מדריך: עבודה עם האינטגרציה של טרנזילה ל-Base44
עבור ביצוע חיובים רגילים או תשלומים

האינטגרציה ל-Base44 מחברת את האפליקציה או האתר של בית העסק לדף התשלום (iFrame) של המסוף שלו, ומאפשרת קבלת תשלום או תשלומים באמצעות כרטיסי אשראי וארנקים דיגיטליים.

במה האינטגרציה תומכת?

- חיוב רגיל או תשלומים באמצעות **כרטיס אשראי**.
- חיוב באמצעות **ארנק PayPal** (זמין כהרחבה למסופי טרנזילה, אינו כרוך בתוספת תשלום. לצורך ההגדרת הארנק יש ליצור קשר עם המוקד הטכני שלנו: 073-2224444).
- חיוב באמצעות **ארנק Google Pay** (זמין כהרחבה למסופי טרנזילה, אינו כרוך בתוספת תשלום).
- חיוב באמצעות **ארנק bit** (זמין כהרחבה למסופי טרנזילה, כרוך בתוספת תשלום).
- פירוט מוצרים ב**מסמכים חשבונאיים** (זמין כהרחבה למסופי טרנזילה).

האינטגרציה אינה מקימה דף Notify (דף Notify משמש לקבלת תשובה עם פרטי העסקה כפי שבוצעה במקור, כולל אם עברה או נכשלה, ומאפשר עדכון של ההזמנה (שולמה/לא שולמה) ב-Back Order של האפליקציה/אתר).
לצורך הקמת דף Notify יש להיעזר במסמך "**סגירת הזמנה ב-Base44 באמצעות דף Notify**".

בנוסף, לא ניתן לבצע זיכויים ישירות מהאפליקציה/האתר באמצעות האינטגרציה.
יש לבצע זיכויים באמצעות מערכת my.tranzila של המסוף.

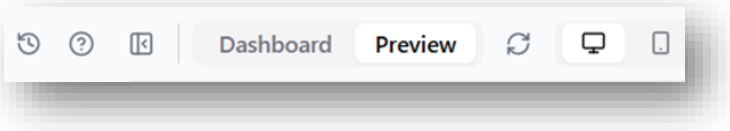
דרישות לעבודה עם האינטגרציה

- קודם כל – נדרש **מסוף טרנזילה פעיל**, כולל **שם המסוף** ו**סיסמת PW**.
- **חשבון Builder ומעלה** של Base44, המאפשר בניה עם פונקציית Backend.
- **HandShake מופעל** בדף הסליקה.
- **Process חדש מופעל** בדף הסליקה.

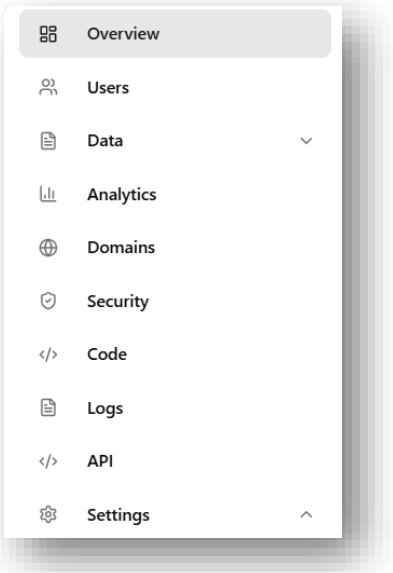
חיבור האפליקציה/האתר לדף הסליקה iFrame

לפני שמתחילים, צריך לוודא ששירות ה-**Backend** של האפליקציה פעיל. להפעלת שירות ה-Backend יש לפעול על פי ההנחיות הבאות:

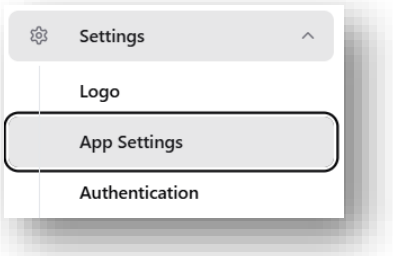
בתוך אפליקציית Base44 לוחצים על **Dashboard** באמצע הסרגל העליון:



