SPOKE INTEGRATION BUNDLE - Revised Implementation Plan

Last Updated: 2026-01-03 **Status**: APPROVED - Ready for Implementation **Authors**: Software Factory Planning (Opus 4.5 revision)

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Goal

Create a reusable "spoke integration bundle" that enables rapid integration of new merchant websites into the Innopay ecosystem. The bundle must support two tech stacks:

1. **Lovable Stack**: Vite + React + Supabase (croque-bedaine, future Lovable-built spokes)
2. **Next.js Stack**: Next.js + Prisma + Vercel Postgres (indiesmenu, custom-built spokes)

Key Decisions Made

Decision	Choice	Rationale
State Management	Custom useReducer state machine	Lightweight, portable, no dependencies
Backend (Supabase spokes)	Client-side sync via admin page	No serverless functions needed, browser tab is always open
Backend (Next.js spokes)	Direct API routes + Prisma	Copy from indiesmenu
Credential Storage	localStorage (current approach)	UX priority, low-value wallets, revisit later
Testing	4-level strategy with mocked blockchain	Unit → Integration → Staging → Smoke

Success Criteria

- New spoke integration: 3-5 days (not weeks)
- Zero payment flow bugs (tested code)
- Framework flexibility (Lovable OR Next.js)
- Clear documentation for each stack

ARCHITECTURE DECISIONS

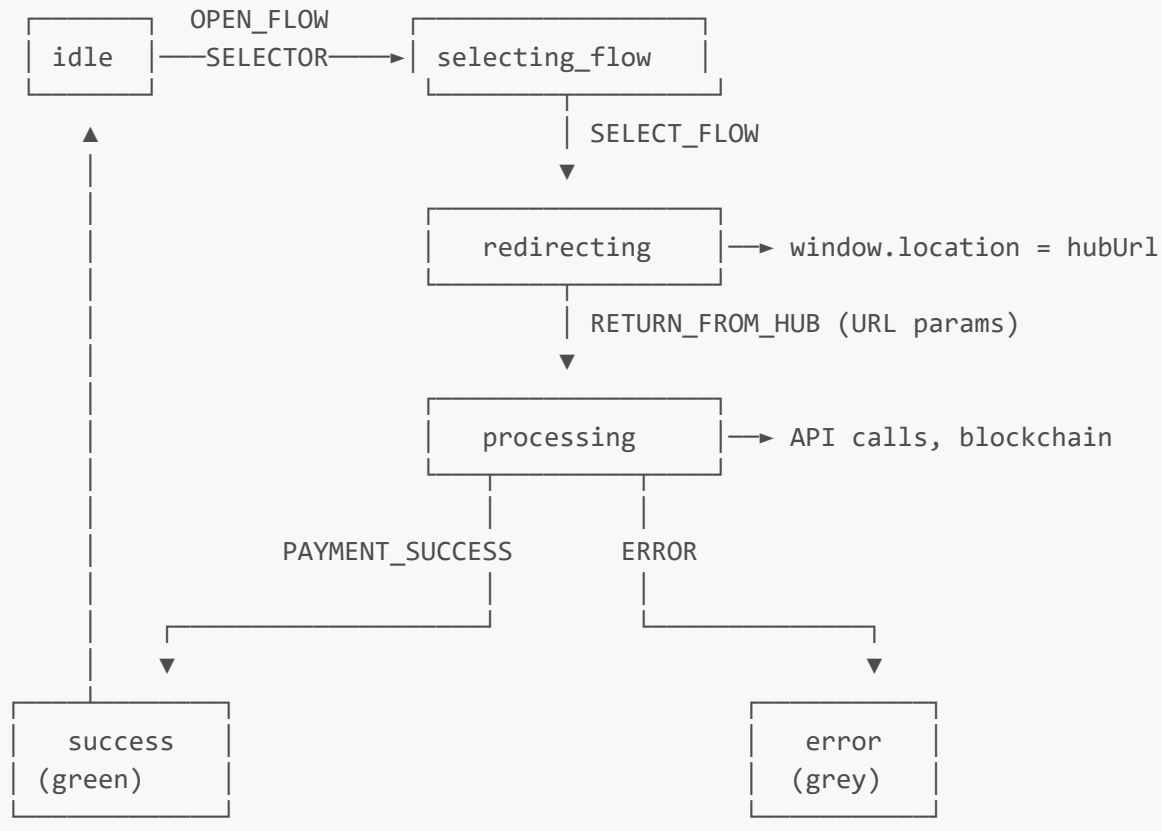
1. Payment Flow State Machine (useReducer)

Why: The current scattered `useState` calls in page.tsx create implicit states and impossible combinations. A state machine makes states explicit and transitions predictable.

Implementation: Custom reducer with TypeScript discriminated unions.

```
// States are mutually exclusive - no more "is yellow banner AND green banner
visible?"
type PaymentState =
  | { status: 'idle' }
  | { status: 'selecting_flow'; cartTotal: number }
  | { status: 'redirecting'; flow: FlowType; returnUrl: string }
  | { status: 'processing'; flow: FlowType; sessionId?: string }
  | { status: 'success'; flow: FlowType; orderId?: string }
  | { status: 'error'; flow: FlowType; error: string; canRetry: boolean }
  | { status: 'account_created'; credentials: Credentials };
```

Visual State Diagram:



```
        │                                                    │
        └──────────────────── RESET ─────────────────────────┘
```

Benefits:

