REALNODE PROTOCOL

B2B Integration Guide — React & Next.js (NPM)

Technical deployment manual for modern JavaScript environments.

Component Architecture (React)
This document details the integration of the RealNode protocol via our official NPM package. Built specifically for React and Next.js, this package provides ready-to-use components (checkout buttons, quota badges, device managers) to secure your transaction funnel without the need to design complex interfaces.

1 Step 0: Account Creation and API Keys Retrieval

The very first step is to create an account on the RealNode portal and retrieve your access keys. These keys securely identify your application to our infrastructure.

1. **Registration:** Visit the admin portal (`https://app.realnode.emkaylabs.tech/signup`) to create your enterprise account. You will need to provide your company name, a professional email address, and an administrator password.
2. **Retrieving Keys:** The portal generates a unique key pair for your environment:
   - **Public Key** (format: `pk_live_...`): To be used in your frontend React code. It is entirely safe to expose this key within the browser.
   - **Secret Key** (format: `sk_live_...`): To be used strictly on your backend server to cryptographically validate transactions. Never share it or bundle it within your frontend code.

2 Step 1: Installing the NPM Package

Open your terminal at the root of your React or Next.js project and run the following command:

```
npm install @emkaylabs/realnode-sdk
```

What does this command do? It downloads and installs the `@emkaylabs/realnode-sdk` package into your project. This package includes:

- The core HTTP client that communicates with the RealNode API (`/verify`, `/consume`, `/check-session`, etc.).
- Production-ready React components (`RealNodeProvider`, `RealNodeCheckoutButton`, `RealNodeDeviceMan` `RealNodeQuotaBadge`).
- Session management and secure local storage utilities (device token lifecycle).

Once installed, the package will be available in your `node_modules` directory and referenced in your `package.json`.

3 Step 2: Global Configuration — The Provider

The `RealNodeProvider` component acts as the central hub. It initializes the security engine and exposes the SDK context to all nested components. It must be mounted **exactly once**, as high up in your component tree as possible.

Where to place this code? Inside your root entry file: `App.jsx` for Create React App, or `_app.jsx` / `layout.jsx` for Next.js.

```
import { RealNodeProvider } from '@emkaylabs/realnode-sdk/react';

export default function App({ children }) {
  return (
    <RealNodeProvider apiKey="pk_live_YOUR_PUBLIC_KEY">
      {children}
    </RealNodeProvider>
  );
}
```

Under the hood initialization sequence:

1. The Provider automatically issues a `GET /sdk/config?api_key=pk_live_...` request to our edge servers.
2. The servers return the active configuration mapped to your subscription plan: the protection tier and enabled features.
3. Based on the payload, the SDK dynamically mounts the appropriate modules without requiring any code changes on your end:
   - **Insight:** Passive behavioral analysis only. No visible UI interference.
   - **Sentinel:** The full kinetic behavioral matrix is activated in the background.
   - **Vault:** The kinetic matrix and the biometric modal (FaceID, TouchID, Windows Hello) are activated. The modal is automatically injected into the DOM.
4. The SDK silently attempts to restore an existing session (if the user has already completed a verification earlier during the same event window) to prevent redundant biometric prompts.

Public Key Safety
Your public key (`pk_live_...`) can safely reside in your public React bundle. It grants zero access to administrative actions and strictly serves to identify your property to our verification API. Conversely, your Secret Key (`sk_live_...`) must remain entirely isolated on your backend infrastructure.

4 Step 3: Securing the Checkout Button

This is the only necessary modification within your checkout flow. Simply replace your standard HTML purchase button with the `RealNodeCheckoutButton` component.

```
import { RealNodeCheckoutButton } from '@emkaylabs/realnode-sdk/react';

export default function CheckoutPage() {

  const handleSuccess = (proof) => {
    // proof contains: { idh, device_hash, device_token, quantity, status
       }
    // Transmit this payload to your backend to finalize the order.
```

```
    fetch('/api/your-checkout-endpoint', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(proof)
    });
  };

  const handleDenied = () => {
    alert("Access Denied: Suspicious automation detected.");
  };

  return (
    <RealNodeCheckoutButton
      quantity={2}
      onSuccess={handleSuccess}
      onDenied={handleDenied}
      className="your-existing-css-class"
    >
      Confirm Order
    </RealNodeCheckoutButton>
  );
}
```

Execution flow upon user click:

1. The click event is intercepted. The transaction is temporarily halted.
2. Depending on the active plan detected by the Provider:
   - **Insight:** A silent background verification runs. The user notices nothing.
   - **Sentinel:** The behavioral score is computed and verified. The user notices nothing.
   - **Vault:** The biometric modal takes focus. The user verifies their identity via FaceID, TouchID, or Passkey. The hardware operation resolves in under 1 second.
3. Upon successful attestation, `onSuccess` fires, injecting the cryptographic `proof`.
4. If the attestation fails or a bot signature is detected, `onDenied` triggers and the transaction sequence is aborted.

