# **Technical Specification for the "Página Rei" Automation Project**

## **Section 1: System Architecture and Automation Blueprint**

This section establishes the foundational understanding of the project's architecture. It moves from a high-level overview to a detailed data flow, providing the "why" behind the system's design before delving into the "how." The goal is to provide the developer with a complete mental model of the system's operation.

### **1.1. Executive Summary: The "Página Rei" Ecosystem**

The core mission of the "Página Rei" project is to generate high-quality, AI-driven product reviews on-demand, triggered by user searches on a WordPress site. The architecture is built upon four key pillars: a dynamic Frontend (WordPress with HTMX), an intelligent Backend (the "Página Rei Core" plugin), a powerful Automation Layer (a self-hosted n8n instance), and essential External Services (Amazon Product Advertising API and OpenAI).1

Strategic technical decisions form the bedrock of this project. A self-hosted n8n instance is employed to ensure predictable, fixed operational costs, avoiding the variable, per-operation fees common to many cloud automation platforms. A custom database table within WordPress is used for high-performance caching, isolating transient data to prevent performance degradation of the core application. Finally, the AWS Lambda Proxy pattern is mandated for handling the complex authentication requirements of the Amazon API, ensuring robust and secure communication.1

### **1.2. Component Breakdown and Responsibilities**

Each component in the architecture has a distinct and well-defined role, designed for modularity and independent operation.

* **Frontend (WordPress with HTMX):** This is the user-facing layer, responsible for capturing the initial search query and dynamically rendering the final content without requiring full page reloads. The use of the HTMX library is a strategic choice to deliver a modern, reactive user experience while avoiding the complexity of a full-fledged JavaScript framework, keeping the development paradigm closer to the traditional WordPress/PHP environment.1
* **Backend (Página Rei Core Plugin):** This custom WordPress plugin is the "brain" of the application's WordPress-side logic. Its responsibilities include: managing the custom cache table (wp\_reviews\_temporarios), handling the initial AJAX request submitted by the frontend, performing the critical cache check, triggering the n8n webhook upon a "cache miss," and exposing secure REST API endpoints for the n8n workflow to communicate back and persist data.1
* **Automation Layer (Self-Hosted n8n):** This is the central orchestration engine for the entire content generation process. It is a stateless service responsible for executing the multi-step logic: receiving the trigger from WordPress, calling the Amazon API via the Lambda proxy, preparing and cleaning the product data, prompting the OpenAI model for content generation, and posting the final HTML result back to the WordPress backend. The self-hosted nature of n8n is a key business decision to ensure financial scalability.1
* **AWS Lambda Proxy:** This is a specialized, serverless microservice whose sole purpose is to handle the complex AWS Signature Version 4 (SigV4) cryptographic signing process required by the Amazon Product Advertising API (PA-API) 5.0. It acts as a secure and isolated intermediary for all n8n requests destined for Amazon, abstracting away the most difficult authentication challenge.1
* **External Services (Amazon PA-API & OpenAI):** These are the third-party data and intelligence sources. The Amazon PA-API provides the raw product data, including titles, features, and affiliate links. The OpenAI API provides the generative AI capabilities (via models like GPT-4) to transform that raw data into a high-quality, human-readable review.1

### **1.3. Data Flow Diagram and Sequence of Events**

The interaction between these components follows two distinct paths, depending on the state of the cache. This design is intentionally architected for decoupled scalability and resilience. The initial user request is handled immediately by WordPress, providing a fast user experience by returning either cached content or a loading interface.1 The most time-consuming part of the process—the external API calls and AI generation—is offloaded to n8n via a non-blocking, asynchronous webhook.1 This ensures that a slow response from Amazon or OpenAI does not cause the user's browser to hang or time out. The frontend then uses a polling mechanism to periodically check for the result, decoupling the user's view from the backend's processing state.1 This creates a resilient system; if the n8n workflow fails, the user is not left with a broken page. The error can be logged and handled in the background without directly impacting the live user session. Furthermore, this separation allows components to be scaled independently; the n8n instance can be given more resources to handle higher processing loads without altering the WordPress server configuration.