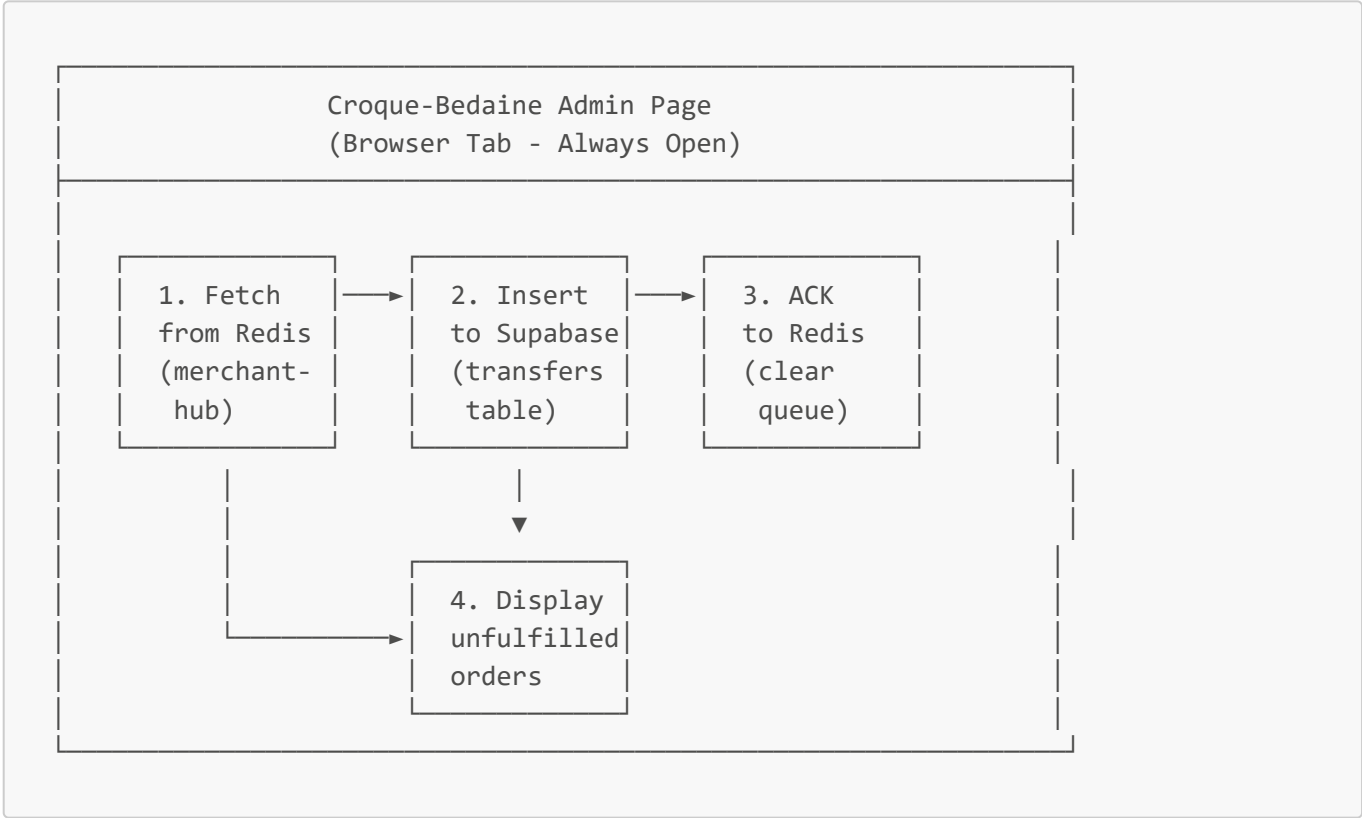
- Single source of truth for UI state
- Impossible states eliminated
- Every transition logged for debugging
- Pure function - easy to unit test
- Works identically in Next.js and Vite

---

2. Backend Architecture (Dual-Stack Support)

For Lovable/Supabase Spokes (Option 1: Client-Side Sync)

Architecture:



Data Flow:

1. **Wake-up**: Admin page calls `merchant-hub/api/wake-up` on mount
2. **Poll/Subscribe**: Every 6 seconds, fetch new transfers from Redis stream
3. **Sync**: Insert new transfers into Supabase `transfers` table
4. **ACK**: Acknowledge consumed messages to Redis
5. **Display**: Query Supabase for unfulfilled transfers, render list
6. **Fulfill**: Update `fulfilled_at` timestamp when order is served

No Serverless Functions Needed:

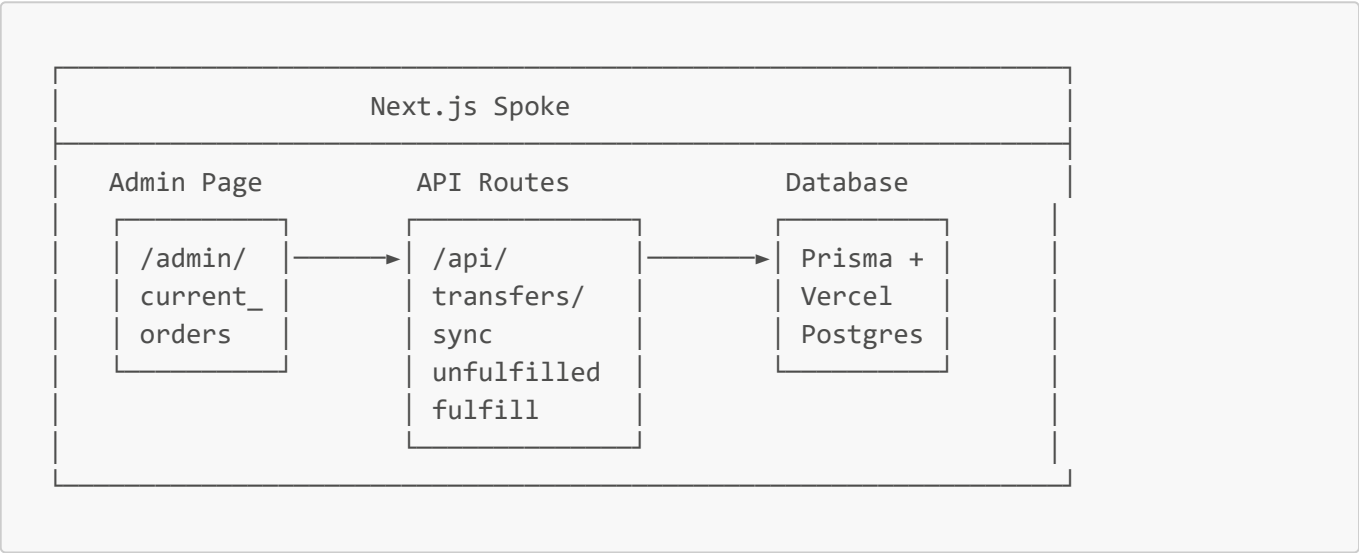
- All operations happen client-side
- Supabase JS client handles DB operations
- Redis HTTP API for merchant-hub communication
- Admin page must be open (which it always is during service)

Authentication:

- Supabase anonymous auth or simple password auth
- RLS policies allow all operations for authenticated users
- Risk is minimal (worst case: someone sees/modifies transfer records)

For Next.js/Prisma Spokes

Architecture: Same as indiesmenu - copy API routes directly.



Reuse: Copy these files from indiesmenu:

- `app/api/transfers/sync-from-merchant-hub/route.ts`
- `app/api/transfers/unfulfilled/route.ts`
- `app/api/fulfill/route.ts`
- `prisma/schema.prisma` (transfers model)

---

3. Credential Storage (Current Approach)

Decision: Keep localStorage for now. Revisit when:

- Wallet values increase significantly
- Security audit is conducted
- User complaints about credential loss

Current Pattern:

```
// After successful account creation (Flow 4, 5)
localStorage.setItem('accountName', credentials.accountName);
localStorage.setItem('masterPassword', credentials.masterPassword);
```

```
localStorage.setItem('activeKey', credentials.activeKey);
localStorage.setItem('postingKey', credentials.postingKey);
```

Risk Mitigation (Do Now):

- Add CSP headers to reduce XSS surface
- Avoid `dangerouslySetInnerHTML` with user input
- Sanitize all user-provided data before display
- Keep wallet balances low (encourage small top-ups)

Future Consideration (TODO):

