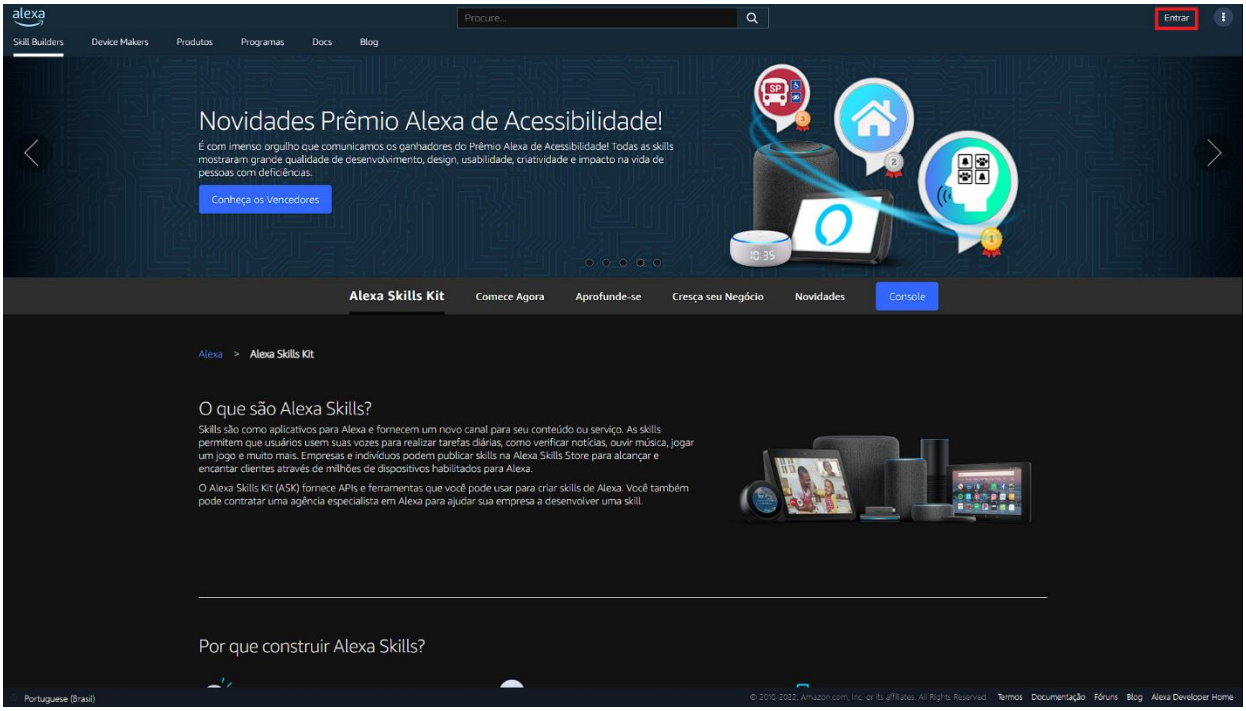
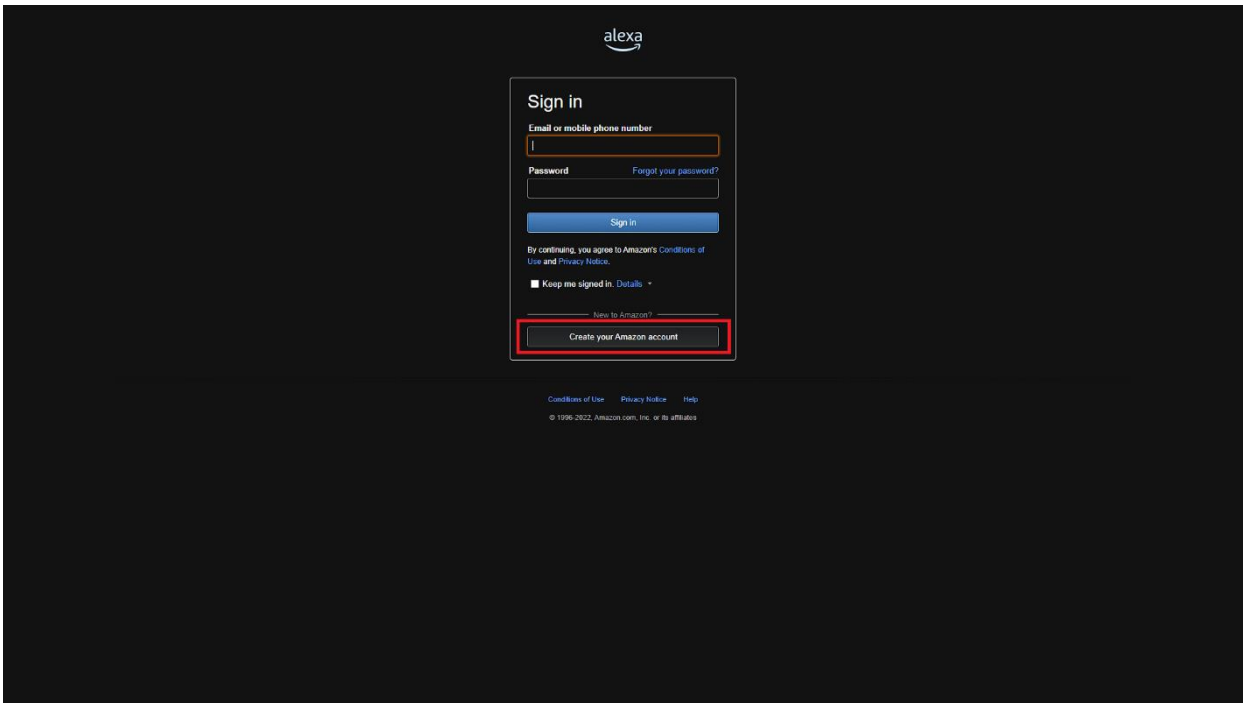
1. PRIMEIROS PASSOS

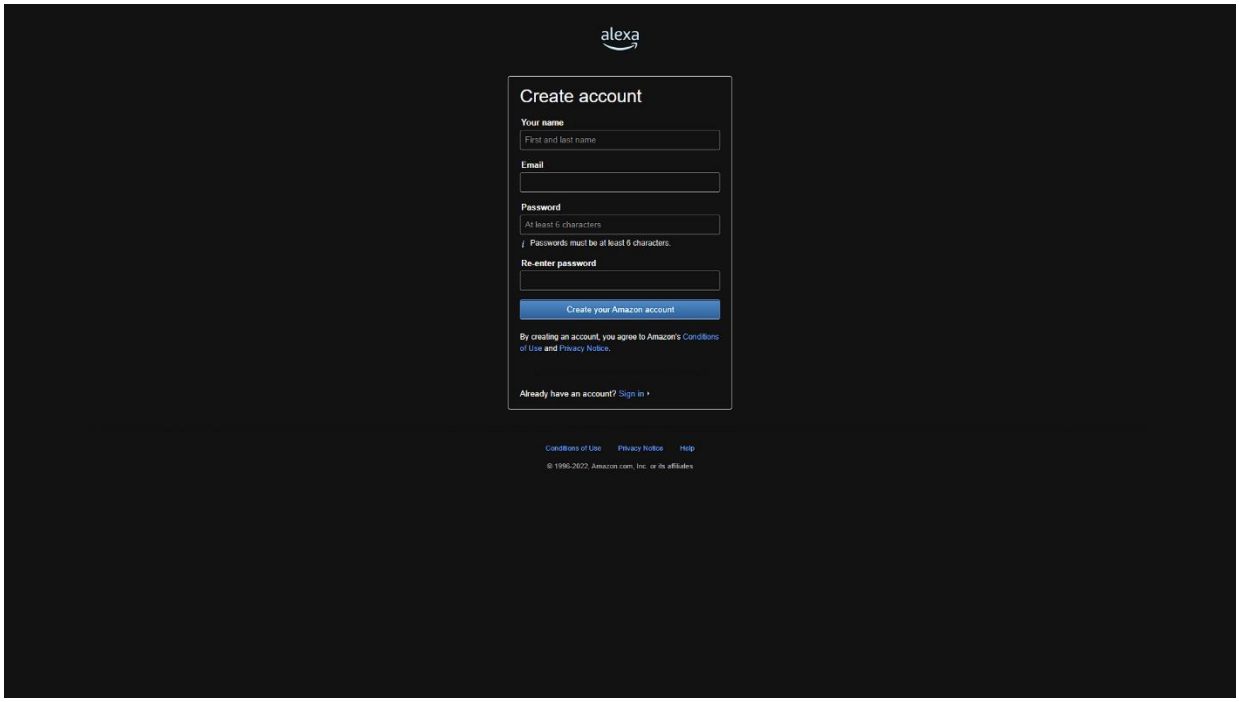
O primeiro passo para começar o desenvolvimento de nossa *skill* é acessar o seguinte endereço online: https://developer.amazon.com/pt-BR/alexa/alexa-skills-kit.