לוחצים על **Settings**:



לוחצים על **App Settings**:



לוחצים על **Activate** להפעלת שירות ה-Backend.
אם בכפתור כתוב "Activated" והוא צבוע באפור כמו בתמונה – השירות כבר פעיל:

כעת נעתיק את ה-Prompt הארוך הבא במלואו ונדביק אותו בחלון השאילתה של האפליקציה:

```
Connect Tranzila integration:

Tranzila iFrame Integration Guide for Base44
Prerequisites
Required parameters:
• supplier - Terminal name
• TranzilaPW - Terminal token password
CRITICAL: Never generate code or UI without these required parameters

Overview:
The Tranzila iFrame integration allows Base44 to embed a secure credit card payment form directly into your website via an <iframe> , keeping users on your domain while ensuring full PCI-DSS compliance.
```

Integration Steps:

Step 1: Handshake Function
The Tranzila handshake is a security measure. Before initiating a payment through the Tranzila iFrame, your application needs to request a unique transaction identifier (thtk) from Tranzila's server. This thtk links the payment amount and details to a specific transaction, preventing fraud and ensuring PCI-DSS compliance by never exposing sensitive details on the client-side.

Prerequisites: Before you begin, ensure you have the following secrets configured in your Base44 app settings:
- supplier: Your Tranzila terminal name.
- TranzilaPW: Your Tranzila terminal token password.

1. Create the Backend Function File:
- In your Base44 project, navigate to the functions folder.
- Create a new JavaScript file (e.g., tranzilaCreateHandshake.js). The name you choose here will be used when you import and call the function from your frontend.

2. Implement the Function Logic:
Open the newly created file (tranzilaCreateHandshake.js) and add the following code structure. This is a Deno Deploy handler that processes incoming requests.

```
import { createClient } from 'npm:@base44/sdk@0.1.0';

// Initialize the Base44 SDK client
const base44 = createClient({
  appId: Deno.env.get('BASE44_APP_ID'),
});

// Define the Deno service handler
Deno.serve(async (req) => {
  try {
    // A. Authentication and User Context (Highly Recommended for Secure Operations)
    // Extract the Authorization token from the request headers
    const authHeader = req.headers.get('Authorization');
    if (!authHeader) {
      return new Response('Unauthorized', { status: 401 });
    }
    const token = authHeader.split(' ')[1]; // Get the actual token value
    base44.auth.setToken(token); // Set the token for the SDK client

    // Optionally, fetch user details to ensure a valid authenticated user
    const user = await base44.auth.me();
    if (!user) {
      return new Response('Unauthorized', { status: 401 });
    }

    // B. Extract Input Parameters from Request Body
    // The frontend will send the payment sum in the request body
    const { sum } = await req.json();

    if (!sum || sum <= 0) {
```

```
            return new Response(JSON.stringify({ error: 'Invalid sum' }), {
                status: 400,
                headers: { 'Content-Type': 'application/json' }
            });
        }

        // C. Access Secrets (Terminal Name and Password)
        // Retrieve your Tranzila secrets from environment variables (set in Base44 dashboard)
        const supplier = Deno.env.get('supplier');
        const TranzilaPW = Deno.env.get('TranzilaPW');

        if (!supplier || !TranzilaPW) {
            // This should ideally be caught during development, not in production
            console.error("Missing Tranzila credentials in environment variables.");
            return new Response(JSON.stringify({ error: 'Server configuration error: Missing Tranzila credentials' }), {
                status: 500,
                headers: { 'Content-Type': 'application/json' }
            });
        }

        // D. Construct the Tranzila Handshake URL
        const handshakeUrl =
    `https://api.tranzila.com/v1/handshake/create?supplier=${supplier}&sum=${sum}&TranzilaPW=${TranzilaPW}`;

        // E. Make the API Call to Tranzila
        const response = await fetch(handshakeUrl);
        const data = await response.text(); // Tranzila returns a plain text response like 'thtk=...'

        // F. Extract the 'thtk' (Transaction Handshake Token) - CORRECTED LOGIC
        // The response typically looks like "thtk=your_unique_token_string"
        const thtkPrefix = "thtk=";
        let thtk = data.trim();
        if (thtk.startsWith(thtkPrefix)) {
            thtk = thtk.substring(thtkPrefix.length); // Extract only the token value
        }

        // G. Return the Response to the Frontend
        return new Response(JSON.stringify({
            thtk, // Now 'thtk' will only contain the token string, e.g., "t9ad2b3ab5b8c7617498e57a63af00c81"
            supplier,
            sum
        }), {
            status: 200,
            headers: { 'Content-Type': 'application/json' }
        });

    } catch (error) {
        // H. Basic Error Handling
        // ... keep existing code (error handling) ...
    }
```

```
});
```

