```
# Airtable Secrets

AIRTABLE_API_KEY="YOUR_AIRTABLE_PERSONAL_ACCESS_TOKEN"

AIRTABLE_BASE_ID="appXXXXXXXXXXXXXX"

AIRTABLE_TABLE_NAME="Leads" # Or whatever you call your table
```

Next.js App Router Twilio Webhook (route.ts)

This plan prioritizes security (signature validation), performance (raw body, minimal parsing), and maintainability (dedicated service/utility functions).

1. Robust Dependencies

You will need the following libraries for a secure and efficient implementation:

- twilio: For the secure signature validation function.
- airtable: For interacting with your database.
- next: Already included for the App Router handler.

```
npm install twilio airtable @types/airtable

# Ensure you have twilio-node installed
```

2. File Structure for Scalability

Keep your core business logic separate from the Next.js routing logic.

```
/app
  /api
    /twilio-webhook
      route.ts       <-- The Next.js Route Handler
/lib
  /services
    airtable.ts      <-- Airtable API client and logic
    twilio.ts        <-- Twilio validation logic
```

3. The route.ts - High-Efficiency Webhook Handler

This file in app/api/twilio-webhook/route.ts focuses on security and delegating heavy lifting.

TypeScript

```
// app/api/twilio-webhook/route.ts

import { NextRequest } from 'next/server';

import { validateTwilioSignature } from '@/lib/services/twilio';

import { updateRecordInAirtable } from '@/lib/services/airtable';


// Next.js config to ensure the request body is NOT automatically parsed.

// This is CRITICAL for Twilio Signature Validation which requires the raw body.

export const config = {

  api: {

    bodyParser: false, // Pages Router equivalent; still good practice to be explicit

  },

};


/**

 * Handle incoming POST requests from Twilio.

 * @param request The incoming NextRequest object.

 */

export async function POST(request: NextRequest) {

  // 1. RAW BODY RETRIEVAL (Most Efficient)

  // Twilio webhooks are typically application/x-www-form-urlencoded

  const rawBody = await request.text();
}



// 2. SIGNATURE VALIDATION (Security Priority)
const twilioSignature = request.headers.get('x-twilio-signature');
const url = request.url; // Use the full URL including query params

if (!twilioSignature) {
  console.error('Missing Twilio signature header');
  return new Response('Unauthorized - Missing Signature', { status: 401 });
}

// Use a dedicated utility function
if (!validateTwilioSignature(twilioSignature, url, rawBody)) {
  console.error('Twilio signature validation failed for URL:', url);
  return new Response('Unauthorized - Invalid Signature', { status: 403 });
}

// 3. PARSE FORM DATA (Only after security check)
// Use URLSearchParams to easily parse the application/x-www-form-urlencoded raw
string
const formData = new URLSearchParams(rawBody);
const callSid = formData.get('CallSid'); // Example key from Twilio
const digits = formData.get('Digits'); // Example key from Twilio

if (!callSid || !digits) {
  return new Response('Missing required parameters in payload', { status: 400 });
}
```

Classified - Confidential

```
  try {
   // 4. BUSINESS LOGIC DELEGATION (Scalability)
   // Delegate the Airtable interaction to a dedicated service function
   await updateRecordInAirtable(callSid, {
    'IVR_Digits': digits, // Map to your Airtable field name
    'Status': 'Completed',
   });

   // 5. SUCCESS RESPONSE (Twilio expects TwiML or an empty 200)
   // Returning an empty 200 OK is the most efficient response for a simple update
   return new Response('', { status: 200 });

  } catch (error) {
   console.error('Airtable Update Error:', error);
   // Respond with a 500 but log the error internally
   return new Response('Internal Server Error', { status: 500 });
  }
}
```

4. Dedicated Service Utilities (/lib/services)

lib/services/twilio.ts (Security)

TypeScript

```
// lib/services/twilio.ts
import twilio from 'twilio';
```

```
// Use environment variables for all secrets
const authToken = process.env.TWILIO_AUTH_TOKEN;

/**
 * Validates the X-Twilio-Signature against the request.
 */
export function validateTwilioSignature(
  signature: string,
  url: string,
  rawBody: string,
): boolean {
  if (!authToken) {
    console.error('TWILIO_AUTH_TOKEN is not set.');
    // In production, you might want this to fail loudly for security
    return false;
  }

  // Twilio's validation function handles the raw body as form data automatically
  // It requires the URL to be the one Twilio actually sent the request to.
  // Note: The 'twilio-node' package is required for this utility.
  return twilio.validateRequest(authToken, signature, url, rawBody);
}
```

lib/services/airtable.ts (Scalability/Efficiency)

TypeScript

```
// lib/services/airtable.ts
```

```
import Airtable from 'airtable';

// Use environment variables for all secrets
const baseId = process.env.AIRTABLE_BASE_ID;
const tableName = process.env.AIRTABLE_TABLE_NAME || 'Responses';
const airtableApiKey = process.env.AIRTABLE_API_KEY;

if (!airtableApiKey || !baseId) {
    throw new Error('Airtable credentials must be set in environment variables.');
}

// Initialize Airtable outside the handler for better performance (avoid re-initialization)
const base = new Airtable({ apiKey: airtableApiKey }).base(baseId);

/**
 * Updates a record in Airtable based on a unique identifier (CallSid).
 */
export async function updateRecordInAirtable(
  callSid: string,
  fields: Record<string, any>,
) {
  // 1. FIND RECORD (Efficiently using Airtable formula query)
  // Assumes you have a field named 'CallSid' in your Airtable table
  const records = await base(tableName).select({
    maxRecords: 1, // Only need the first match
    filterByFormula: `{CallSid} = '${callSid}'`,
```

Classified - Confidential

```
  }).firstPage();

  const recordId = records.length > 0 ? records[0].id : null;

  if (!recordId) {
    console.warn(`No Airtable record found for CallSid: ${callSid}`);
    // You might choose to create a new record here instead of throwing an error
    return;
  }

  // 2. UPDATE RECORD
  await base(tableName).update([
    {
      id: recordId,
      fields: fields,
    },
  ]);

  console.log(`Successfully updated Airtable record ${recordId} for CallSid: ${callSid}`);
}
```

Summary of Robustness and Efficiency

Feature	**Implementation**	**Benefit**
Security	Used request.text() (raw body) and twilio.validateRequest() in validateTwilioSignature utility.	Prevents automatic JSON parsing; ensures the signature validation uses the exact raw body string as required by Twilio.

Feature	**Implementation**	**Benefit**
Performance	Used URLSearchParams(rawBody) to parse the Twilio application/x-www-form-urlencoded body.	Faster and more direct than await request.formData() if only specific keys are needed and body is simple.
Scalability	Separated core logic (Twilio validation, Airtable interaction) into dedicated service files (/lib/services).	The route.ts remains thin, focused only on routing, security checks, and delegating; makes testing/maintenance easier.
Maintainability	Airtable base initialization is outside the handler function.	Avoids re-initializing the Airtable client on every request, which is marginally more efficient.
Correctness	Handled all environment variables with checks and imported them from the server environment.	Keeps secrets out of the codebase and ensures the app fails gracefully if they are missing.

Classified - Confidential