Após isso, devemos efetuar nosso cadastro no site.

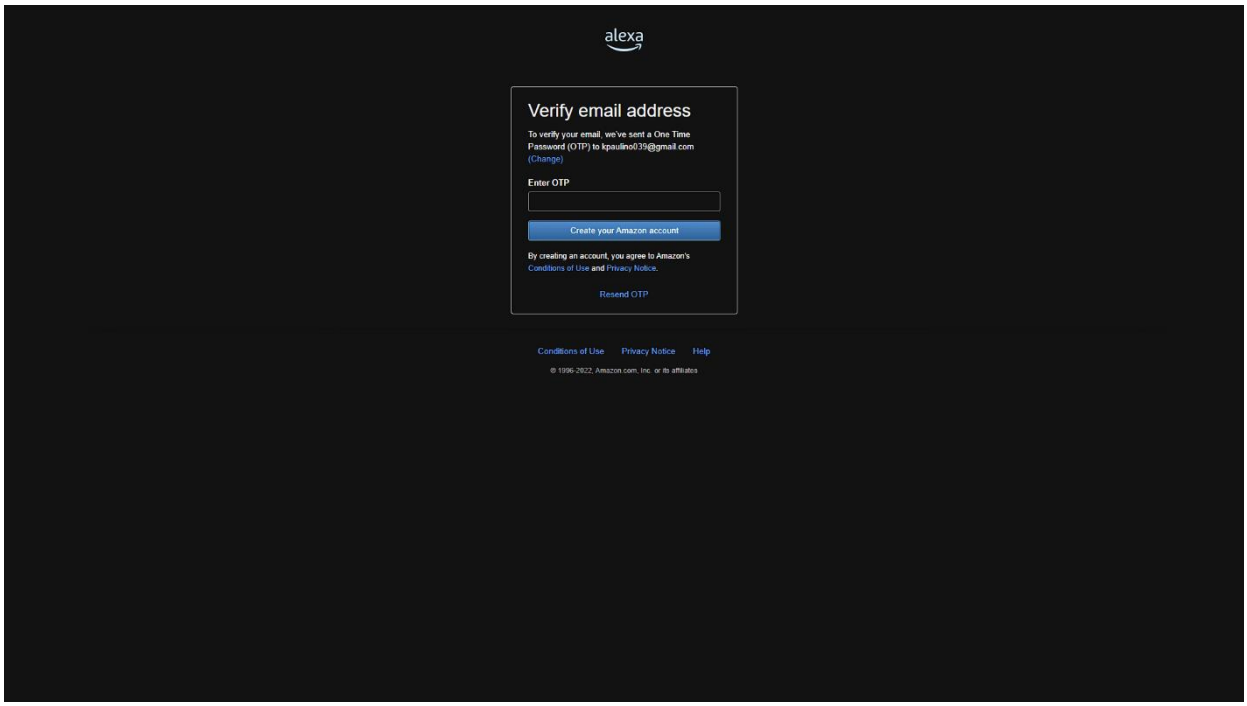


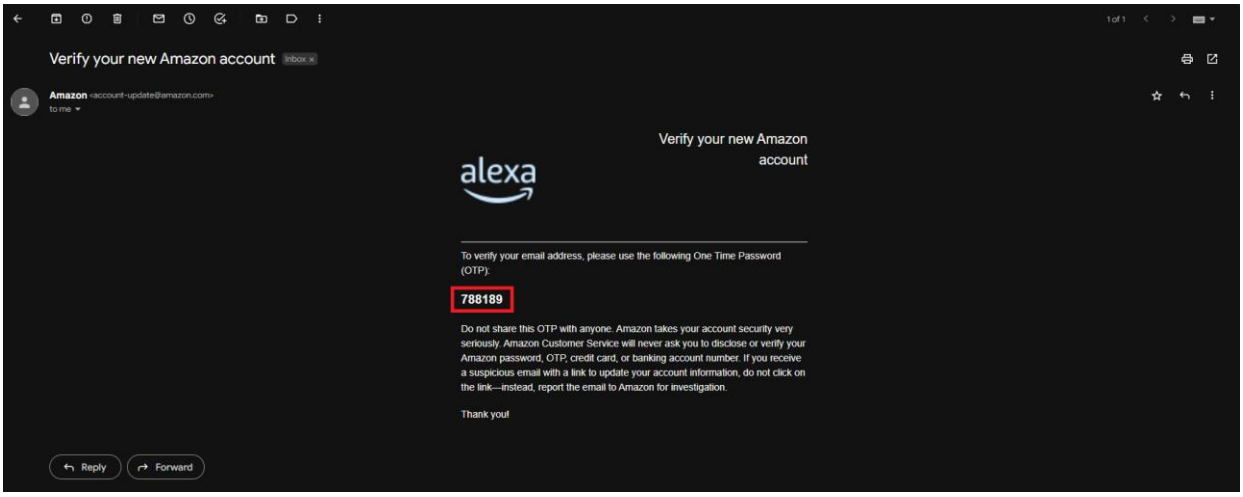


Efetue o cadastro preenchendo todos os campos com seus dados (Nome, E-mail e senha).