3. Using the Function in Your Frontend (e.g., Checkout.js): Once your backend function is deployed (this happens automatically when you save the file), you can call it from any React component or page in your Base44 application.

```
import { tranzilaCreateHandshake } from "@/functions/tranzilaCreateHandshake"; // Adjust path if needed

// ... inside your React component (e.g., CheckoutPage)

const initializePayment = async () => {
  // ... ensure all customer details are valid ...

  try {
    const totalAmount = getTotalPrice(); // Get total sum from cart
    const { data } = await tranzilaCreateHandshake({ sum: totalAmount });

    // 'data' will contain { thtk: "...", supplier: "...", sum: ... }
    // You can then use this data to dynamically generate and submit the Tranzila iFrame form.
    setHandshakeData(data); // Store the handshake data in component state

    // After setting state, trigger the iFrame form submission (e.g., via a ref or direct DOM manipulation)
    setTimeout(() => {
      const form = document.getElementById('tranzila-form');
      if (form) {
        form.submit();
      }
    }, 100);

  } catch (error) {
    console.error('Failed to initialize payment:', error);
    // Display an error message to the user
  }
};
```

Key Considerations for this Function:

Security:

- Secrets: Never expose supplier or TranzilaPW in your frontend code. Always retrieve them securely using Deno.env.get() in your backend function.
- Authentication: The base44.auth part ensures that only authenticated users of your app can call this function, adding a layer of security.
- Amount Validation: While the handshake performs some validation, always validate the sum server-side (within your backend function) before making external API calls to prevent manipulation.
- Response Handling: Tranzila's handshake endpoint returns a plain text string. The function parses this to extract the thtk and returns it as a JSON object to your frontend, making it easier to work with.
- Error Handling: The try...catch block in the backend function is essential for catching any issues (network errors, invalid parameters, Tranzila API errors) and returning a meaningful error response to your frontend.
- Function Naming: Use camelCase for function file names (e.g., tranzilaCreateHandshake.js). This name will be automatically converted to tranzilaCreateHandshake when importing into your frontend.

Step 2: iFrame Configuration

Base URL: https://direct.tranzila.com/[terminalname]/iframenew.php

Method: POST

Core Parameters:

Payment Parameters:

Parameter – sum
Values – Positive decimal
Description – Transaction amount

Parameter – cred_type
Values – 1 = Credit card, 6 = Credit, 8 = Installments
Description – Payment type

Parameter – currency
Values – 1 = NIS, 2 = USD, 978 = EUR, 826 = GBP
Description – Currency type

Parameter – tranmode
Values – A = Standard, V = Verification (J5), K = Create token, N =Verification (J2)
Description – Transaction type

Parameter – new_process
Values – 1
Description – Enable handshake verification

Parameter – thtk
Values – String from handshake
Description – Transaction identifier

Add Payment Options:

