY TAKEAWAYS

Architecture Benefits

1. **Centralized Security:** All sensitive operations (blockchain, Stripe) happen in hub
2. **Spoke Flexibility:** Each restaurant can use different tech stacks and UI/UX
3. **Reusable Infrastructure:** Hub APIs can be used by any spoke
4. **Easy Scaling:** Add new restaurants without touching existing code
5. **Maintainability:** Clear separation of concerns

Technical Decisions

1. **Next.js for Hub + Indiesmenu:** Server-side rendering, API routes, easy deployment

2. **Vite for Croque-Bedaine:** Faster builds, modern tooling, SPA architecture
3. **Prisma ORM:** Type-safe database access, easy migrations
4. **React Query:** Smart caching, automatic refetching, optimistic updates
5. **Tailwind CSS:** Utility-first styling, consistent across projects

Current Status (2026-01-02)

Hub (innopay):

- Production ready
- All payment flows working
- Debt tracking implemented
- Credential handover working

Merchant-Hub (merchant-hub):

- Core infrastructure complete
- Batched queries implemented ($O(1)$ scaling)
- Distributed leader election working
- Redis Streams integration complete
- Multi-environment support (prod + dev accounts)
- Vercel Cron fallback configured
- Co page integration pending (Indies & Croque)
- Transfer consumption logic needed
- Status/metrics endpoints planned

Spoke 1 (indiesmenu):

- Production ready
- All flows tested and working
- Balance refresh optimized
- React Query migration (Phases 1-3 complete)
- Optional optimizations remaining (Phases 4-5)
- Merchant-hub integration pending

Spoke 2 (croque-bedaine):

- In development
- Hub integration TBD
- Merchant-hub integration TBD
- Modern UI with shadcn/ui
- Vite build setup complete

TROUBLESHOOTING

Common Issues

Balance not updating after payment:

- Check `refetchBalance()` is being called

- Verify React Query DevTools shows fresh data
- Check console for `[useBalance]` logs

Hub not accessible from spoke:

- Verify `NEXT_PUBLIC_HUB_URL` environment variable
- Check hub is running on correct port
- Verify CORS settings if needed

Database migration errors:

- Run `npx prisma generate` after pulling new migrations
- Check PostgreSQL connection string
- Verify database exists and is accessible

Stripe webhook not working:

- Verify webhook secret matches environment variable
- Check webhook URL is correct in Stripe dashboard
- Test with Stripe CLI: `stripe listen --forward-to localhost:3000/api/webhooks`

Last Updated: 2026-01-02 **Maintainer:** Development Team **Questions:** Refer to individual project documentation or code comments

New in 2026-01-02:

- NEW Merchant-Hub: Centralized HAF polling infrastructure with O(1) scaling
- NEW Batched queries: 3 total queries regardless of restaurant count
- NEW Distributed leader election for polling coordination
- NEW Multi-environment support (prod + dev accounts polled simultaneously)