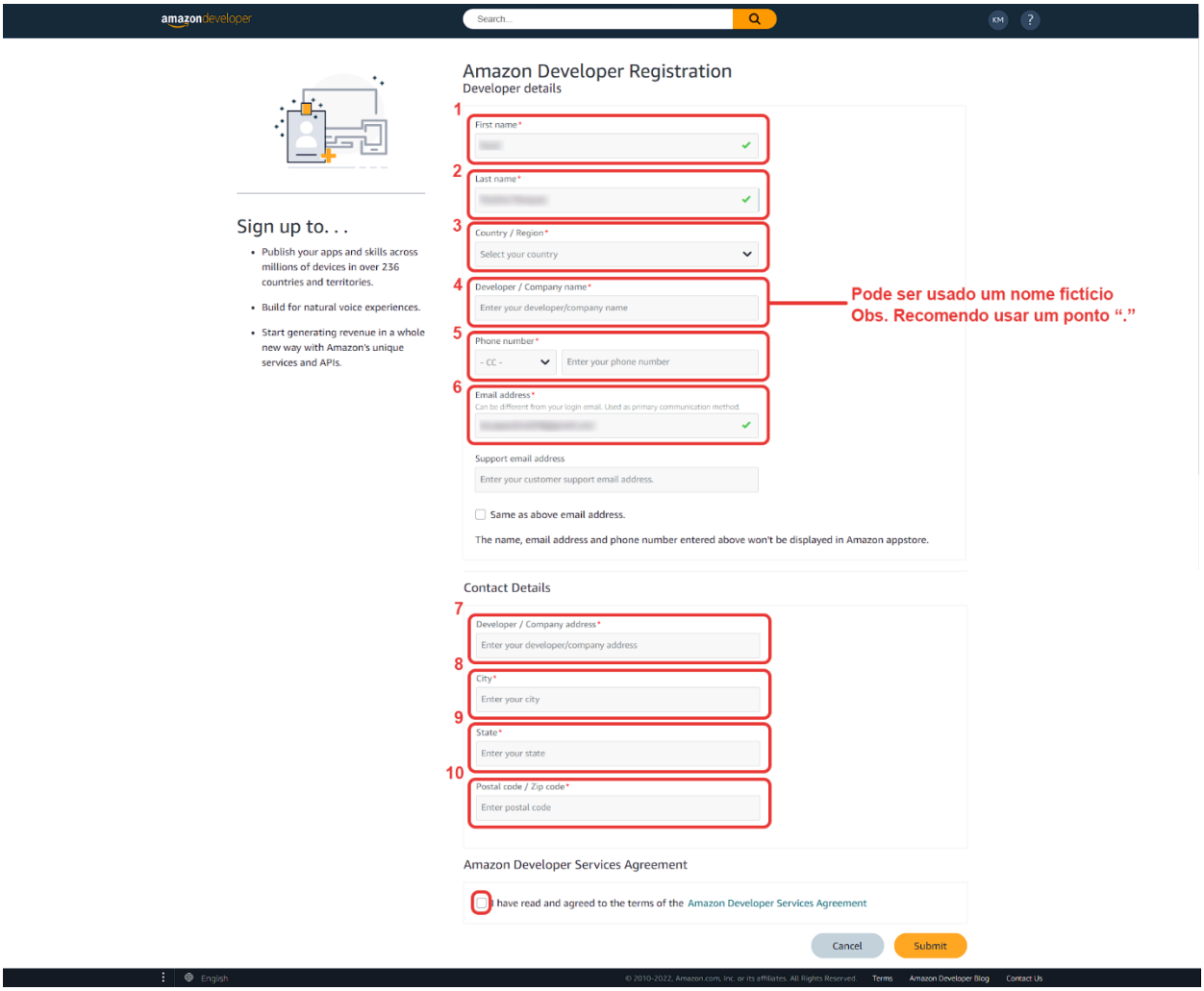


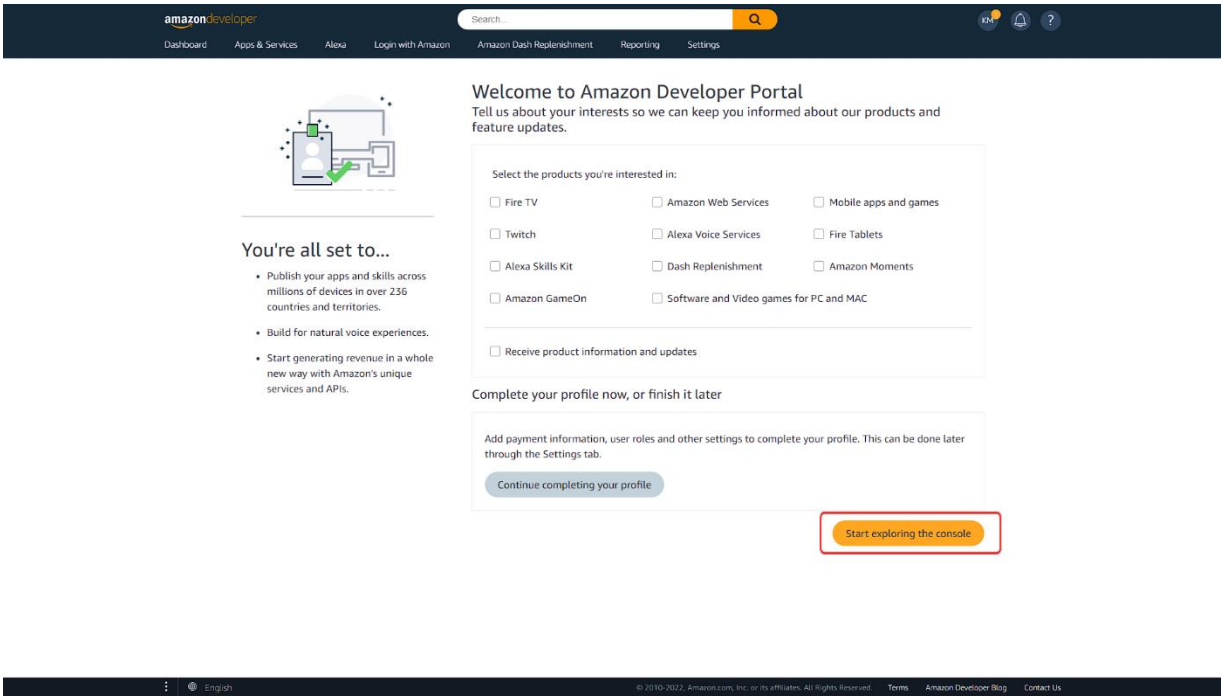
Após a criação, é necessário inserir um código chamado *OTP* que será enviado para o e-mail cadastrado na seguinte tela:



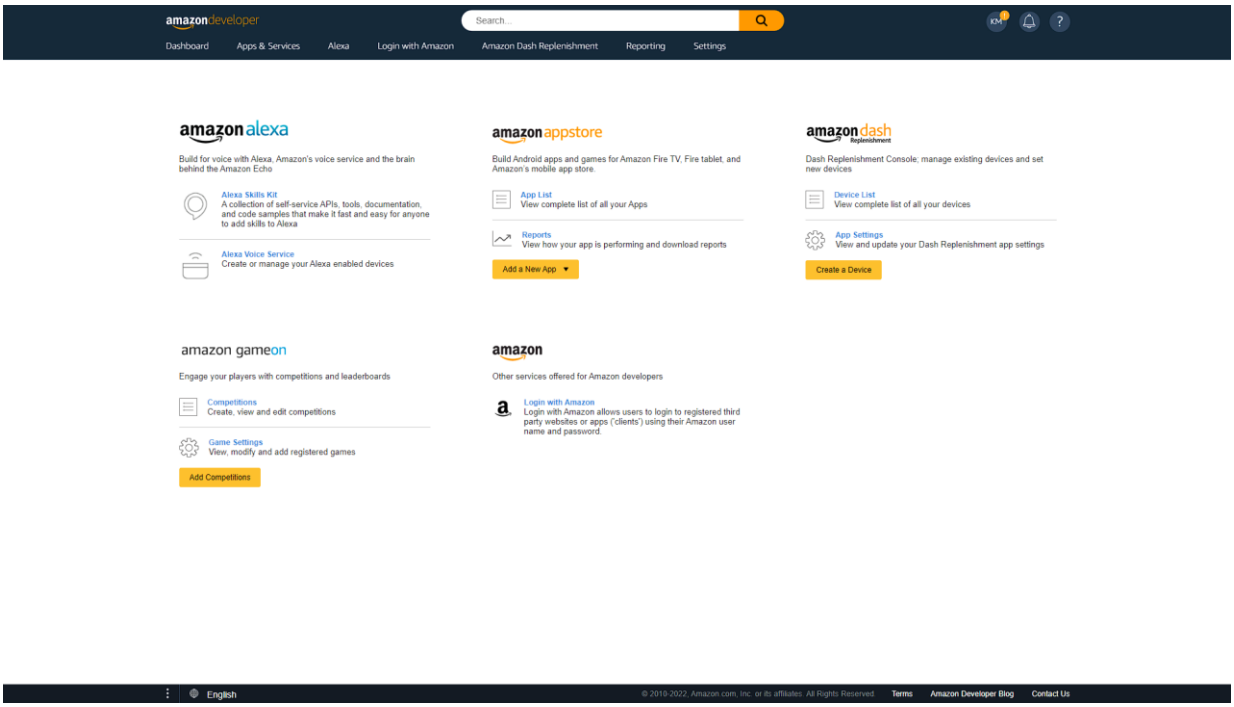


Após efetuar o cadastro, será necessário completar algumas informações obrigatórias pedidas pelo site:

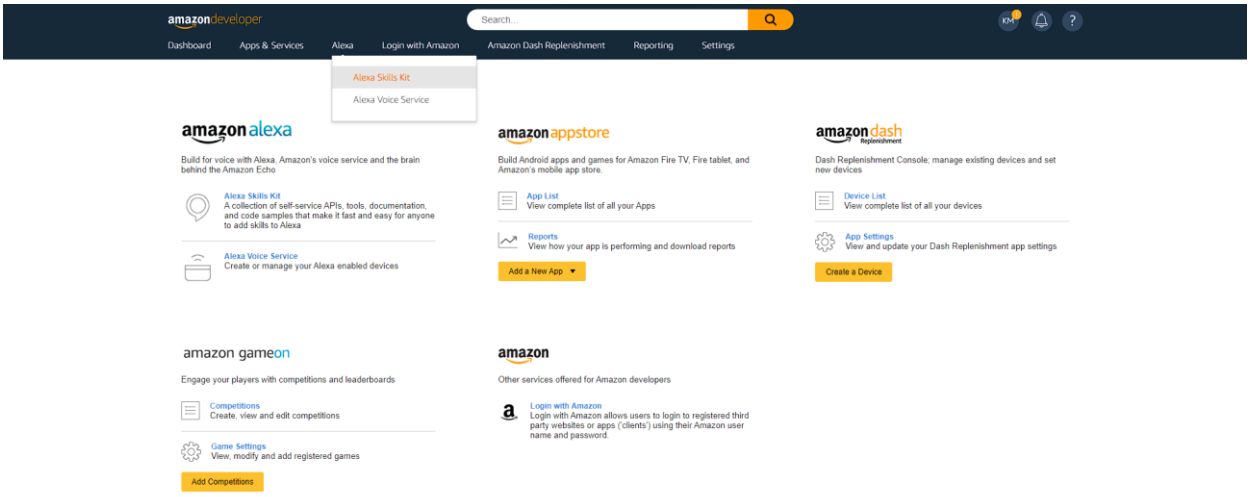


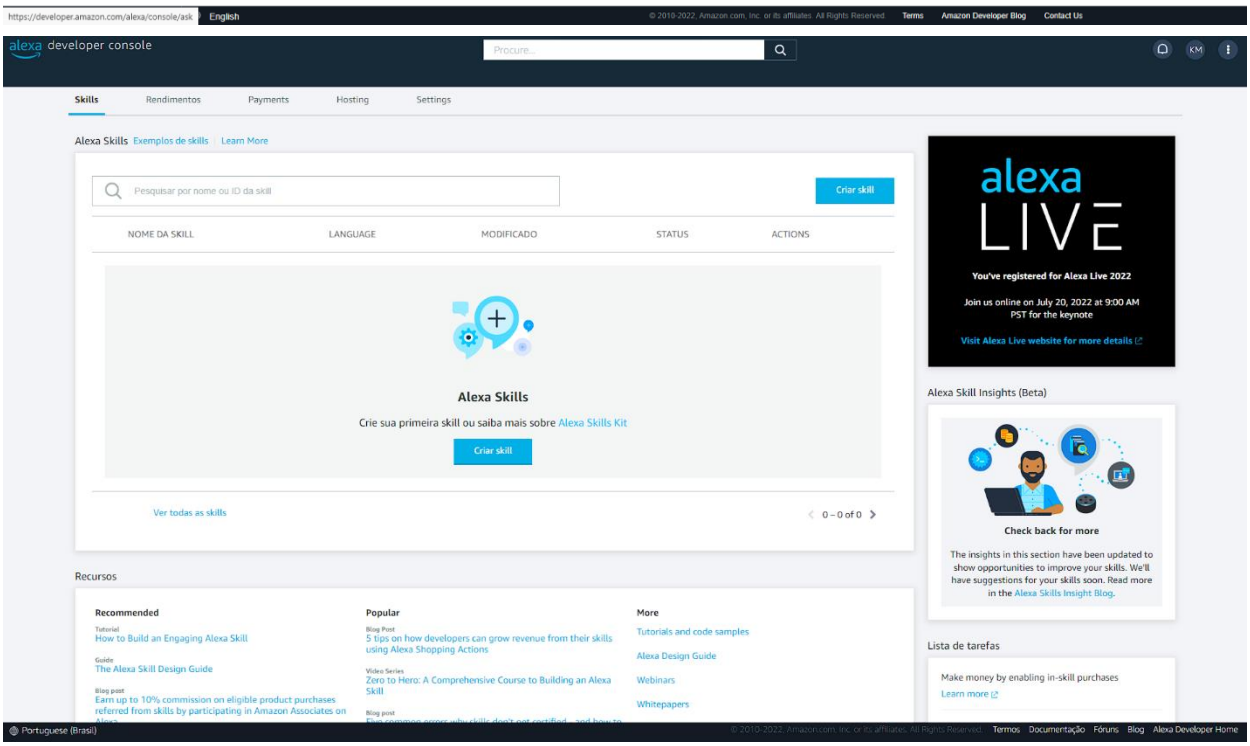


Ao concluir o cadastro, você seguirá para a seguinte página:



Para acessar a seção de criação de skills:

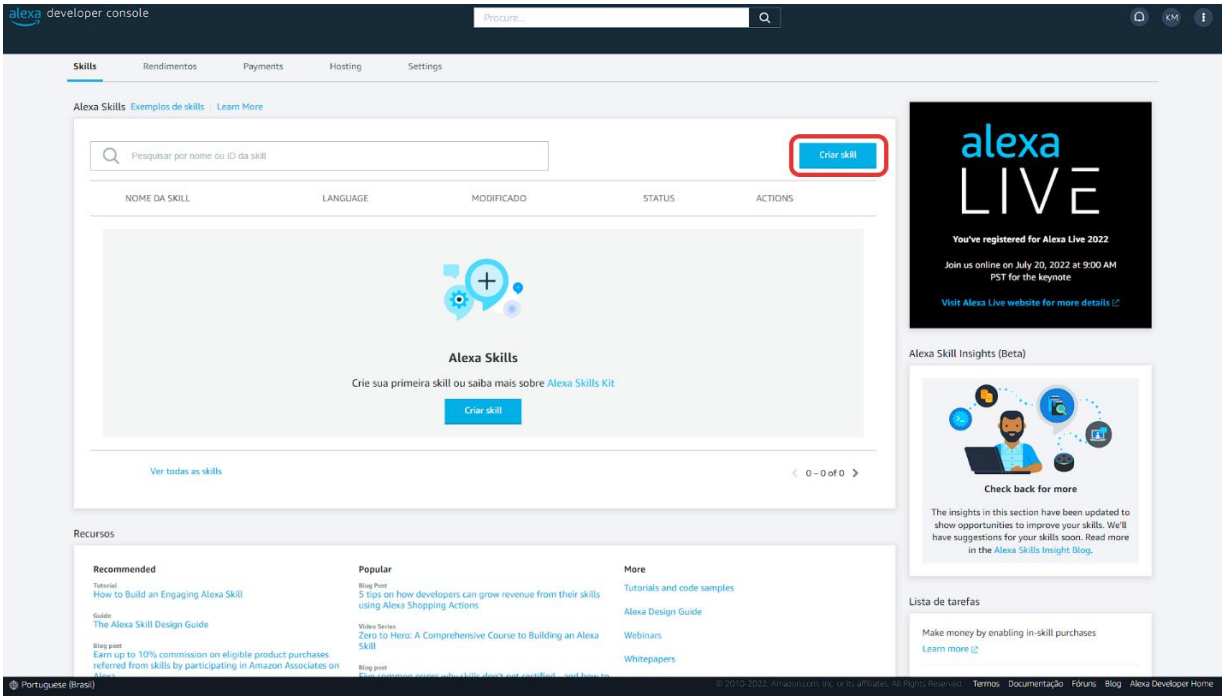




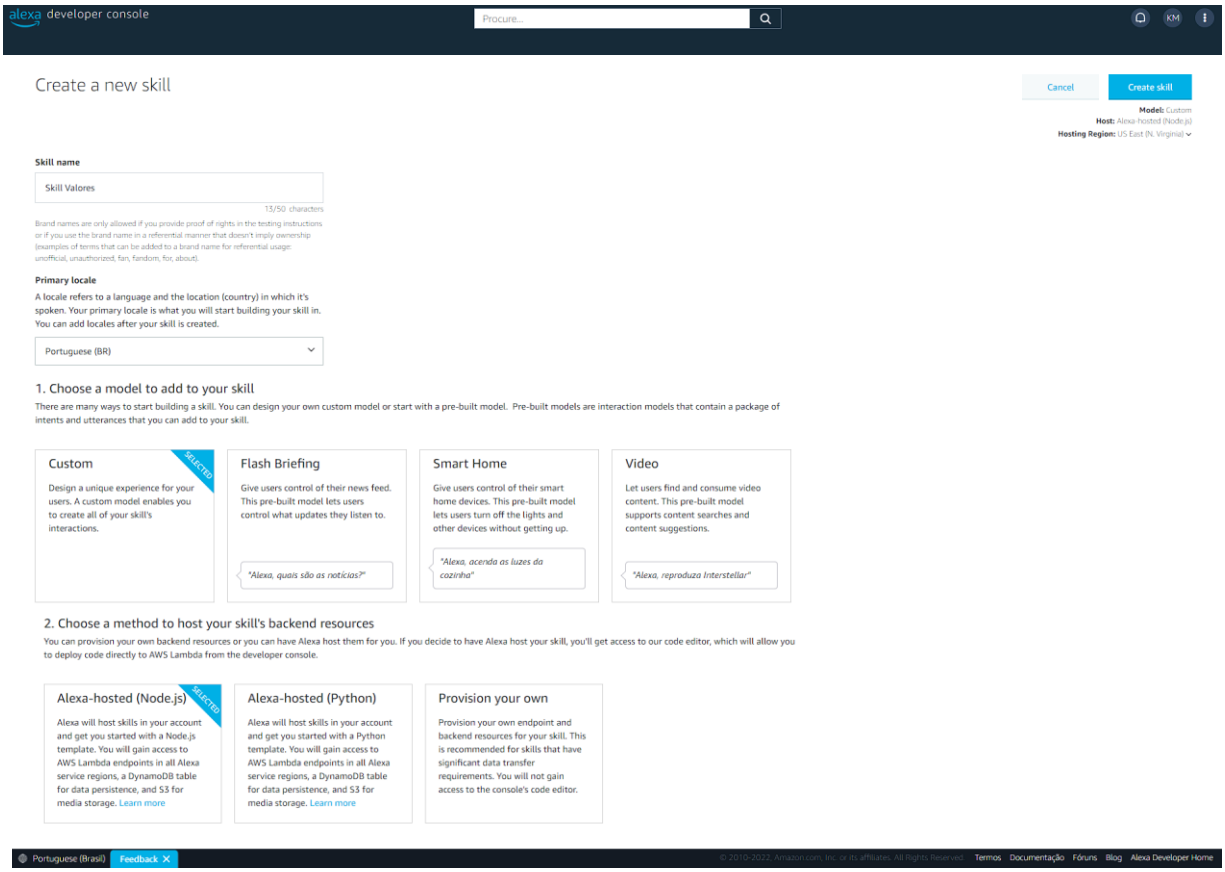
2. CRIAÇÃO DA SKILL

Logo em seguida ao nosso login, podemos construir nossa skill.

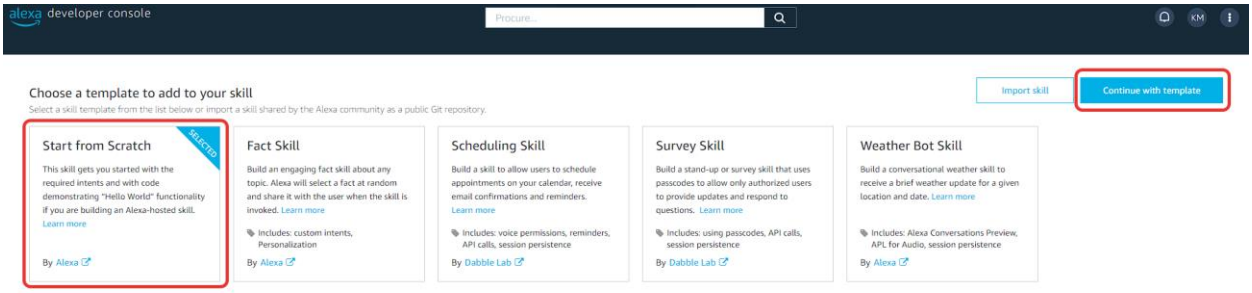
Para iniciar devemos pressionar o botão contendo a seguinte legenda: ***Criar skill***.

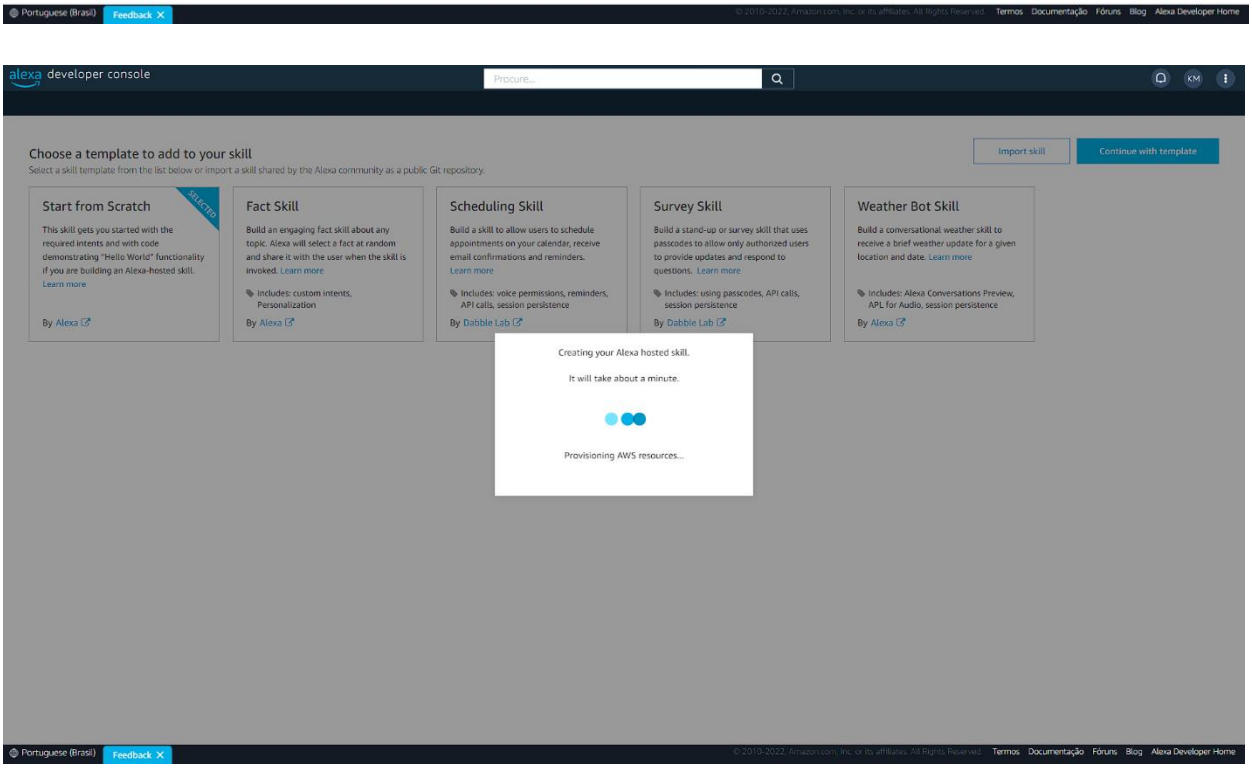


Na janela que será aberta devemos nomear a skill no campo **Skill name** como *skill valores* (ou algum de sua preferência), manter o **Primary locale** como *Portuguese (BR)* e permanecer o restante das configurações padrões.