Parameter – ppnewwin
Values – 2
Description – Add the option to pay with paypal

Parameter – bit_pay
Values – 1
Description – Add the option to pay with Bit

Parameter – google_pay
Values – 1
Description – Add the option to pay with Google, Note: add the allowpaymentrequest=true to the iframe

Customer Information Fields:

Parameter – company
Description – Company name

Parameter – contact
Description – Contact name

Parameter – email
Description – Email address

Parameter – address
Description – Street address

Parameter – phone
Description – Phone number

Parameter – city
Description – City

Parameter – pdesc
Description – Product description for invoice

Parameter – remarks
Description – Additional remarks

Display Customization:

Parameter – trBgColor
Values – Hexadecimal
Description – Background color

Parameter – trTextColor
Values – Hexadecimal
Description – Text color

Parameter – trButtonColor
Values – Hexadecimal
Description – Payment button color

Parameter – buttonLabel
Values – String
Description – Payment button text

Parameter – hidesum
Values – 1
Description – Hide payment sum (only with VK/K/NK tranmode)

Parameter – nologo
Values – 1
Description – Remove Tranzila logo

Parameter – hide_cc
Values – 1
Description – Hide credit card option

Parameter – accessibility
Values – 2
Description – Show accessibility button

Language Support:

Parameter – lang
Values – il
Description – Hebrew

Parameter – lang
Values – us
Description – English

Parameter – lang
Values – ar
Description – Arabic

Parameter – lang
Values – ru
Description – Russian

Parameter – lang
Values – es
Description – Spanish

Parameter – lang
Values – de
Description – German

Parameter – lang
Values – fr
Description – French

Parameter – lang
Values – jp
Description – Japanese

Installments Configuration:

For installment payments, set cred_type=8 and use these parameters:

Parameter – maxpay
Description – Maximum number of installments

Product Details for Invoice:

To include itemized products on the invoice:
1. Enable product details: Add u71=1 parameter
2. Create product array:

```
json
[
{
"product_name": "Product 1",
"product_quantity": 2,
"product_price": 50.00
},
{
"product_name": "Product 2",
"product_quantity": 1,
"product_price": 100.00
}
]
```

3. Stringify without spaces:

```
json
[{"product_name":"Product 1","product_quantity":2,"product_price":50.00},{"product_name":"Product
2","product_quantity":1,"product_price":100.00}]
```

4. URL encode (use rawurlencode, not urlencode):
• Spaces must be encoded as 20%, not +
• Send as json_purchase_data parameter
Invoice Limitations:
• Maximum 6 products with logo and signature
• Maximum 10 products without lower logo/signature
• Product description: max 118 characters
• Total must equal sum of (quantity × price) for all products

Implementation Examples:

Basic Payment Form:

```
Html
<form action="https://direct.tranzila.com/[terminalname]/iframenew.php"
    target="tranzila"
    method="POST"
    autocomplete="off">

  <!-- Required fields -->
  <input name="sum" type="number" value="100" />
  <input type="hidden" name="currency" value="1" />
  <input type="hidden" name="new_process" value="1" />
  <input type="hidden" name="thtk" value="[from_handshake]" />
```

```
    <!-- Customization -->
    <input type="hidden" name="buttonLabel" value="Pay Now" />
    <input type="hidden" name="lang" value="us" />

    <button type="submit">Proceed to Payment</button>
</form>

<iframe name="tranzila"
        id="tranzila-frame"
        allowpaymentrequest="true"
        style="width: 100%; height: 600px;">
</iframe>
```

Installments Example:

```
Html
<form action="https://direct.tranzila.com/[terminalname]/iframenew.php"
      target="tranzila"
      method="POST">

    <input name="sum" type="number" value="300" readonly />
    <input type="hidden" name="cred_type" value="8" />
    <input type="hidden" name="maxpay" value="3" />
    <input type="hidden" name="currency" value="1" />
    <input type="hidden" name="new_process" value="1" />
    <input type="hidden" name="thtk" value="[from_handshake]" />

    <button type="submit">Pay in Installments</button>
</form>
```