- HttpOnly cookies for credential storage
- Hub-managed transaction signing
- Hardware wallet support (Hive Keychain)

BUNDLE COMPONENTS

Component Inventory

```
spoke-bundle/
├── components/
│   ├── Draggable.tsx              # Base draggable UI (139 lines)
│   ├── MiniWallet.tsx             # Wallet display, extends Draggable (130 lines)
│   └── BottomBanner.tsx           # Contact/legal footer (95 lines)
│
├── state/
│   ├── paymentStateMachine.ts     # State machine types + reducer (NEW)
│   └── paymentActions.ts          # Action creators + side effects (NEW)
│
├── hooks/
│   ├── usePaymentFlow.ts          # Main orchestration hook (NEW)
│   ├── useBalance.ts              # React Query balance fetching (100 lines)
│   └── useInnopayCart.ts          # Cart adapter for innopay (NEW, ~50 lines)
│
├── lib/
│   ├── utils.ts                   # Core utilities (dehydrateMemo, etc.)
│   ├── hive.ts                    # Hive blockchain operations
│   └── config.ts                  # Environment-aware URL resolution
│
├── context/
│   └── CartContext.tsx            # Reference implementation (301 lines)
│
├── database/
│   ├── prisma-schema.prisma       # For Next.js spokes
│   └── supabase-migration.sql     # For Lovable/Supabase spokes
│
└── admin/
    ├── CurrentOrdersPage.tsx      # Template for admin page
    └── supabase-sync.ts           # Client-side sync logic for Supabase
```

New Components to Create

1. Payment State Machine (`state/paymentStateMachine.ts`)

```
// Types
export type FlowType = 3 | 4 | 5 | 6 | 7;

export interface Credentials {
  accountName: string;
  masterPassword: string;
  activeKey: string;
  postingKey: string;
}

export type PaymentState =
  | { status: 'idle' }
  | { status: 'selecting_flow'; cartTotal: number; hasAccount: boolean }
  | { status: 'redirecting'; flow: FlowType; returnUrl: string }
  | { status: 'processing'; flow: FlowType; sessionId?: string; message?: string }
  | { status: 'success'; flow: FlowType; orderId?: string; message: string }
  | { status: 'error'; flow: FlowType; error: string; canRetry: boolean }
  | { status: 'account_created'; credentials: Credentials; flow: FlowType }
  | { status: 'waiter_called'; table: string };

export type PaymentEvent =
  | { type: 'OPEN_FLOW_SELECTOR'; cartTotal: number; hasAccount: boolean }
  | { type: 'SELECT_FLOW'; flow: FlowType }
  | { type: 'START_REDIRECT'; returnUrl: string }
  | { type: 'RETURN_FROM_HUB'; params: URLSearchParams }
  | { type: 'PROCESSING_UPDATE'; message: string }
  | { type: 'PAYMENT_SUCCESS'; orderId?: string; message: string }
  | { type: 'ACCOUNT_CREATED'; credentials: Credentials }
  | { type: 'WAITER_CALLED'; table: string }
  | { type: 'ERROR'; error: string; canRetry?: boolean }
  | { type: 'RETRY' }
  | { type: 'RESET' }
  | { type: 'DISMISS_BANNER' };

// Reducer
export function paymentReducer(state: PaymentState, event: PaymentEvent):
PaymentState {
  console.log('[PaymentStateMachine]', state.status, '→', event.type);

  switch (state.status) {
    case 'idle':
      if (event.type === 'OPEN_FLOW_SELECTOR') {
        return {
          status: 'selecting_flow',
          cartTotal: event.cartTotal,
          hasAccount: event.hasAccount
```

```
        };
      }
      if (event.type === 'RETURN_FROM_HUB') {
        // Handle page reload with URL params
        return { status: 'processing', flow: 3 }; // Detected from params
      }
      break;

    case 'selecting_flow':
      if (event.type === 'SELECT_FLOW') {
        return { status: 'redirecting', flow: event.flow, returnUrl: '' };
      }
      if (event.type === 'RESET' || event.type === 'DISMISS_BANNER') {
        return { status: 'idle' };
      }
      break;

    case 'redirecting':
      if (event.type === 'START_REDIRECT') {
        return { ...state, returnUrl: event.returnUrl };
      }
      // After redirect, browser will reload with new URL params
      // State will be reconstructed from URL in useEffect
      break;

    case 'processing':
      if (event.type === 'PROCESSING_UPDATE') {
        return { ...state, message: event.message };
      }
      if (event.type === 'PAYMENT_SUCCESS') {
        return {
          status: 'success',
          flow: state.flow,
          orderId: event.orderId,
          message: event.message
        };
      }
      if (event.type === 'ACCOUNT_CREATED') {
        return {
          status: 'account_created',
          credentials: event.credentials,
          flow: state.flow
        };
      }
      if (event.type === 'WAITER_CALLED') {
        return { status: 'waiter_called', table: event.table };
      }
      if (event.type === 'ERROR') {
        return {
          status: 'error',
          flow: state.flow,
          error: event.error,
          canRetry: event.canRetry ?? true
        };
```

```
      }
      break;

    case 'success':
    case 'account_created':
    case 'waiter_called':
      if (event.type === 'RESET' || event.type === 'DISMISS_BANNER') {
        return { status: 'idle' };
      }
      break;

    case 'error':
      if (event.type === 'RETRY' && state.canRetry) {
        return { status: 'selecting_flow', cartTotal: 0, hasAccount: false };
      }
      if (event.type === 'RESET' || event.type === 'DISMISS_BANNER') {
        return { status: 'idle' };
      }
      break;
  }

  console.warn('[PaymentStateMachine] Invalid transition:', state.status, '→',
event.type);
  return state;
}

export const initialPaymentState: PaymentState = { status: 'idle' };
```