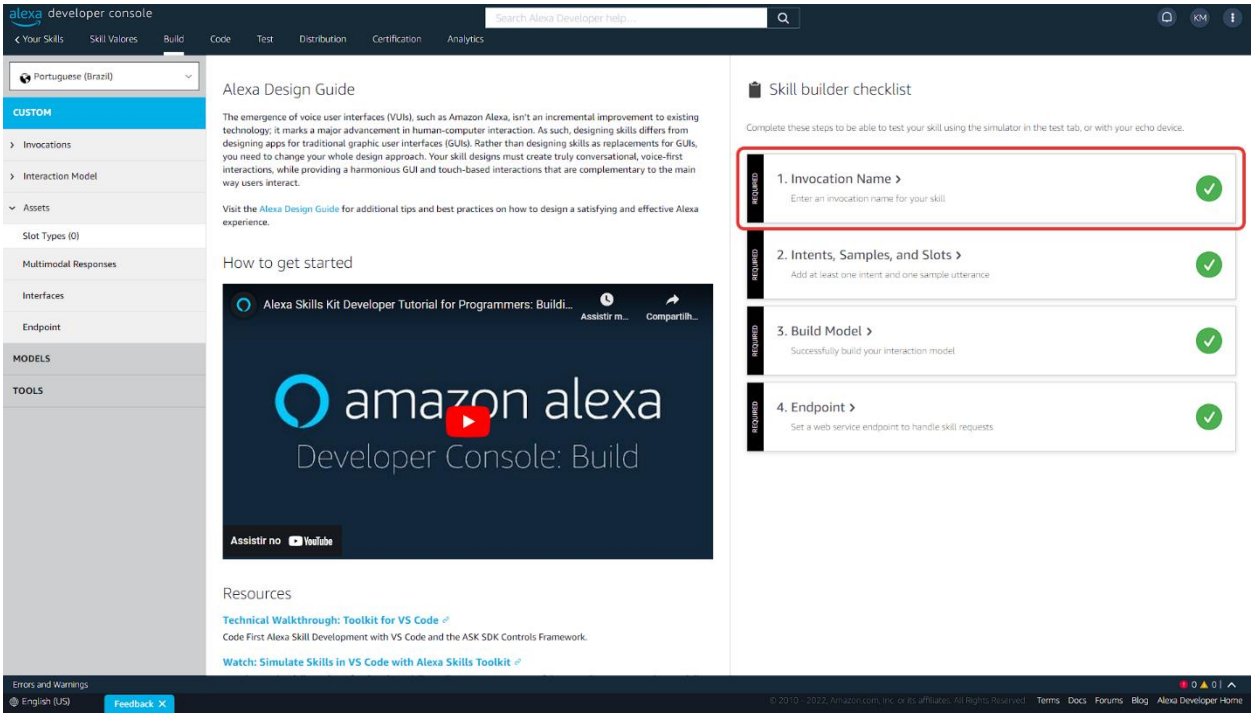


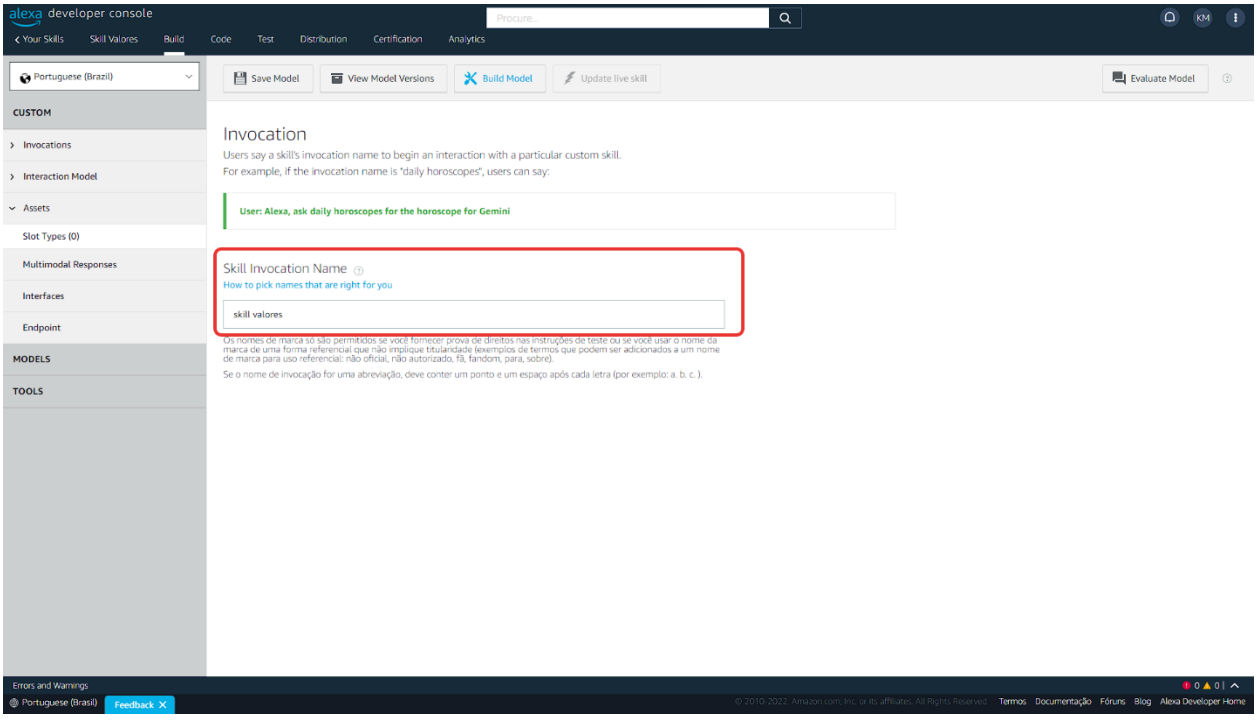
Manter em **Start from Scratch** e **Continue with template**.



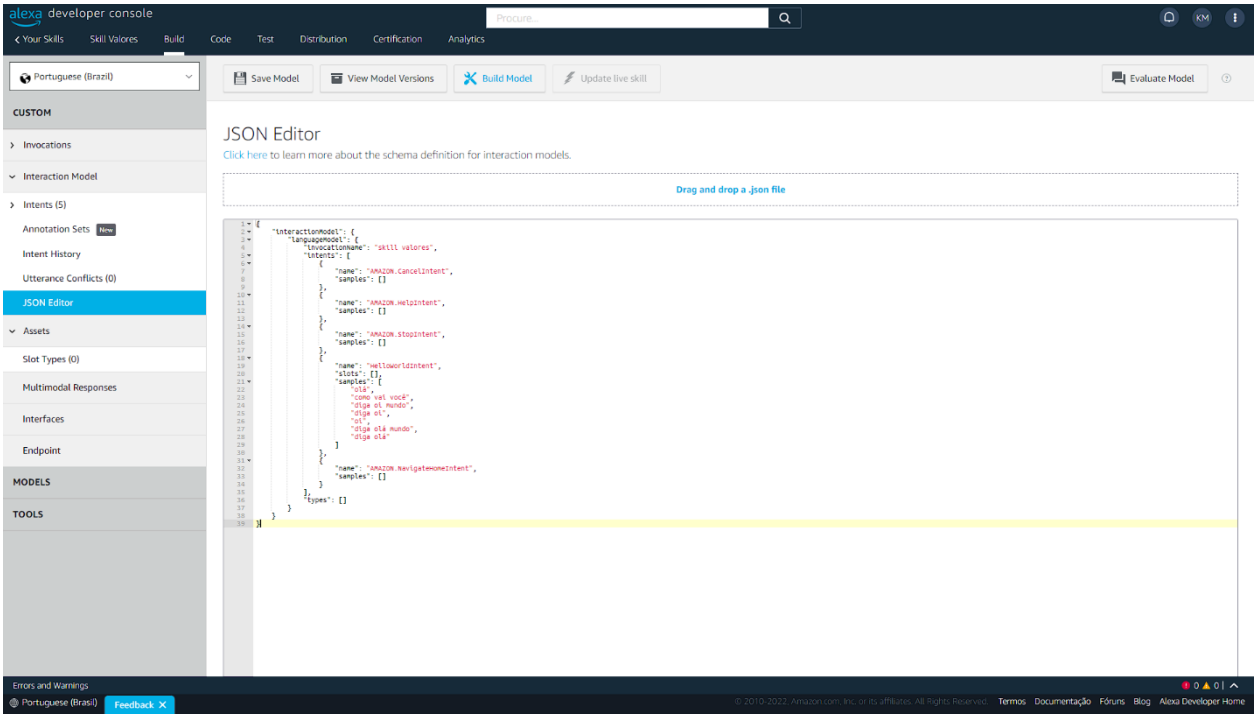


Depois da criação da skill, abra o **Invocation Name** e certificar se no **Skill Invocation Name** está *skill valores*.





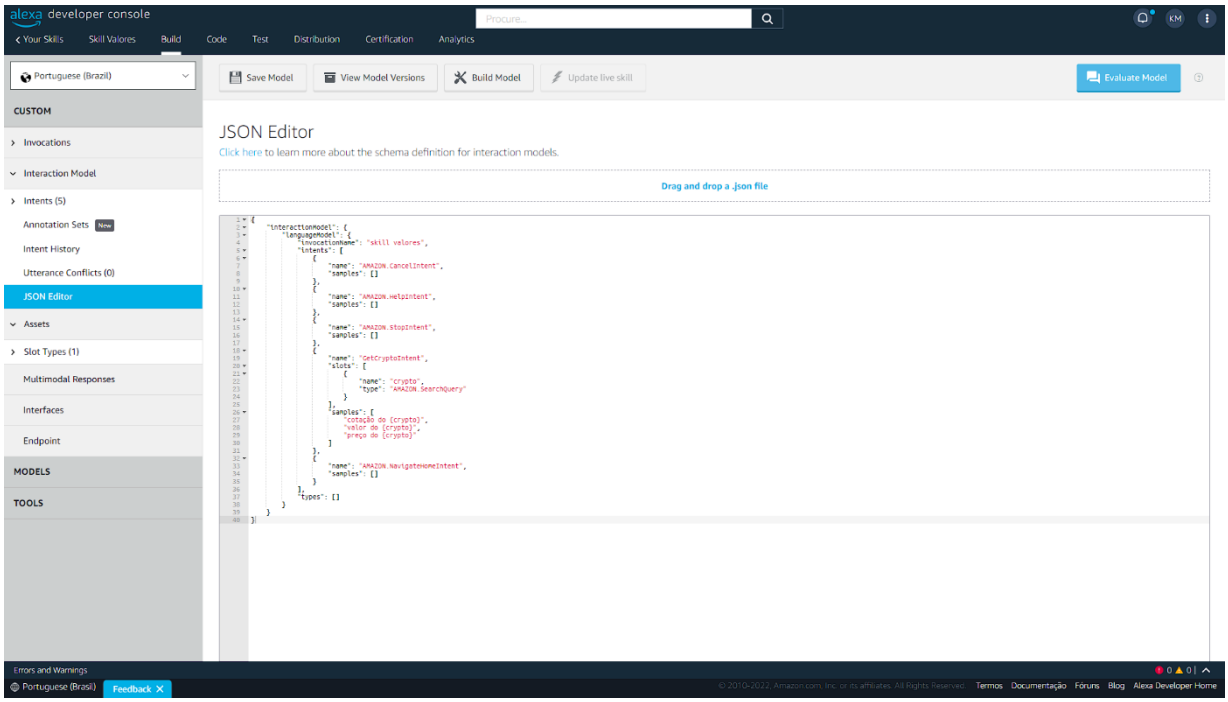
No painel ao lado esquerdo apertar em **Interaction Model** e apertar em **JSON Editor.**