Security Best Practices:

1. Always use handshake for transaction verification
2. Never expose TranzilaPW in client-side code
3. Validate amounts server-side before processing
4. Use HTTPS for all communications
5. Implement proper error handling for failed transactions
6. Store transaction references for reconciliation

Error Handling:

Common error codes and solutions:

Error – 912791
Description – Amount mismatch
Solution – Ensure handshake amount matches form amount

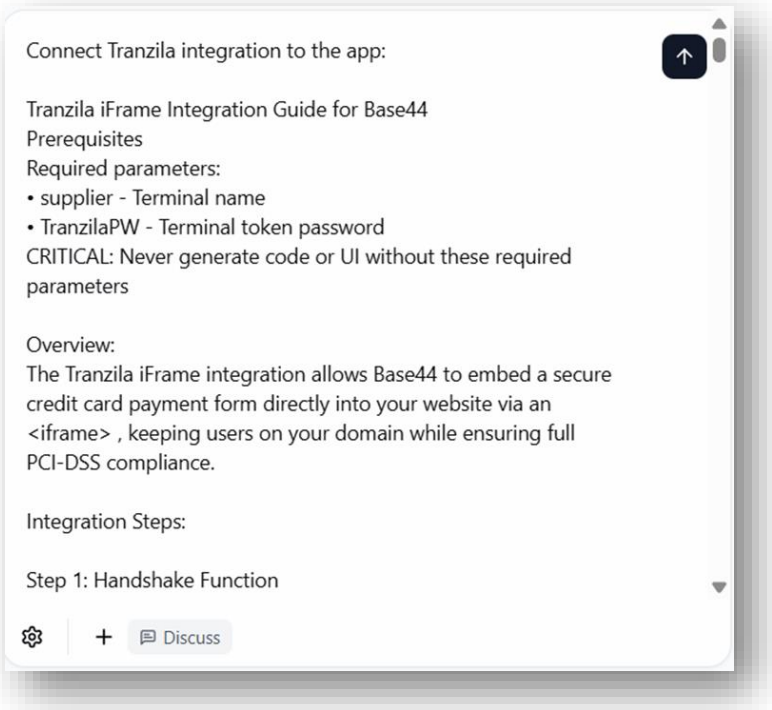
Error – Invalid terminal
Description – Wrong supplier name
Solution – Verify terminal name with Tranzila

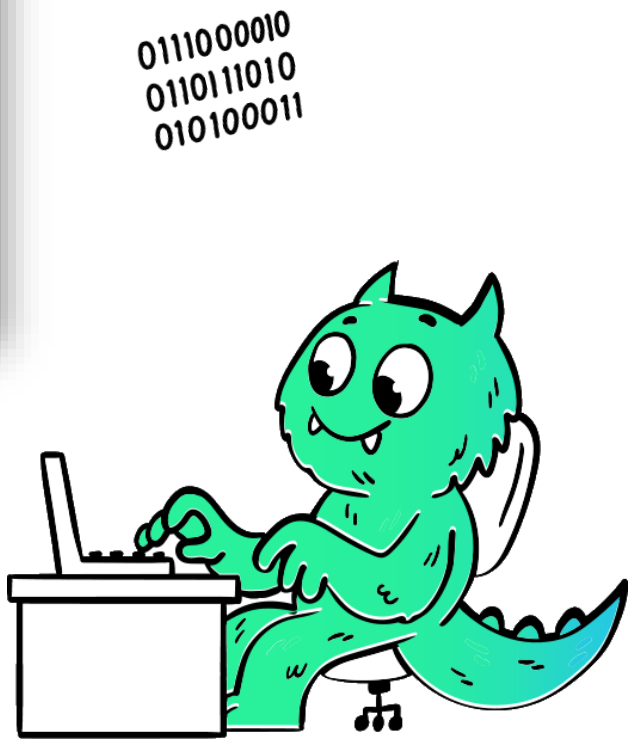
Error – Authentication failed
Description – Wrong password
Solution – Check TranzilaPW parameter