* **Cache Hit Flow:**
  1. User submits a search term via the HTMX-powered form.
  2. The WordPress AJAX Handler receives the request.
  3. It calculates a hash of the search term and queries the wp\_reviews\_temporarios table.
  4. A valid, non-expired record is found (Cache Hit).
  5. The stored HTML content is returned directly to the browser.
  6. HTMX injects the content into the page. The flow completes in milliseconds.
* **Cache Miss Flow:**
  1. User submits a search term.
  2. The WordPress AJAX Handler receives the request and finds no valid data in the cache (Cache Miss).
  3. The handler executes two actions concurrently:
     + **Action A (Frontend Response):** It immediately returns a "loading UI" snippet to the frontend. This HTML contains HTMX attributes that instruct the browser to start polling a specific REST endpoint every few seconds (e.g., hx-get="/get-review" hx-trigger="every 3s").1
     + **Action B (Backend Trigger):** It fires an asynchronous POST request (blocking: false) to the n8n Webhook URL, passing the user's search term.1
  4. The n8n workflow is triggered and begins its execution sequence (calling the Lambda Proxy, which calls Amazon; calling OpenAI).
  5. Upon completion, n8n makes a final, authenticated POST request to a custom WordPress REST endpoint (/add-review) to save the newly generated HTML content into the wp\_reviews\_temporarios cache table.1
  6. Meanwhile, the user's browser continues polling the /get-review endpoint. Once the data is saved in step 5, the next poll request results in a "Cache Hit."
  7. The /get-review endpoint returns the new HTML content.
  8. HTMX receives this content and swaps the "loading UI" with the final review. The flow is complete for the user.

## **Section 2: Amazon PA-API 5.0 Integration Protocol**

This section provides a deep dive into the specifics of the Amazon Product Advertising API, serving as the definitive guide for its correct and compliant implementation. The API's design enforces a symbiotic relationship where access and capacity are tied to performance, making the following protocols essential for the project's viability.

### **2.1. Credential Acquisition and Management**

Access to the PA-API 5.0 is a privilege granted to established affiliates, not a testing ground for new ones. The following steps are mandatory prerequisites 1:

1. **Become an Approved Associate:** The first step is to have a fully approved Amazon Associates account for the target marketplace (e.g., Brazil).
2. **Generate Qualifying Sales:** The account must demonstrate value by generating a minimum of three qualifying sales using traditional affiliate links. This is a non-negotiable gatekeeper requirement from Amazon.
3. **Request API Access:** Once the sales prerequisite is met, access to the PA-API can be requested through the Associates Central dashboard under "Tools > Product Advertising API." This action will generate the first pair of credentials.

These credentials consist of an **Access Key** and a **Secret Key**. The Secret Key is displayed only once upon creation and must be copied and stored in a secure location immediately. Exposing these keys constitutes a major security failure. The recommended secure storage methods are 1:

* **For WordPress:** As constants within the wp-config.php file, which is typically outside the public web root and not under version control.
* **For n8n:** Using the built-in, encrypted Credentials management system.

### **2.2. API Rate Limits and Usage Policies: The Business Viability Core**

The PA-API is a metered resource governed by a strict quota system that is fundamental to the project's business model.

* **Dynamic Quota System:** A new account receives an initial limit of 1 Transaction Per Second (TPS) and 8,640 Transactions Per Day (TPD). This limit is adjusted dynamically based on the shipped item revenue generated *specifically by API-driven links* in the preceding 30 days. The earning rate is approximately 1 TPD for every 5 cents (USD) in revenue.1 This creates a "pay-to-play" ecosystem where successful, high-converting applications are rewarded with more API capacity.
* **Inactivity Risk:** An account that fails to generate any qualifying sales via the API for 30 consecutive days will have its API access revoked. Access is restored only after a new sale is generated.1
* **Compliance Rules:** To ensure proper attribution and avoid suspension, all implementations must adhere to the following 1:
  1. Use the product URLs provided by the API without alteration or shortening.
  2. Ensure the Associate account and PA-API credentials are under the same primary email address.
  3. Include the correct PartnerTag (Associate ID) in every API request.

The direct link between sales performance and API availability makes the caching strategy, detailed in Section 5, more than a technical optimization—it is the linchpin of the entire business model. By fulfilling most requests from a local cache, the limited and valuable API calls are conserved for generating new content, maximizing the potential for revenue-generating clicks and thus ensuring the continued operation and growth of the API quota.