2. Payment Flow Hook (`hooks/usePaymentFlow.ts`)

```
import { useReducer, useMemo, useEffect, useCallback } from 'react';
import { useSearchParams } from 'next/navigation'; // or 'react-router-dom' for
Vite
import { paymentReducer, initialPaymentState, PaymentState, FlowType, Credentials
} from '../state/paymentStateMachine';

interface UsePaymentFlowOptions {
  cartTotal: number;
  cartMemo: string;
  table: string;
  hubUrl: string;
  restaurantId: string;
  hiveAccount: string;
  onCartClear: () => void;
  onCredentialsReceived?: (credentials: Credentials) => void;
}

export function usePaymentFlow(options: UsePaymentFlowOptions) {
  const {
    cartTotal,
    cartMemo,
```

```
    table,
    hubUrl,
    restaurantId,
    hiveAccount,
    onCartClear,
    onCredentialsReceived,
  } = options;

  const [state, dispatch] = useReducer(paymentReducer, initialPaymentState);
  const searchParams = useSearchParams();

  // Check if user has an account
  const hasAccount = useMemo(() => {
    return !!localStorage.getItem('accountName');
  }, []);

  // Derive UI state from machine state
  const ui = useMemo(() => ({
    showFlowSelector: state.status === 'selecting_flow',
    showYellowBanner: state.status === 'processing',
    showGreenBanner: state.status === 'success' || state.status ===
'waiter_called',
    showGreyBanner: state.status === 'error',
    showMiniWallet: state.status === 'account_created' || hasAccount,
    isLoading: state.status === 'redirecting' || state.status === 'processing',
    bannerMessage: getBannerMessage(state),
    errorMessage: state.status === 'error' ? state.error : null,
    canRetry: state.status === 'error' && state.canRetry,
  }), [state, hasAccount]);

  // Handle return from hub (URL params)
  useEffect(() => {
    const orderSuccess = searchParams.get('order_success');
    const sessionId = searchParams.get('session_id');
    const credentialToken = searchParams.get('credential_token');
    const error = searchParams.get('error');

    if (orderSuccess === 'true' || sessionId || credentialToken) {
      dispatch({ type: 'RETURN_FROM_HUB', params: searchParams });
      processHubReturn(searchParams, dispatch, onCartClear, onCredentialsReceived,
hubUrl);
    } else if (error) {
      dispatch({ type: 'ERROR', error: decodeURIComponent(error) });
    }
  }, [searchParams]);

  // Actions
  const actions = useMemo(() => ({
    openFlowSelector: () => {
      dispatch({
        type: 'OPEN_FLOW_SELECTOR',
        cartTotal,
        hasAccount
      });
```

```
    },

    selectFlow: (flow: FlowType) => {
      dispatch({ type: 'SELECT_FLOW', flow });

      // Build hub URL based on flow
      const returnUrl = buildReturnUrl(flow, restaurantId);
      const hubRedirectUrl = buildHubUrl(flow, {
        hubUrl,
        restaurantId,
        hiveAccount,
        amount: cartTotal,
        memo: cartMemo,
        table,
        returnUrl,
      });

      dispatch({ type: 'START_REDIRECT', returnUrl: hubRedirectUrl });

      // Redirect to hub
      window.location.href = hubRedirectUrl;
    },

    callWaiter: async () => {
      dispatch({ type: 'SELECT_FLOW', flow: 6 });
      dispatch({ type: 'PROCESSING_UPDATE', message: 'Appel en cours...' });

      try {
        // Execute waiter call (small transfer with memo)
        await executeWaiterCall(table, hiveAccount, hubUrl);
        dispatch({ type: 'WAITER_CALLED', table });
      } catch (err) {
        dispatch({ type: 'ERROR', error: 'Échec de l\'appel serveur', canRetry:
true });
      }
    },

    retry: () => dispatch({ type: 'RETRY' }),
    reset: () => dispatch({ type: 'RESET' }),
    dismissBanner: () => dispatch({ type: 'DISMISS_BANNER' }),
  }), [cartTotal, cartMemo, table, hubUrl, restaurantId, hiveAccount,
hasAccount]);

  return { state, ui, actions, hasAccount };
}

// Helper functions
function getBannerMessage(state: PaymentState): string {
  switch (state.status) {
    case 'processing':
      return state.message || 'Traitement en cours...';
    case 'success':
      return state.message || 'Commande transmise avec succès!';
    case 'waiter_called':
```

```
        return `Un serveur arrive à la table ${state.table}`;
      case 'error':
        return state.error;
      default:
        return '';
    }
  }

  function buildReturnUrl(flow: FlowType, restaurantId: string): string {
    const base = window.location.origin;
    return `${base}/menu?flow=${flow}&restaurant=${restaurantId}`;
  }

  function buildHubUrl(flow: FlowType, params: {
    hubUrl: string;
    restaurantId: string;
    hiveAccount: string;
    amount: number;
    memo: string;
    table: string;
    returnUrl: string;
  }): string {
    const { hubUrl, restaurantId, hiveAccount, amount, memo, table, returnUrl } =
  params;

    const searchParams = new URLSearchParams({
      restaurant: restaurantId,
      recipient: hiveAccount,
      amount: amount.toFixed(2),
      memo: encodeURIComponent(memo),
      table,
      return_url: returnUrl,
    });

    switch (flow) {
      case 3: // Guest checkout
        return `${hubUrl}/checkout/guest?${searchParams}`;
      case 4: // Create account only
        return `${hubUrl}/user/create?${searchParams}&account_only=true`;
      case 5: // Create account + pay
        return `${hubUrl}/user/create?${searchParams}`;
      case 6: // Pay with account (handled locally, not redirect)
        return ''; // Should not redirect
      case 7: // Topup + pay
        return `${hubUrl}/checkout/topup?${searchParams}`;
      default:
        return hubUrl;
    }
  }

  async function processHubReturn(
    params: URLSearchParams,
    dispatch: React.Dispatch<any>,
    onCartClear: () => void,
```

```
  onCredentialsReceived?: (credentials: Credentials) => void,
  hubUrl?: string
) {
  dispatch({ type: 'PROCESSING_UPDATE', message: 'Finalisation de la commande...'
});

  try {
    const sessionId = params.get('session_id');
    const credentialToken = params.get('credential_token');
    const orderSuccess = params.get('order_success');

    // Handle credential token (Flows 4, 5)
    if (credentialToken && hubUrl) {
      const response = await fetch(`${hubUrl}/api/account/credentials`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ credentialToken }),
      });

      if (response.ok) {
        const { credentials } = await response.json();

        // Store credentials
        localStorage.setItem('accountName', credentials.accountName);
        localStorage.setItem('masterPassword', credentials.masterPassword);
        localStorage.setItem('activeKey', credentials.activeKey);
        localStorage.setItem('postingKey', credentials.postingKey);

        onCredentialsReceived?.(credentials);
        dispatch({ type: 'ACCOUNT_CREATED', credentials });
      }
    }

    // Handle payment success
    if (orderSuccess === 'true' || sessionId) {
      onCartClear();
      dispatch({
        type: 'PAYMENT_SUCCESS',
        orderId: sessionId || undefined,
        message: 'Votre commande a été transmise en cuisine!'
      });
    }

    // Clean URL params
    const cleanUrl = window.location.pathname;
    window.history.replaceState({}, '', cleanUrl);

  } catch (err) {
    console.error('[processHubReturn] Error:', err);
    dispatch({
      type: 'ERROR',
      error: 'Une erreur est survenue lors de la finalisation',
      canRetry: true
    });
```

```
  }
}

async function executeWaiterCall(
  table: string,
  hiveAccount: string,
  hubUrl: string
): Promise<void> {
  // Implementation depends on whether user has account
  const hasAccount = !!localStorage.getItem('accountName');

  if (hasAccount) {
    // Use Flow 6 pattern - sign and broadcast locally
    // ... (implementation from existing code)
  } else {
    // Redirect to hub for guest waiter call
    window.location.href = `${hubUrl}/waiter?
table=${table}&recipient=${hiveAccount}`;
  }
}
```