Copiar o seguinte código e colar no editor como mostra o exemplo e logo após salvar em **Save Model**, com o código salvo, clicar em **Build Model.**

```
{
  "interactionModel": {
    "languageModel": {
      "invocationName": "skill valores",
      "intents": [
        {
          "name": "AMAZON.CancelIntent",
          "samples": []
        },
        {
          "name": "AMAZON.HelpIntent",
          "samples": []
        },
        {
          "name": "AMAZON.StopIntent",
          "samples": []
```

```
      },
      {
        "name": "GetCryptoIntent",
        "slots": [
          {
            "name": "crypto",
            "type": "AMAZON.SearchQuery"
          }
        ],
        "samples": [
          "cotação do {crypto}",
          "valor do {crypto}",
          "preço do {crypto}"
        ]
      },
      {
        "name": "AMAZON.NavigateHomeIntent",
        "samples": []
      }
    ],
    "types": []
  }
 },
 "version": "7"
}
```



Vá para a aba **Code**, copie o seguinte código e cole em **index.js**.

```
/* *
 * This sample demonstrates handling intents from an Alexa skill using the Alexa Skills Kit SDK (v2).
 * Please visit https://alexa.design/cookbook for additional examples on implementing slots, dialog management,
 * session persistence, api calls, and more.
 * */
const Alexa = require('ask-sdk-core');


const LaunchRequestHandler = {
  canHandle(handlerInput) {
    return Alexa.getRequestType(handlerInput.requestEnvelope) === 'LaunchRequest';
  },
  handle(handlerInput) {
    const speakOutput = 'Qual criptomoeda deseja cotação?';


    return handlerInput.responseBuilder
      .speak(speakOutput)
```

```
            .reprompt(speakOutput)
            .getResponse();
        }
    };

    const GetCryptoIntentHandler = {
        canHandle(handlerInput) {
            return Alexa.getRequestType(handlerInput.requestEnvelope) === 'IntentRequest'
                && Alexa.getIntentName(handlerInput.requestEnvelope) === 'GetCryptoIntent';
        },
        handle(handlerInput) {

            const crypto = handlerInput.requestEnvelope.request.intent.slots.crypto.value;

            const axios = require('axios');

            return axios.get(`https://api.binance.com/api/v3/avgPrice?symbol=${crypto.toUpperCase()}USDT`)
                .then(response => {
                    const price = parseFloat(response.data.price).toFixed(2).replace(".", ",");

                    const speakOutput = `O preço de ${crypto} atualmente é $${price}`;

                    return handlerInput.responseBuilder
                        .speak(speakOutput)
                        .getResponse();
                })
```

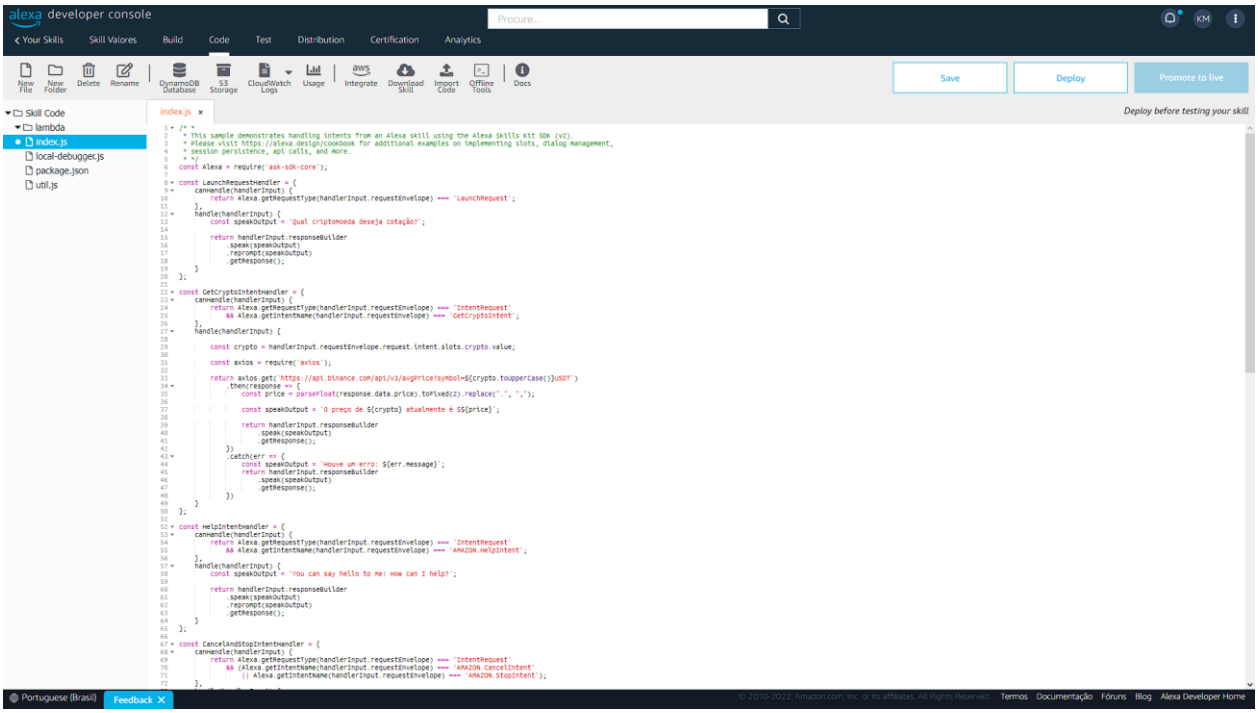
```
            .catch(err => {
                const speakOutput = `Houve um erro: ${err.message}`;
                return handlerInput.responseBuilder
                    .speak(speakOutput)
                    .getResponse();
            })
        }
    };

    const HelpIntentHandler = {
        canHandle(handlerInput) {
            return Alexa.getRequestType(handlerInput.requestEnvelope)
=== 'IntentRequest'
                && Alexa.getIntentName(handlerInput.requestEnvelope) ===
'AMAZON.HelpIntent';
        },
        handle(handlerInput) {
            const speakOutput = 'You can say hello to me! How can I help?';

            return handlerInput.responseBuilder
                .speak(speakOutput)
                .reprompt(speakOutput)
                .getResponse();
        }
    };

    const CancelAndStopIntentHandler = {
        canHandle(handlerInput) {
            return Alexa.getRequestType(handlerInput.requestEnvelope)
=== 'IntentRequest'
                && (Alexa.getIntentName(handlerInput.requestEnvelope) ===
'AMAZON.CancelIntent'
```

```
                || Alexa.getIntentName(handlerInput.requestEnvelope) ===
'AMAZON.StopIntent');
        },
        handle(handlerInput) {
            const speakOutput = 'Goodbye!';

            return handlerInput.responseBuilder
                .speak(speakOutput)
                .getResponse();
        }
    };
    /* *
     * FallbackIntent triggers when a customer says something that
doesn’t map to any intents in your skill
     * It must also be defined in the language model (if the locale supports
it)
     * This handler can be safely added but will be ingnored in locales
that do not support it yet
     * */
    const FallbackIntentHandler = {
        canHandle(handlerInput) {
            return     Alexa.getRequestType(handlerInput.requestEnvelope)
=== 'IntentRequest'
                &&  Alexa.getIntentName(handlerInput.requestEnvelope)  ===
'AMAZON.FallbackIntent';
        },
        handle(handlerInput) {
            const speakOutput = 'Sorry, I don\'t know about that. Please try
again.';

            return handlerInput.responseBuilder
                .speak(speakOutput)
                .reprompt(speakOutput)
```

```
            .getResponse();
        }
    };
    /* *
     * SessionEndedRequest notifies that a session was ended. This handler will be triggered when a currently open
     * session is closed for one of the following reasons: 1) The user says "exit" or "quit". 2) The user does not
     * respond or says something that does not match an intent defined in your voice model. 3) An error occurs
     * */
    const SessionEndedRequestHandler = {
        canHandle(handlerInput) {
            return Alexa.getRequestType(handlerInput.requestEnvelope) === 'SessionEndedRequest';
        },
        handle(handlerInput) {
            console.log(`~~~~ Session ended: ${JSON.stringify(handlerInput.requestEnvelope)}`);
            // Any cleanup logic goes here.
            return handlerInput.responseBuilder.getResponse(); // notice we send an empty response
        }
    };
    /* *
     * The intent reflector is used for interaction model testing and debugging.
     * It will simply repeat the intent the user said. You can create custom handlers for your intents
     * by defining them above, then also adding them to the request handler chain below
     * */
    const IntentReflectorHandler = {
        canHandle(handlerInput) {
```

```
            return Alexa.getRequestType(handlerInput.requestEnvelope) === 'IntentRequest';
        },
        handle(handlerInput) {
            const intentName = Alexa.getIntentName(handlerInput.requestEnvelope);
            const speakOutput = `You just triggered ${intentName}`;

            return handlerInput.responseBuilder
                .speak(speakOutput)
                //.reprompt('add a reprompt if you want to keep the session open for the user to respond')
                .getResponse();
        }
    };
    /**
     * Generic error handling to capture any syntax or routing errors. If you receive an error
     * stating the request handler chain is not found, you have not implemented a handler for
     * the intent being invoked or included it in the skill builder below
     * */
    const ErrorHandler = {
        canHandle() {
            return true;
        },
        handle(handlerInput, error) {
            const speakOutput = 'Sorry, I had trouble doing what you asked. Please try again.';
            console.log(`~~~~ Error handled: ${JSON.stringify(error)}`);

            return handlerInput.responseBuilder
                .speak(speakOutput)
```

```
            .reprompt(speakOutput)
            .getResponse();
    }
};
exports.handler = Alexa.SkillBuilders.custom()
    .addRequestHandlers(
        LaunchRequestHandler,
        GetCryptoIntentHandler,
        HelpIntentHandler,
        CancelAndStopIntentHandler,
        FallbackIntentHandler,
        SessionEndedRequestHandler,
        IntentReflectorHandler)
    .addErrorHandlers(
        ErrorHandler)
    .withCustomUserAgent('sample/hello-world/v1.2')
    .lambda();
```