### **2.3. Data Mapping: From API Resources to Project Needs**

To generate high-quality reviews, specific data points must be requested from the API. The PA-API 5.0 uses a granular system of Resources that must be explicitly requested in each call. A critical finding from analyzing the API documentation and developer community feedback is that the PA-API 5.0 **does not provide access to customer review counts or average star ratings**.1 Resources that may have existed in the past or in beta are not available to the general public. Attempting to acquire this data through other means, such as web scraping, is a direct violation of Amazon's terms of service and would jeopardize the affiliate account.

**Mitigation Strategy:** The application must be transparent about this limitation. The user interface should not display "0 stars" or placeholder data. Instead, the AI-generated content should omit any mention of ratings and include a clear call-to-action with the affiliate link, such as: "Veja as avaliações de outros compradores diretamente na Amazon" (See reviews from other buyers directly on Amazon). The AI prompt must also be explicitly instructed not to invent or "hallucinate" this information.1

The following table serves as the official data dictionary for the project.

| Campo no "Página Rei" | Resource na PA-API 5.0 | Operação Principal | Exemplo de Valor (JSON) | Status de Disponibilidade |
| --- | --- | --- | --- | --- |
| Título do Produto | ItemInfo.Title.DisplayValue | GetItems/SearchItems | "Arranhador para Gatos com Torre" | ✅ Disponível |
| Preço Principal | Offers.Listings.Price.DisplayAmount | GetItems/SearchItems | "R$ 199,90" | ✅ Disponível |
| Link de Afiliado | DetailPageURL | GetItems/SearchItems | "https://www.amazon.com.br/dp/ASIN...&tag=seu-id-20" | ✅ Disponível |
| Imagem Principal | Images.Primary.Large.URL | GetItems/SearchItems | "https://m.media-amazon.com/images/I/....jpg" | ✅ Disponível |
| Características | ItemInfo.Features.DisplayValues | GetItems | ["Material resistente", "Fácil de montar"] | ✅ Disponível |
| Faixa de Preço (Variações) | VariationSummary.Price | GetVariations | {"LowestPrice":..., "HighestPrice":...} | ✅ Disponível |
| Número de Avaliações | CustomerReviews.Count (obsoleto) | N/A | N/A | ❌ Indisponível |
| Nota Média (Estrelas) | CustomerReviews.StarRating (obsoleto) | N/A | N/A | ❌ Indisponível |

### **2.4. Localization for the Brazilian Market**

To ensure all data, pricing, and affiliate links are correctly targeted for the Brazilian market, every API request must be configured with the following parameters 1:

* **Host Header:** webservices.amazon.com.br
* **SigV4 Signing Region:** us-east-1
* **Request Payload Marketplace:** www.amazon.com.br

## **Section 3: The AWS Lambda Proxy for SigV4 Authentication**

This section provides the complete technical specification for implementing the most complex component of the integration: the authentication proxy.

### **3.1. Architectural Rationale: Why the Proxy is Mandatory**

The Amazon PA-API 5.0 requires all requests to be authenticated using the AWS Signature Version 4 (SigV4) process. This is a complex cryptographic protocol that involves creating a canonical request, a "string to sign," and using the Secret Key to generate a unique HMAC-SHA256 signature for each request.1

The native HTTP Request node in n8n, even when configured with AWS credentials, is incapable of performing this specific SigV4 signing process for arbitrary APIs like the PA-API. This is a known limitation of the platform, and direct calls will invariably fail with "Missing Authentication Token" errors.1 The standard, robust architectural pattern to solve this is to delegate the authentication complexity to a specialized microservice. For this project, an AWS Lambda function will serve as this secure proxy.1

### **3.2. Complete Lambda Function Code (Node.js)**

The following production-ready Node.js code should be deployed to AWS Lambda. It is designed to receive a payload from n8n, use the AWS SDK v3 to perform the SigV4 signing, execute the request to the PA-API, and return the response.