3. Innopay Cart Adapter (`hooks/useInnopayCart.ts`)

For Lovable/Supabase spokes that already have their own cart:

```
// This wraps an existing cart hook to add innopay-specific functionality

import { useMemo, useState, useEffect, useCallback } from 'react';

interface CartItem {
  id: string;
  name: string;
  price: number;
  quantity: number;
  options?: Record<string, string>;
}

interface ExistingCart {
  items: CartItem[];
  totalPrice: number;
  clearCart: () => void;
}

interface InnopayCartExtensions {
  table: string;
  setTable: (table: string) => void;
  getMemo: () => string;
  getTotalEurPrice: () => string;
}

export function useInnopayCart(existingCart: ExistingCart): ExistingCart &
```

```
InnopayCartExtensions {
  // Table tracking (innopay requirement)
  const [table, setTableState] = useState(() => {
    if (typeof window === 'undefined') return '';

    // Check URL first (QR code source of truth)
    const urlParams = new URLSearchParams(window.location.search);
    const urlTable = urlParams.get('table');
    if (urlTable) {
      localStorage.setItem('innopay_table', urlTable);
      return urlTable;
    }

    // Fall back to localStorage
    return localStorage.getItem('innopay_table') || '';
  });

  const setTable = useCallback((newTable: string) => {
    setTableState(newTable);
    localStorage.setItem('innopay_table', newTable);
  }, []);

  // Generate memo for blockchain transfer
  const getMemo = useCallback(() => {
    const itemStrings = existingCart.items.map(item => {
      let line = `${item.quantity}x${item.name}`;
      if (item.options) {
        const optStr = Object.values(item.options).join(',');
        if (optStr) line += `(${optStr})`;
      }
      line += `@${item.price.toFixed(2)}`;
      return line;
    });

    const itemsMemo = itemStrings.join(';');
    const tableMemo = table ? ` TABLE ${table}` : '';

    return `${itemsMemo};${tableMemo}`;
  }, [existingCart.items, table]);

  // EUR price (innopay uses EUR)
  const getTotalEurPrice = useCallback(() => {
    return existingCart.totalPrice.toFixed(2);
  }, [existingCart.totalPrice]);

  return {
    ...existingCart,
    table,
    setTable,
    getMemo,
    getTotalEurPrice,
  };
}
```

4. Supabase Sync Logic (`admin/supabase-sync.ts`)

For Lovable/Supabase spokes. Uses the actual merchant-hub API endpoints:

Merchant-Hub API Reference:

- `POST /api/wake-up` - Inform hub that a co page is open, may become poller
- `GET /api/transfers/consume?restaurantId=X` - Consume transfers from Redis stream (returns `messageId` for each)
- `POST /api/transfers/ack` - Acknowledge messages after successful DB insert
- `POST /api/poll` - Trigger HAF polling (only if this page is the elected poller)
- `POST /api/heartbeat` - Maintain poller heartbeat (every 5s if poller)

```
import { createClient, SupabaseClient } from '@supabase/supabase-js';

interface Transfer {
  messageId: string;      // Redis stream message ID (needed for ACK)
  id: string;             // HAF operation ID
  from_account: string;
  amount: string;
  symbol: 'HBD' | 'EURO' | 'OCLT';
  memo: string;
  parsed_memo?: string;
  received_at: string;
  fulfilled_at?: string;
}

interface SyncResult {
  consumed: number;
  inserted: number;
  acked: number;
  errors: string[];
}

export class SupabaseTransferSync {
  private supabase: SupabaseClient;
  private merchantHubUrl: string;
  private restaurantId: string;
  private consumerId: string;
  private isPoller: boolean = false;
  private heartbeatInterval: number | null = null;

  constructor(
    supabaseUrl: string,
    supabaseAnonKey: string,
    merchantHubUrl: string,
    restaurantId: string
  ) {
    this.supabase = createClient(supabaseUrl, supabaseAnonKey);
    this.merchantHubUrl = merchantHubUrl;
    this.restaurantId = restaurantId;
```

```
    // Unique consumer ID for this browser tab
    this.consumerId =
`${restaurantId}-${Date.now()}-${Math.random().toString(36).slice(2)}`;
  }

  // Call on page mount
  async wakeUp(): Promise<{ shouldStartPolling: boolean; poller: string | null }>
{
    const response = await fetch(`${this.merchantHubUrl}/api/wake-up`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ shopId: this.restaurantId }),
    });

    const data = await response.json();
    this.isPoller = data.shouldStartPolling;

    if (this.isPoller) {
      // Start heartbeat to maintain poller status
      this.startHeartbeat();
    }

    return {
      shouldStartPolling: data.shouldStartPolling,
      poller: data.poller,
    };
  }

  // Maintain heartbeat if we're the poller (every 5 seconds)
  private startHeartbeat() {
    if (this.heartbeatInterval) return;

    this.heartbeatInterval = window.setInterval(async () => {
      try {
        await fetch(`${this.merchantHubUrl}/api/heartbeat`, {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({ shopId: this.restaurantId }),
        });
      } catch (err) {
        console.error('[Heartbeat] Failed:', err);
      }
    }, 5000);
  }

  // Stop heartbeat (call on page unload)
  stopHeartbeat() {
    if (this.heartbeatInterval) {
      clearInterval(this.heartbeatInterval);
      this.heartbeatInterval = null;
    }
  }

  // Trigger HAF polling (only call if we're the poller)
```

```
  async triggerPoll(): Promise<void> {
    if (!this.isPoller) return;

    try {
      await fetch(`${this.merchantHubUrl}/api/poll`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ shopId: this.restaurantId }),
      });
    } catch (err) {
      console.error('[Poll] Failed:', err);
    }
  }

  // Call every 6 seconds - consume from Redis, insert to Supabase, ACK
  async sync(): Promise<SyncResult> {
    const result: SyncResult = { consumed: 0, inserted: 0, acked: 0, errors: [] };

    try {
      // 1. Consume transfers from merchant-hub Redis stream
      const consumeUrl = new URL(`${this.merchantHubUrl}/api/transfers/consume`);
      consumeUrl.searchParams.set('restaurantId', this.restaurantId);
      consumeUrl.searchParams.set('consumerId', this.consumerId);
      consumeUrl.searchParams.set('count', '20');

      const consumeResponse = await fetch(consumeUrl.toString());

      if (!consumeResponse.ok) {
        result.errors.push(`Consume failed: ${consumeResponse.status}`);
        return result;
      }

      const { transfers, pending } = await consumeResponse.json();
      result.consumed = transfers.length;

      if (transfers.length === 0) {
        // No new transfers, but log if there are pending (unacked) ones
        if (pending > 0) {
          console.log(`[Sync] No new transfers, but ${pending} pending
(unacked)`);
        }
        return result;
      }

      console.log(`[Sync] Consumed ${transfers.length} transfers from Redis`);

      // 2. Insert into Supabase (upsert to handle duplicates)
      const { error: insertError } = await this.supabase
        .from('transfers')
        .upsert(
          transfers.map((t: Transfer) => ({
            id: t.id,
            from_account: t.from_account,
            amount: t.amount,
```

```
            symbol: t.symbol,
            memo: t.memo,
            parsed_memo: t.parsed_memo ? JSON.parse(t.parsed_memo) : null,
            received_at: t.received_at,
            // fulfilled_at intentionally omitted (null = unfulfilled)
          })),
          { onConflict: 'id' }
        );

      if (insertError) {
        result.errors.push(`Insert failed: ${insertError.message}`);
        // Don't ACK if insert failed - messages will be redelivered
        return result;
      }

      result.inserted = transfers.length;
      console.log(`[Sync] Inserted ${transfers.length} transfers to Supabase`);

      // 3. ACK to Redis (only after successful insert!)
      const messageIds = transfers.map((t: Transfer) => t.messageId);

      const ackResponse = await fetch(`${this.merchantHubUrl}/api/transfers/ack`,
{
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          restaurantId: this.restaurantId,
          messageIds,
        }),
      });

      if (ackResponse.ok) {
        const ackData = await ackResponse.json();
        result.acked = ackData.acknowledged;
        console.log(`[Sync] ACKed ${result.acked} messages`);
      } else {
        result.errors.push(`ACK failed: ${ackResponse.status}`);
      }

    } catch (err) {
      result.errors.push(`Sync error: ${err}`);
    }

    return result;
  }

  // Get unfulfilled transfers from Supabase (for display)
  async getUnfulfilled(): Promise<Transfer[]> {
    const { data, error } = await this.supabase
      .from('transfers')
      .select('*')
      .is('fulfilled_at', null)
      .order('received_at', { ascending: false });
```

```
    if (error) {
      console.error('[SupabaseTransferSync] getUnfulfilled error:', error);
      return [];
    }

    return data || [];
  }

  // Mark transfer as fulfilled (order served)
  async fulfill(transferId: string): Promise<boolean> {
    const { error } = await this.supabase
      .from('transfers')
      .update({ fulfilled_at: new Date().toISOString() })
      .eq('id', transferId);

    return !error;
  }
}
```