Important: Biometrics are requested only once per session.
The SDK preserves the attestation proof in memory. If a user has successfully cleared a biometric challenge (e.g., during a prior purchase), they will not be prompted again for subsequent transactions on the same page, provided the session remains valid.

5 Step 4: User Interface Components (Recommended)

RealNode supplies pre-built, styleable components that seamlessly blend into your existing UI, requiring zero wireframing or CSS boilerplate.

5.1 4.1. Real-Time Quota Display

Available across all plans. This badge displays the remaining ticket inventory available for the current user session. The numerical value is automatically synchronized with our edge servers via a persistent Server-Sent Events (SSE) connection.

```
import { RealNodeQuotaBadge } from '@emkaylabs/realnode-sdk/react';

// Place this component adjacent to the checkout button.
// Auto-renders: "Tickets remaining: 3 / 5"
<RealNodeQuotaBadge label="Tickets remaining" />
```

5.2 4.2. User Dashboard — Device Management

This component injects a complete, interactive dashboard into your users' "My Account" area, empowering them to:

- Audit the list of authorized hardware devices (smartphones, laptops) bound to their identity.
- Remotely revoke access for a lost or stolen device.
- Detach hardware bindings prior to device resale.
- Transfer account access to a different user securely.
- Recover access using an emergency Master Code provided by your administrative team.

```
import { RealNodeDeviceManager } from '@emkaylabs/realnode-sdk/react';

export default function MyAccount() {
  return (
    <div>
      <h2>Security Settings</h2>
      {/* Automatically generates the entire management interface. */}
      {/* Zero additional HTML or CSS logic required.              */}
      <RealNodeDeviceManager apiKey="pk_live_YOUR_PUBLIC_KEY" />
    </div>
  );
}
```

6 Step 5: Final Server-Side Validation (Backend)

The cryptographic `proof` emitted by the browser must be validated by your backend infrastructure before fulfilling the order. This is a mandatory security checkpoint: your server performs an authenticated call to the RealNode API using your **Secret Key** to guarantee the proof has not been tampered with and the inventory quota remains valid.

```
// Example: Node.js / Express backend
app.post('/api/your-checkout-endpoint', async (req, res) => {
  const proof = req.body;
  // proof = { idh, device_hash, device_token, quantity, status }
```

```
  const response = await fetch('https://api.emkaylabs.tech/consume', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'X-RealNode-API-Key':    'pk_live_YOUR_PUBLIC_KEY',
      'X-RealNode-Secret-Key': 'sk_live_YOUR_SECRET_KEY'
    },
    body: JSON.stringify({
      quantity:     proof.quantity,
      idh:          proof.idh,
      device_hash:  proof.device_hash,
      device_token: proof.device_token,
      client_id:    'your-website-id',
      // event_id: 'CONCERT-2024-PARIS' // Optional: for event-scoped
          quotas
    })
  });

  if (response.ok) {
    // RealNode successfully validated the attestation. Proceed with
        payment.
    await processPayment();
    res.status(200).json({ success: true });
  } else {
    // Quota exceeded, or invalid cryptographic signature.
    res.status(403).json({ error: "Transaction denied by RealNode Engine."
        });
  }
});
```

Standard `/consume` API Responses:

HTTP Code	Meaning	Recommended Action
200	Transaction validated, quota deducted	Finalize the order
403	Invalid signature or bot detected	Reject the transaction
429	Quota exhausted for this user IDH	Display a cooldown message
401	Invalid API key credentials	Verify your keys in the dashboard

Dedicated B2B Support
As a RealNode enterprise client, you receive priority integration support. For any architectural inquiries, reach out to our engineering team at **security@emkaylabs.tech**. We guarantee a technical response within 24 business hours.

7 Appendix: Advanced Configurations

7.1 Event-Scoped Quota Management (Vault Plan)

For highly controlled drops (limited edition tickets, exclusive sneaker releases, etc.), you can attach an internal event identifier to the `/consume` payload (see Step 5).

How does it work?

- No prior configuration is necessary within the RealNode dashboard.
- Simply append your own arbitrary string (e.g., `CONCERT-2024-PARIS`).
- The RealNode Engine will automatically isolate and track hardware quotas for that specific user strictly within the context of that event.