JavaScript

// lambda\_proxy\_paapi5.js  
// This function acts as a proxy to sign and call the Amazon PA-API 5.0.  
const https = require('https');  
const { HttpRequest } = require("@aws-sdk/protocol-http");  
const { SignatureV4 } = require("@aws-sdk/signature-v4");  
const { Sha256 } = require("@aws-crypto/sha256-js");  
  
// Credentials must be stored as environment variables in the Lambda configuration for security.  
const ACCESS\_KEY = process.env.AMAZON\_PAAPI\_ACCESS\_KEY;  
const SECRET\_KEY = process.env.AMAZON\_PAAPI\_SECRET\_KEY;  
  
exports.handler = async (event) => {  
 // The 'event' object contains the payload sent from the n8n AWS Lambda node.  
 // Expected format: { "operation": "SearchItems", "payload": {... } }  
 const { operation, payload } = event;  
  
 // Initialize the SigV4 signer with credentials, region, service, and hash algorithm.  
 const signer = new SignatureV4({  
 credentials: { accessKeyId: ACCESS\_KEY, secretAccessKey: SECRET\_KEY },  
 region: 'us-east-1', // The region for PA-API signing is us-east-1  
 service: 'ProductAdvertisingAPI',  
 sha256: Sha256,  
 });  
  
 const apiPath = `/paapi5/${operation.toLowerCase()}`;  
 const amzTarget = `com.amazon.paapi5.v1.ProductAdvertisingAPIv1.${operation}`;  
  
 // Construct the HTTP request object that will be signed.  
 const request = new HttpRequest({  
 hostname: 'webservices.amazon.com.br', // Target hostname for the Brazilian marketplace  
 path: apiPath,  
 method: 'POST',  
 headers: {  
 'host': 'webservices.amazon.com.br',  
 'content-type': 'application/json; charset=utf-8',  
 'content-encoding': 'amz-1.0',  
 'x-amz-target': amzTarget  
 },  
 body: JSON.stringify(payload)  
 });  
  
 // Sign the request object. This adds the 'Authorization' header and other required headers.  
 const signedRequest = await signer.sign(request);  
  
 // Execute the signed request using Node's native https module.  
 return new Promise((resolve, reject) => {  
 const req = https.request(signedRequest, (res) => {  
 let data = '';  
 res.on('data', (chunk) => { data += chunk; });  
 res.on('end', () => {  
 // On success, resolve the promise with the status code and parsed JSON body.  
 resolve({  
 statusCode: res.statusCode,  
 body: JSON.parse(data)  
 });  
 });  
 });  
  
 req.on('error', (e) => {  
 // On failure, reject the promise with an error object.  
 reject({  
 statusCode: 500,  
 body: e.message  
 });  
 });  
  
 // Write the request body and end the request.  
 req.write(signedRequest.body);  
 req.end();  
 });  
};

### **3.3. Deployment and Configuration on AWS**

The developer must follow these steps to deploy the function:

1. Navigate to the AWS Lambda console and create a new function, selecting a recent Node.js runtime (e.g., Node.js 18.x or later).
2. Deploy the code provided above to the function.
3. In the function's "Configuration" tab, navigate to "Environment variables" and create two variables 1:
   * AMAZON\_PAAPI\_ACCESS\_KEY: Set the value to your PA-API Access Key.
   * AMAZON\_PAAPI\_SECRET\_KEY: Set the value to your PA-API Secret Key.
4. Ensure the function's execution role (IAM role) has basic permissions to run and write logs to Amazon CloudWatch (e.g., the AWSLambdaBasicExecutionRole policy).
5. Adjust memory and timeout settings as needed. A starting point of 256 MB of memory and a 15-second timeout is reasonable for this task.7

## **Section 4: n8n Workflow Implementation: The Automation Core**

This section provides a granular, node-by-node guide to constructing the complete automation workflow in a self-hosted n8n instance.