IMPLEMENTATION PLAN FOR CROQUE-BEDAINE

> **IMPORTANT: Additive Integration**
>
> This integration is **additive only**. It will NOT break or replace any existing croque-bedaine functionality:
>
> - Existing menu browsing works unchanged
> - Existing cart functionality stays intact (we wrap it, not replace it)
> - Existing Supabase tables (categories, dishes, drinks, etc.) are untouched
> - Existing i18n system is preserved and reused
> - We're adding new files/folders, not modifying existing ones (except for integration points)

Phase 1: Setup & UI Components (Day 1)

Tasks:

1. **Create innopay folder structure:**

```
croque-bedaine/src/
├── lib/
│   └── innopay/
│       ├── config.ts
│       ├── utils.ts
│       └── hive.ts
├── state/
│   └── innopay/
│       └── paymentStateMachine.ts
├── hooks/
│   └── innopay/
│       ├── usePaymentFlow.ts
│       ├── useInnopayCart.ts
```

```
│       └── useBalance.ts
└── components/
    └── innopay/
        ├── Draggable.tsx
        ├── MiniWallet.tsx
        ├── BottomBanner.tsx
        ├── FlowSelector.tsx
        └── StatusBanners.tsx
```

2. **Copy UI components from indiesmenu:**
   - `Draggable.tsx` - copy as-is
   - `MiniWallet.tsx` - copy as-is
   - `BottomBanner.tsx` - copy as-is
3. **Copy utility functions:**
   - `dehydrateMemo`, `hydrateMemo`
   - `generateHiveTransferUrl`, `generateDistriatedHiveOp`
   - `createEuroTransferOperation`
   - Environment URL resolution functions
4. **Adapt for Vite:**
   - Change `'use client'` directives (not needed in Vite)
   - Change `next/navigation` imports to `react-router-dom`
   - Update environment variable access (`import.meta.env.VITE_*`)

Deliverables:

- ☐ UI components rendering without errors
- ☐ Dragging works on desktop and mobile
- ☐ Environment variables configured

Phase 2: State Machine & Payment Flows (Day 2)

Tasks:

1. **Implement state machine:**
   - Create `paymentStateMachine.ts`
   - Write unit tests for reducer transitions
2. **Implement usePaymentFlow hook:**
   - Integrate with existing cart
   - Handle URL param returns from hub
   - Implement flow selection logic
3. **Create useInnopayCart adapter:**

- Wrap croque-bedaine's existing `useCart`
- Add table tracking
- Add memo generation

4. **Integrate into cart/checkout page:**
   - Add flow selector UI (Draggable component)
   - Add status banners (yellow, grey, green)
   - Wire up "Order Now" and "Call Waiter" buttons

Deliverables:

- [ ] State machine unit tests passing
- [ ] Flow selector appears when clicking "Order Now"
- [ ] Redirect to hub works for Flows 3, 4, 5, 7
- [ ] Return from hub updates UI correctly

Phase 3: Balance & MiniWallet (Day 3)

Tasks:

1. **Implement useBalance hook:**
   - Copy from indiesmenu
   - Configure React Query caching
2. **Integrate MiniWallet:**
   - Show after account creation (Flow 4, 5)
   - Display balance
   - Implement minimize/restore
3. **Implement Flow 6 (Pay with Account):**
   - Check balance sufficiency
   - Execute dual-currency transfer
   - Handle success/error states
4. **Implement account import:**
   - UI for importing existing account
   - Validate credentials
   - Store in localStorage

Deliverables:

- [ ] MiniWallet appears after account creation
- [ ] Balance displays correctly
- [ ] Flow 6 payment works end-to-end
- [ ] Account import works

Phase 4: Database & Admin Page (Day 4)

Note: Croque-bedaine already has a Supabase schema with tables for categories, dishes, drinks, ingredients, allergens, etc. We only need to **verify/add** the `transfers` table - this is **additive**, not replacing anything.