Checklist Before Going Live:

- Obtained production supplier and TranzilaPW
- Implemented handshake mechanism
- Configured all required form parameters
- Tested payment flow end-to-end
- Implemented error handling
- Customized display (colors, language, button text)
- Tested installments (if applicable)
- Verified invoice generation (if using products)
- Secured all sensitive parameters server-side
- Implemented transaction logging

נלחץ על החץ השחור להעלאת השאילתה לביצוע:

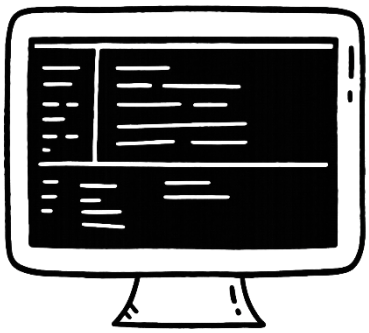




שגיאות אפשריות ופתרונן:

- שגיאה אפשרית: במקום דף תשלום iFrame מתקבלת הודעה language not supported.
הסבר: הערך של הפרמטר **lang** שמועבר לא תקין: עברית **il**, אנגלית **us**, ערבית **ar** (כל הערכים האפשריים מפורטים במדריך למפתח).
פתרון: נכתוב את שורת ה-Prompt הבאה עם הערך של השפה הרצויה, לדוגמה:

Change lang parameter value to il



- שגיאה אפשרית: בעת ניסיון מעבר לדף תשלום iFrame מתקבלת שגיאה 401 או 500.
הסבר: השגיאות האלה מציינות שתהליך ה-HandShake אינו עובד תקין.
פתרון: נעתיק, נדביק ונפעיל את ה-Prompt הבא:

```
Fix Tranzila handshake function, here is the instruction:

Step 1: Handshake Function
The Tranzila handshake is a security measure. Before initiating a payment through the Tranzila iFrame, your application needs to request a unique transaction identifier (thtk) from Tranzila's server. This thtk links the payment amount and details to a specific transaction, preventing fraud and ensuring PCI-DSS compliance by never exposing sensitive details on the client-side.

Prerequisites: Before you begin, ensure you have the following secrets configured in your Base44 app settings:
• supplier: Your Tranzila terminal name.
• TranzilaPW: Your Tranzila terminal token password.

1. Create the Backend Function File:
• In your Base44 project, navigate to the functions folder.
• Create a new JavaScript file (e.g., tranzilaCreateHandshake.js). The name you choose here will be used when you import and call the function from your frontend.
2. Implement the Function Logic:
Open the newly created file (tranzilaCreateHandshake.js) and add the following code structure. This is a Deno Deploy handler that processes incoming requests.

import { createClient } from 'npm:@base44/sdk@0.1.0';

// Initialize the Base44 SDK client
const base44 = createClient({
  appId: Deno.env.get('BASE44_APP_ID'),
```

```
});

// Define the Deno service handler
Deno.serve(async (req) => {
  try {
    // A. Authentication and User Context (Highly Recommended for Secure Operations)
    // Extract the Authorization token from the request headers
    const authHeader = req.headers.get('Authorization');
    if (!authHeader) {
      return new Response('Unauthorized', { status: 401 });
    }
    const token = authHeader.split(' ')[1]; // Get the actual token value
    base44.auth.setToken(token); // Set the token for the SDK client

    // Optionally, fetch user details to ensure a valid authenticated user
    const user = await base44.auth.me();
    if (!user) {
      return new Response('Unauthorized', { status: 401 });
    }

    // B. Extract Input Parameters from Request Body
    // The frontend will send the payment sum in the request body
    const { sum } = await req.json();

    if (!sum || sum <= 0) {
      return new Response(JSON.stringify({ error: 'Invalid sum' }), {
        status: 400,
        headers: { 'Content-Type': 'application/json' }
      });
    }

    // C. Access Secrets (Terminal Name and Password)
    // Retrieve your Tranzila secrets from environment variables (set in Base44 dashboard)
    const supplier = Deno.env.get('supplier');
    const TranzilaPW = Deno.env.get('TranzilaPW');

    if (!supplier || !TranzilaPW) {
      // This should ideally be caught during development, not in production
      console.error("Missing Tranzila credentials in environment variables.");
      return new Response(JSON.stringify({ error: 'Server configuration error: Missing Tranzila credentials'
}), {
        status: 500,
        headers: { 'Content-Type': 'application/json' }
      });
    }
```

```
        // D. Construct the Tranzila Handshake URL
        const handshakeUrl =
`https://api.tranzila.com/v1/handshake/create?supplier=${supplier}&sum=${sum}&TranzilaPW=${TranzilaP
W}`;

        // E. Make the API Call to Tranzila
        const response = await fetch(handshakeUrl);
        const data = await response.text(); // Tranzila returns a plain text response like 'thtk=...'

        // F. Extract the 'thtk' (Transaction Handshake Token) - CORRECTED LOGIC
        // The response typically looks like "thtk=your_unique_token_string"
        const thtkPrefix = "thtk=";
        let thtk = data.trim();
        if (thtk.startsWith(thtkPrefix)) {
          thtk = thtk.substring(thtkPrefix.length); // Extract only the token value
        }

        // G. Return the Response to the Frontend
        return new Response(JSON.stringify({
          thtk, // Now 'thtk' will only contain the token string, e.g., "t9ad2b3ab5b8c7617498e57a63af00c81"
          supplier,
          sum
        }), {
          status: 200,
          headers: { 'Content-Type': 'application/json' }
        });

    } catch (error) {
        // H. Basic Error Handling
        // ... keep existing code (error handling) ...
    }
});
```