### **4.1. Credential Setup in n8n**

Before building the workflow, two sets of credentials must be configured within the n8n interface under "Credentials":

1. **AWS Credential:** Create a new credential of type "AWS". Populate the Access Key ID and Secret Access Key fields with the keys generated from the Amazon Associates account. This credential will be used by the AWS Lambda node.1
2. **HTTP Basic Auth Credential:** Create a new credential of type "HTTP Basic Auth". This will be used to authenticate with the WordPress REST API. The User field should be the WordPress username, and the Password field should be the **Application Password** generated in WordPress (see Section 5.4).1

### **4.2. Node-by-Node Workflow Configuration**

The workflow consists of a linear sequence of nodes. The following table details the configuration for each.

| Node # | Node Type | Purpose in Workflow | Key Parameters & Configuration (with examples) | Credential Used |
| --- | --- | --- | --- | --- |
| 1 | **Webhook** | Entry point triggered by WordPress on a cache miss. | **Method:** POST **Path:** A unique path, e.g., pagina-rei-generate **Response Mode:** On Received | None |
| 2 | **Code** | Prepare the payload for the SearchItems operation. | **JavaScript:** const searchTerm = $json.body.search\_term; return { operation: "SearchItems", payload: { Keywords: searchTerm, PartnerTag: "seu-id-br-20", PartnerType: "Associates", Marketplace: "www.amazon.com.br", SearchIndex: "PetProducts", ItemCount: 5, Resources: } }; | None |
| 3 | **AWS Lambda** | Invoke the proxy to search for products on Amazon. | **Function Name:** Name of your deployed Lambda function (e.g., lambda\_proxy\_paapi5) **Payload:** {{ $('Node 2').item.json }} | AWS Credential |
| 4 | **Code** | Extract ASINs from the SearchItems response. | **JavaScript:** const items = $json.body.SearchResult.Items; const asins = items.map(item => item.ASIN); return { asins }; | None |
| 5 | **Code** | Prepare the payload for the GetItems operation. | **JavaScript:** const asins = $json.asins; return { operation: "GetItems", payload: { ItemIds: asins, PartnerTag: "seu-id-br-20", PartnerType: "Associates", Marketplace: "www.amazon.com.br", Resources: } }; | None |
| 6 | **AWS Lambda** | Invoke the proxy to get detailed data for the selected products. | **Function Name:** Name of your deployed Lambda function **Payload:** {{ $('Node 5').item.json }} | AWS Credential |
| 7 | **Code** | Format the detailed product data into a clean text prompt for the AI. | **JavaScript:** (This code will iterate through the GetItems result and format it into a structured string, e.g., "Product 1:, Price: [Price], Features: [Feature 1, Feature 2]...") | None |
| 8 | **OpenAI** | Generate the review content using the prepared prompt. | **Model:** gpt-4-turbo **Messages > Role:** User **Messages > Content:** (A detailed prompt using E-E-A-T principles, instructing the AI to act as an expert and generate HTML content based on the text from Node 7. Example: "Aja como um especialista... Baseado nos seguintes dados: {{ $('Node 7').item.json.prompt\_text }}...") | OpenAI Credential |
| 9 | **HTTP Request** | POST the final HTML content back to the WordPress REST API to be cached. | **Method:** POST **URL:** https://your-site.com/wp-json/pagina-rei/v1/add-review **Body Content Type:** JSON **Body:** {"search\_term": "{{ $('Webhook').item.json.body.search\_term }}", "review\_content": "{{ $('OpenAI').item.json.choices.message.content }}"} | HTTP Basic Auth Credential |

## **Section 5: WordPress Integration: The Caching and Persistence Layer**

This section details the implementation of the Página Rei Core plugin, which serves as the application's foundation within the WordPress environment.

### **5.1. The wp\_reviews\_temporarios Cache Table**

To ensure high performance and prevent "database bloat" by mixing transient data with core WordPress content, a dedicated database table is used for caching.1 This table is created upon plugin activation using the

dbDelta() function.

The SQL statement for creating the table is as follows 1:

SQL

CREATE TABLE wp\_reviews\_temporarios (  
 id bigint(20) unsigned NOT NULL AUTO\_INCREMENT,  
 search\_hash varchar(32) NOT NULL,  
 search\_term text NOT NULL,  
 review\_content longtext NOT NULL,  
 expiration\_ts bigint(20) unsigned NOT NULL,  
 created\_at datetime NOT NULL DEFAULT CURRENT\_TIMESTAMP,  
 PRIMARY KEY (id),  
 UNIQUE KEY search\_hash (search\_hash),  
 KEY expiration\_ts (expiration\_ts)  
);