Tasks:

1. **Verify/create transfers table in Supabase:**

   First, check if the `transfers` table exists. If not, create it:

```
-- Run in Supabase SQL editor (only if transfers table doesn't exist)
CREATE TABLE IF NOT EXISTS transfers (
  id VARCHAR PRIMARY KEY,
  from_account VARCHAR NOT NULL,
  amount VARCHAR NOT NULL,
  symbol VARCHAR NOT NULL CHECK (symbol IN ('HBD', 'EURO', 'OCLT')),
  memo TEXT NOT NULL,
  parsed_memo JSONB,
  received_at TIMESTAMP WITH TIME ZONE NOT NULL,
  fulfilled_at TIMESTAMP WITH TIME ZONE,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Index for efficient unfulfilled queries
CREATE INDEX IF NOT EXISTS idx_transfers_unfulfilled
  ON transfers(received_at DESC)
  WHERE fulfilled_at IS NULL;

-- RLS policies (permissive for admin operations)
ALTER TABLE transfers ENABLE ROW LEVEL SECURITY;

-- Allow all operations (transfers are public on blockchain anyway)
CREATE POLICY "Allow all operations on transfers" ON transfers
  FOR ALL USING (true) WITH CHECK (true);
```

2. **Implement SupabaseTransferSync class:**
   - Copy from bundle (see code above)
   - Configure with environment variables
   - Handles: wake-up, heartbeat, poll trigger, consume, insert, ACK

3. **Create admin/current_orders page:**
   - Add React Router route: `/admin/current-orders`
   - On mount: call `sync.wakeUp()`
   - If poller: start 6-second poll trigger interval
   - Every 6 seconds: call `sync.sync()` then `sync.getUnfulfilled()`
   - Display unfulfilled orders with hydrated memo
   - "C'est parti!" button calls `sync.fulfill(id)`
   - On unmount: call `sync.stopHeartbeat()`

4. **Test sync flow:**
    - Place test order via innopay hub (use staging/test accounts)
    - Verify transfer appears in admin page
    - Verify fulfill button works
    - Verify fulfilled orders disappear from list

Deliverables:

- [ ] Transfers table exists in Supabase
- [ ] Admin page displays unfulfilled transfers
- [ ] Sync from merchant-hub (consume → insert → ACK) works
- [ ] Fulfill updates `fulfilled_at` timestamp
- [ ] Poller election works (if multiple tabs open)

Phase 5: Testing & Polish (Day 5)

Tasks:

1. **End-to-end testing:**
    - Test all flows (3, 4, 5, 6, 7)
    - Test on mobile (iOS Safari, Android Chrome)
    - Test error scenarios
2. **UI polish:**
    - Match croque-bedaine's design language
    - Ensure smooth animations
    - Test dark mode (if applicable)
3. **Documentation:**
    - Update croque-bedaine README
    - Document environment variables
    - Create testing checklist
4. **Deployment:**
    - Deploy to staging
    - Test with real Stripe test mode
    - Verify merchant-hub integration

Deliverables:

- [ ] All flows tested and working
- [ ] Mobile UX smooth
- [ ] Staging deployment successful
- [ ] Documentation updated

TESTING STRATEGY

Level 1: Unit Tests (No Blockchain)

What to test:

- State machine transitions (paymentReducer)
- Memo generation/parsing (dehydrateMemo, hydrateMemo)
- Price calculations
- URL building functions

Setup:

```
# For Vite projects
npm install -D vitest @testing-library/react @testing-library/jest-dom
```

Example test file:

```
// src/state/innopay/__tests__/paymentStateMachine.test.ts
import { describe, it, expect } from 'vitest';
import { paymentReducer, initialPaymentState } from '../paymentStateMachine';

describe('paymentReducer', () => {
  it('should start in idle state', () => {
    expect(initialPaymentState.status).toBe('idle');
  });

  it('should transition from idle to selecting_flow', () => {
    const state = paymentReducer(
      initialPaymentState,
      { type: 'OPEN_FLOW_SELECTOR', cartTotal: 50, hasAccount: false }
    );
    expect(state.status).toBe('selecting_flow');
    expect(state.cartTotal).toBe(50);
  });

  it('should not allow invalid transitions', () => {
    const state = paymentReducer(
      initialPaymentState,
      { type: 'PAYMENT_SUCCESS', message: 'test' }
    );
    expect(state.status).toBe('idle'); // Unchanged
  });

  it('should transition to error state', () => {
    const processingState = { status: 'processing' as const, flow: 3 as const };
    const state = paymentReducer(
      processingState,
      { type: 'ERROR', error: 'Payment failed', canRetry: true }
    );
```

```
    expect(state.status).toBe('error');
    expect(state.error).toBe('Payment failed');
  });
});
```

Level 2: Integration Tests (Mocked APIs)

What to test:

- usePaymentFlow hook behavior
- API call sequences
- UI state derivation

Setup:

```
npm install -D msw
```

Example:

```
// src/hooks/innopay/__tests__/usePaymentFlow.test.ts
import { renderHook, act, waitFor } from '@testing-library/react';
import { rest } from 'msw';
import { setupServer } from 'msw/node';
import { usePaymentFlow } from '../usePaymentFlow';

const server = setupServer(
  rest.post('https://wallet.innopay.lu/api/account/credentials', (req, res, ctx)
=> {
    return res(ctx.json({
      credentials: {
        accountName: 'test.user',
        masterPassword: 'test-password',
        activeKey: '5K...',
        postingKey: '5J...',
      }
    }));
  })
);

beforeAll(() => server.listen());
afterEach(() => server.resetHandlers());
afterAll(() => server.close());

describe('usePaymentFlow', () => {
  const defaultOptions = {
    cartTotal: 25.00,
    cartMemo: '1xSteak@25.00; TABLE 5',
    table: '5',
    hubUrl: 'https://wallet.innopay.lu',
```

```
    restaurantId: 'croque',
    hiveAccount: 'croque.bedaine',
    onCartClear: vi.fn(),
  };

  it('should start in idle state', () => {
    const { result } = renderHook(() => usePaymentFlow(defaultOptions));
    expect(result.current.state.status).toBe('idle');
    expect(result.current.ui.showFlowSelector).toBe(false);
  });

  it('should open flow selector', () => {
    const { result } = renderHook(() => usePaymentFlow(defaultOptions));

    act(() => {
      result.current.actions.openFlowSelector();
    });

    expect(result.current.state.status).toBe('selecting_flow');
    expect(result.current.ui.showFlowSelector).toBe(true);
  });
});
```

Level 3: Staging Environment

Configuration:

```
# .env.staging
VITE_HUB_URL=https://staging.wallet.innopay.lu
VITE_MERCHANT_HUB_URL=https://staging.merchant-hub.vercel.app
VITE_HIVE_ACCOUNT=croque-test
VITE_SUPABASE_URL=https://xxx.supabase.co
VITE_SUPABASE_ANON_KEY=xxx

# In innopay staging
RECIPIENT_OVERRIDE=croque-test
```

Test checklist:

- ☐ Flow 3: Guest checkout with Stripe test card
- ☐ Flow 4: Create account only
- ☐ Flow 5: Create account + pay
- ☐ Flow 6: Pay with existing account
- ☐ Flow 7: Top-up + pay
- ☐ Admin page: Orders appear
- ☐ Admin page: Fulfill works
- ☐ Mobile: iOS Safari
- ☐ Mobile: Android Chrome