3. Using the Function in Your Frontend (e.g., Checkout.js): Once your backend function is deployed (this happens automatically when you save the file), you can call it from any React component or page in your Base44 application.

```
import { tranzilaCreateHandshake } from "@/functions/tranzilaCreateHandshake"; // Adjust path if needed

// ... inside your React component (e.g., CheckoutPage)

const initializePayment = async () => {
    // ... ensure all customer details are valid ...

    try {
        const totalAmount = getTotalPrice(); // Get total sum from cart
        const { data } = await tranzilaCreateHandshake({ sum: totalAmount });
```

```
      // 'data' will contain { thtk: "...", supplier: "...", sum: ... }
      // You can then use this data to dynamically generate and submit the Tranzila iFrame form.
      setHandshakeData(data); // Store the handshake data in component state

      // After setting state, trigger the iFrame form submission (e.g., via a ref or direct DOM manipulation)
      setTimeout(() => {
        const form = document.getElementById('tranzila-form');
        if (form) {
          form.submit();
        }
      }, 100);

    } catch (error) {
      console.error('Failed to initialize payment:', error);
      // Display an error message to the user
    }
};
```

Key Considerations for this Function:

Security:

- Secrets: Never expose supplier or TranzilaPW in your frontend code. Always retrieve them securely using Deno.env.get() in your backend function.
- Authentication: The base44.auth part ensures that only authenticated users of your app can call this function, adding a layer of security.
- Amount Validation: While the handshake performs some validation, always validate the sum server-side (within your backend function) before making external API calls to prevent manipulation.
- Response Handling: Tranzila's handshake endpoint returns a plain text string. The function parses this to extract the thtk and returns it as a JSON object to your frontend, making it easier to work with.
- Error Handling: The try...catch block in the backend function is essential for catching any issues (network errors, invalid parameters, Tranzila API errors) and returning a meaningful error response to your frontend.

Function Naming: Use camelCase for function file names (e.g., tranzilaCreateHandshake.js). This name will be automatically converted to tranzilaCreateHandshake when importing into your frontend.