The search\_hash column is a UNIQUE key based on an MD5 hash of the user's search term, allowing for extremely fast cache lookups. The expiration\_ts column is also indexed to ensure that the periodic cleanup operation is efficient and does not lock the table for long periods.1

### **5.2. Automated Cache Maintenance (WP-Cron)**

To prevent the cache table from growing indefinitely, a daily task is scheduled using WP-Cron to purge expired records. The Pagina\_Rei\_Cron\_Manager class within the plugin will schedule an event that executes a DELETE query on the wp\_reviews\_temporarios table, removing all rows where the expiration\_ts is older than the current time. This automated maintenance is crucial for the long-term health of the database.1

### **5.3. Custom REST API Endpoints: The n8n-WordPress Bridge**

The plugin registers custom REST API endpoints to facilitate communication between n8n and WordPress. This provides a secure and controlled interface.1

* **GET /pagina-rei/v1/get-review**: This public endpoint is used by the frontend HTMX polling mechanism. It accepts a search\_term parameter, calculates its hash, and checks the cache. It returns a simple JSON object indicating the cache status: {"cache\_hit": boolean, "content": string|null}.1
* **POST /pagina-rei/v1/add-review**: This endpoint is used exclusively by the n8n workflow to save a newly generated review. It is protected and requires authentication (via Application Password). It expects a POST request with a JSON body containing the search\_term and the review\_content (HTML).1

### **5.4. Security: Application Passwords for Headless Authentication**

The modern, secure standard for machine-to-machine authentication in WordPress is the use of Application Passwords. This method avoids exposing a user's primary password and allows for integration-specific credentials that can be easily revoked without affecting the user's login.1

To generate a password 1:

1. In the WordPress dashboard, navigate to the profile of an Administrator or Editor user (Users > Profile).
2. Scroll down to the "Application Passwords" section.
3. Enter a descriptive name for the password (e.g., "n8n\_automation") and click "Add New Application Password."
4. WordPress will generate a 24-character password. This password must be copied immediately as it will not be shown again.
5. This password is then used in the "Password" field of the "HTTP Basic Auth" credential in n8n.1

## **Section 6: The Complete End-to-End Automation Flow**

This narrative walkthrough connects all the technical components into a single, coherent sequence of operations from the user's perspective.

### **6.1. User Interaction and Request Initiation**

The process begins when a user types a search query into the form on the "Página Rei" page and clicks the submit button. The hx-post attribute on the form immediately triggers an AJAX request to WordPress's admin-ajax.php endpoint, while the hx-indicator attribute simultaneously displays a loading spinner on the screen, providing instant feedback to the user.1

### **6.2. The "Cache Miss" Logic in WordPress**

The AJAX request triggers the handle\_search\_request function within the Página Rei Core plugin. This function calculates the hash of the search term and queries the wp\_reviews\_temporarios table. In the "cache miss" scenario, no valid record is found. The function then executes two critical actions in parallel: it fires a non-blocking POST request to the n8n webhook to start the generation process, and it immediately returns an HTML snippet for the "loading UI" back to the user's browser.1

### **6.3. The n8n Workflow Execution**

The webhook from WordPress activates the n8n workflow. The workflow proceeds through the nine steps detailed in Section 4.2: it receives the search term, calls the Lambda proxy to run SearchItems on the Amazon API, extracts the resulting ASINs, calls the Lambda proxy again to run GetItems for detailed data, formats this data into a clean prompt, sends the prompt to OpenAI to generate the HTML review, and finally, prepares to send the result back to WordPress.1

### **6.4. The Polling Mechanism and Final Display**

The "loading UI" returned to the browser contains an hx-trigger="every 3s" attribute, which instructs HTMX to make a new GET request to the /pagina-rei/v1/get-review endpoint every three seconds.1 For the first few attempts, while n8n is working, this endpoint will return

{"cache\_hit": false}, and nothing on the screen will change.

Once the n8n workflow completes its final step and POSTs the generated content to the /add-review endpoint, the data is saved in the cache table. The very next poll from the browser to /get-review will now find this new data, resulting in a {"cache\_hit": true} response containing the full HTML of the review. Upon receiving this response, HTMX swaps the loading UI with the final review content, completing the process for the user without them ever having to leave the page.1

## **Section 7: Final Implementation Checklist**

This consolidated checklist provides a logical sequence for deploying all components of the system, ensuring that dependencies are met correctly.