Level 4: Production Smoke Tests

Playwright setup:

```
npm install -D @playwright/test
```

Smoke test:

```
// e2e/smoke.test.ts
import { test, expect } from '@playwright/test';

test('menu page loads', async ({ page }) => {
  await page.goto('https://croque-bedaine.lu/menu?table=1');
  await expect(page.locator('h1')).toContainText('Menu');
});

test('cart functions', async ({ page }) => {
  await page.goto('https://croque-bedaine.lu/menu?table=1');

  // Add item to cart
  await page.click('[data-testid="add-item-croque-monsieur"]');

  // Verify cart updated
  await expect(page.locator('[data-testid="cart-count"]')).toHaveText('1');
});

// DO NOT test real payments in production smoke tests
```

FUTURE SPOKE GUIDELINES

For Lovable-Built Spokes (Vite + Supabase)

1. **Start from croque-bedaine as template**
2. **Copy entire `src/lib/innopay/`, `src/hooks/innopay/`, `src/components/innopay/` folders**
3. **Create Supabase `transfers` table using provided SQL**
4. **Configure environment variables:**

   ```
   VITE_HUB_URL=https://wallet.innopay.lu
   VITE_MERCHANT_HUB_URL=https://merchant-hub-theta.vercel.app
   VITE_HIVE_ACCOUNT=your-restaurant.account
   VITE_RESTAURANT_ID=your-restaurant
   ```

5. **Adapt cart integration (useInnopayCart wrapper)**
6. **Test all flows**

Estimated time: 2-3 days

For Next.js + Prisma Spokes

1. **Start from indiesmenu as template**
2. **Copy API routes, hooks, components**
3. **Copy Prisma schema (transfers model)**
4. **Run migrations**
5. **Configure environment variables**
6. **Test all flows**

Estimated time: 1-2 days

Restaurant-Specific Customizations

Customization	Where	Effort
Branding/colors	Tailwind config, component props	Low
Menu structure	Cart context, memo generation	Medium
Table numbering	URL params, localStorage key	Low
Languages	i18n translations	Medium
Special flows	Payment hook options	Medium-High

TODO: DEFERRED ITEMS

Security Review (Future)

- [ ] Evaluate HttpOnly cookies for credential storage
- [ ] Consider hub-managed transaction signing
- [ ] Audit XSS vectors in all spokes
- [ ] Implement CSP headers
- [ ] Add rate limiting for sensitive operations

Trigger: When wallet balances consistently exceed 100 EUR or security incident occurs.

Hive Keychain / External Wallet Support (Already Present)

Status: External wallet support via `hive://sign/op/` URI scheme is already implemented in indiesmenu.

Known Issue: FreeNow URI Collision

The `hive://` URI scheme is not officially registered. FreeNow (a taxi app popular in Eastern Europe/Middle East) has also registered this scheme. On devices with both apps installed, FreeNow may intercept Hive wallet URLs.

Workarounds to implement:

- [ ] Don't auto-open `hive://` URLs - show a choice modal first

- [ ] Add "FreeNow opened instead?" recovery flow with QR code fallback
- [ ] On Android, use intent URLs with explicit package name: `intent://sign/op/...#Intent;scheme=hive;package=com.aspect.keychain;end`
- [ ] Detect Hive Keychain browser extension (`window.hive_keychain`) and use its API directly on desktop
- [ ] Make Innopay in-browser flow the default, external wallet as opt-in

Automated Test Runs (Next Month)

Note: CI/CD for deployment is already handled by Vercel. This TODO refers to automated *test execution* on pull requests.

- [ ] Configure Vitest to run on Vercel build (or GitHub Actions)
- [ ] Add unit test step to PR checks
- [ ] Create test data seeding scripts for staging
- [ ] Document RECIPIENT_OVERRIDE pattern for local/staging testing

Bundle Extraction (After 3-4 Spokes)

- [ ] Extract `@innopay/ui-components` npm package
- [ ] Extract `@innopay/payment-hooks` npm package
- [ ] Consider standalone payment service (Option C)
- [ ] Evaluate based on real integration experience

i18n Bundle Integration (Next Sprint)

Note: i18n is already implemented in croque-bedaine. This TODO refers to extracting it to the reusable bundle.

- [ ] Extract croque-bedaine's i18n system to `spoke-bundle/i18n/`
- [ ] Create shared innopay translation keys (wallet, payment flows, banners)
- [ ] Add i18n to indiesmenu (currently French-only)
- [ ] Document i18n integration for future spokes

APPENDIX: ENVIRONMENT VARIABLES

Croque-Bedaine (Vite + Supabase)

```
# .env.local

# Innopay Hub
VITE_HUB_URL=https://wallet.innopay.lu

# Merchant Hub (blockchain polling)
VITE_MERCHANT_HUB_URL=https://merchant-hub-theta.vercel.app

# Restaurant Identity
VITE_HIVE_ACCOUNT=croque.bedaine
VITE_RESTAURANT_ID=croque-bedaine
```

```
# Supabase
VITE_SUPABASE_URL=https://xxx.supabase.co
VITE_SUPABASE_ANON_KEY=xxx

# Development only
VITE_RECIPIENT_OVERRIDE=croque-test  # Remove in production!
```

Indiesmenu (Next.js + Prisma)

```
# .env.local

# Innopay Hub
NEXT_PUBLIC_HUB_URL=https://wallet.innopay.lu

# Merchant Hub
NEXT_PUBLIC_MERCHANT_HUB_URL=https://merchant-hub-theta.vercel.app

# Restaurant Identity
NEXT_PUBLIC_HIVE_ACCOUNT=indies.cafe
NEXT_PUBLIC_RESTAURANT_ID=indies

# Database
DATABASE_URL=postgresql://...
POSTGRES_URL=postgresql://...

# Development only
RECIPIENT_OVERRIDE=indies-test  # Remove in production!
```

APPENDIX: GENERATING A PDF

To print this document, convert it to PDF using one of these methods:

Option 1: VS Code Extension (Easiest)

1. Install the "Markdown PDF" extension in VS Code (by yzane)
2. Open this file in VS Code
3. Press `Ctrl+Shift+P` → type "Markdown PDF: Export (pdf)"
4. PDF will be created in the same folder

Option 2: Pandoc (Best Quality)

```
# Install Pandoc first (https://pandoc.org/installing.html)
# On Windows with Chocolatey:
choco install pandoc

# Then convert:
```

```
pandoc SPOKE-INTEGRATION-PLAN-REVISED.md -o SPOKE-INTEGRATION-PLAN-REVISED.pdf --
pdf-engine=xelatex
```

Option 3: Online Converters

- StackEdit - Paste markdown, export to PDF
- Dillinger - Paste markdown, export to PDF
- MD to PDF - Upload file, download PDF

Option 4: GitHub (If committed)

1. Commit this file to GitHub
2. View it on GitHub (renders as formatted markdown)
3. Print the page (Ctrl+P) and select "Save as PDF"

Document Status: APPROVED - Ready for Implementation **Next Action**: Begin Phase 1 for croque-bedaine **Review Date**: After croque-bedaine integration complete