### **7.1. Phase 1: Amazon Account Setup**

* [ ] Ensure the Amazon Associates account is fully approved and has generated the minimum three qualifying sales.1
* [ ] Access the Product Advertising API section in Associates Central and generate a new pair of credentials (Access Key and Secret Key).1
* [ ] Securely store the Secret Key immediately, as it will not be shown again.1

### **7.2. Phase 2: WordPress Backend Setup**

* [ ] Develop and activate the full "Página Rei Core" plugin, including all its PHP classes (Database\_Manager, Rest\_Api\_Manager, Ajax\_Handler, Cron\_Manager).1
* [ ] Upon activation, verify that the wp\_reviews\_temporarios table has been correctly created in the WordPress database.1
* [ ] Add the Amazon API credentials as constants in the wp-config.php file for secure access by the plugin if needed (though the primary flow uses Lambda).1
* [ ] In the WordPress admin panel, create a dedicated user (or use an existing Admin/Editor) and generate a new **Application Password** for n8n. Store this password securely.1
* [ ] Create the "Página Rei" page and assign it the custom page template containing the HTMX form.1

### **7.3. Phase 3: AWS Lambda Deployment**

* [ ] In the AWS Management Console, create a new Lambda function using a recent Node.js runtime.1
* [ ] Deploy the complete proxy function code from Section 3.2.1
* [ ] Configure the function's environment variables: AMAZON\_PAAPI\_ACCESS\_KEY and AMAZON\_PAAPI\_SECRET\_KEY, populating them with the Amazon credentials.1
* [ ] Ensure the function's IAM role has the necessary permissions to execute and write to CloudWatch Logs.1

### **7.4. Phase 4: n8n Workflow and Credential Configuration**

* [ ] In the n8n instance, create the two required credentials: one of type "AWS" and one of type "HTTP Basic Auth" for WordPress.1
* [ ] Build the complete n8n workflow exactly as specified in the table in Section 4.2.1
* [ ] Configure the Webhook node and copy its **Production URL**.1
* [ ] Activate the workflow in n8n.

### **7.5. Phase 5: Final Integration and Testing**

* [ ] Paste the copied n8n Production URL into the appropriate settings field within the WordPress plugin so the handle\_search\_request function can call it.1
* [ ] Perform an end-to-end test by submitting a new search term from the "Página Rei" frontend and verifying that a review is generated and displayed correctly.

#### Referências citadas

1. Guia Técnico Página Rei\_.docx
2. HTMX and WordPress: How I Combined Both Tools - WPShout, acessado em julho 30, 2025, <https://wpshout.com/htmx-and-wordpress/>
3. Supercharging WordPress with HTMX: how to enhance the platform without complex frameworks - Kinsta, acessado em julho 30, 2025, <https://kinsta.com/blog/wordpress-htmx/>
4. AWS Signature Header with n8n - Feature Requests, acessado em julho 30, 2025, <https://community.n8n.io/t/aws-signature-header-with-n8n/25940>
5. Unable to call AWS Textract AnalyzeDocument via HTTP Request or AWS Lambda in n8n – Need guidance - n8n Community, acessado em julho 30, 2025, <https://community.n8n.io/t/unable-to-call-aws-textract-analyzedocument-via-http-request-or-aws-lambda-in-n8n-need-guidance/97672>
6. Best practices for working with AWS Lambda functions, acessado em julho 30, 2025, <https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html>
7. Mastering AWS Lambda Performance: Advanced Optimization Strategies for 2025, acessado em julho 30, 2025, <https://dev.to/rahulladumor/mastering-aws-lambda-performance-advanced-optimization-strategies-for-2025-3bfe>
8. WordPress credentials - n8n Docs, acessado em julho 30, 2025, <https://docs.n8n.io/integrations/builtin/credentials/wordpress/>
9. Wordpress aplication login requirements : r/n8n - Reddit, acessado em julho 30, 2025, <https://www.reddit.com/r/n8n/comments/1igs59d/wordpress_aplication_login_requirements/>
10. htmx ~ Documentation, acessado em julho 30, 2025, <https://htmx.org/docs/>