Copie o seguinte código e cole em **local-debugger.js**.

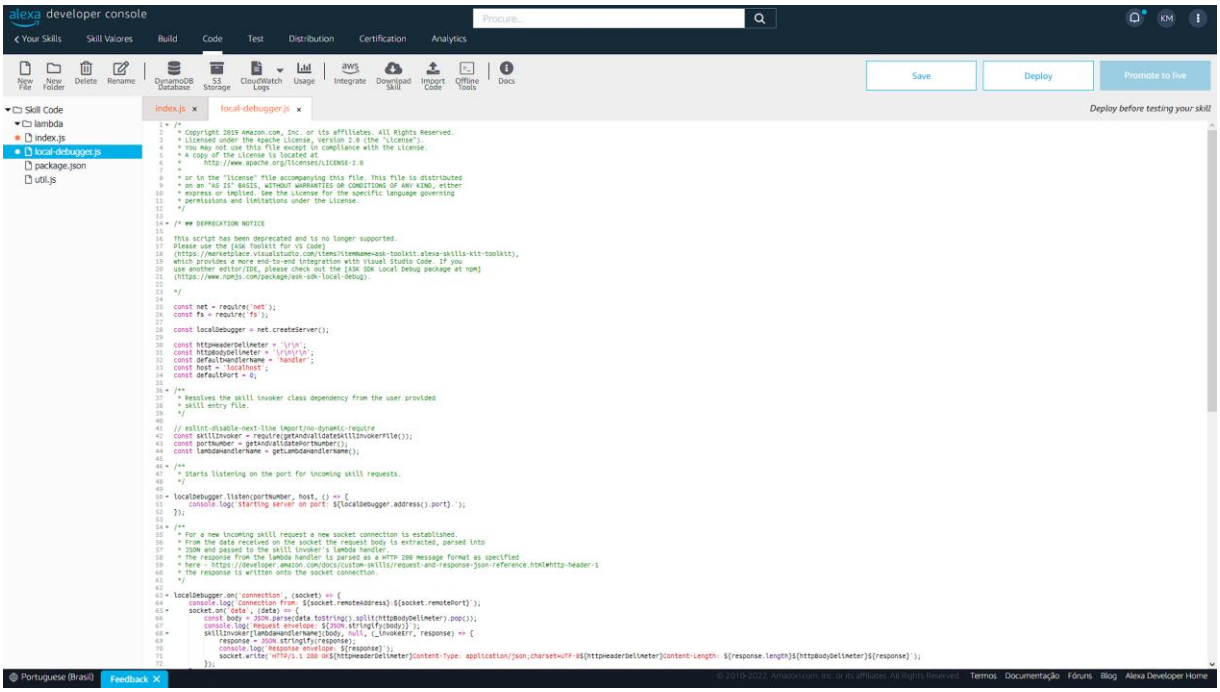
```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

/* ## DEPRECATION NOTICE

This script has been deprecated and is no longer supported.
Please use the [ASK Toolkit for VS Code]
(https://marketplace.visualstudio.com/items?itemName=ask-toolkit.alexa-skills-kit-toolkit),
which provides a more end-to-end integration with Visual Studio Code. If you
use another editor/IDE, please check out the [ASK SDK Local Debug package at npm]
(https://www.npmjs.com/package/ask-sdk-local-debug).

*/

const net = require('net');
const fs = require('fs');
```

```
const localDebugger = net.createServer();

const httpHeaderDelimeter = '\r\n';
const httpBodyDelimeter = '\r\n\r\n';
const defaultHandlerName = 'handler';
const host = 'localhost';
const defaultPort = 0;

/**
 * Resolves the skill invoker class dependency from the user provided
 * skill entry file.
 */

// eslint-disable-next-line import/no-dynamic-require
const skillInvoker = require(getAndValidateSkillInvokerFile());
const portNumber = getAndValidatePortNumber();
const lambdaHandlerName = getLambdaHandlerName();

/**
 * Starts listening on the port for incoming skill requests.
 */

localDebugger.listen(portNumber, host, () => {
    console.log(`Starting server on port: ${localDebugger.address().port}.`);
});

/**
 * For a new incoming skill request a new socket connection is established.
```

```
        * From the data received on the socket the request body is extracted, parsed into
        * JSON and passed to the skill invoker's lambda handler.
        * The response from the lambda handler is parsed as a HTTP 200 message format as specified
        * here - https://developer.amazon.com/docs/custom-skills/request-and-response-json-reference.html#http-header-1
        * The response is written onto the socket connection.
        */

      localDebugger.on('connection', (socket) => {
        console.log(`Connection from: ${socket.remoteAddress}:${socket.remotePort}`);
        socket.on('data', (data) => {
          const body = JSON.parse(data.toString().split(httpBodyDelimeter).pop());
          console.log(`Request envelope: ${JSON.stringify(body)}`);
          skillInvoker[lambdaHandlerName](body, null, (_invokeErr, response) => {
            response = JSON.stringify(response);
            console.log(`Response envelope: ${response}`);
            socket.write(`HTTP/1.1 200 OK${httpHeaderDelimeter}Content-Type: application/json;charset=UTF-8${httpHeaderDelimeter}Content-Length: ${response.length}${httpBodyDelimeter}${response}`);
          });
        });
      });

      /**
       * Validates user specified port number is in legal range [0, 65535].
       * Defaults to 0.
       */
```

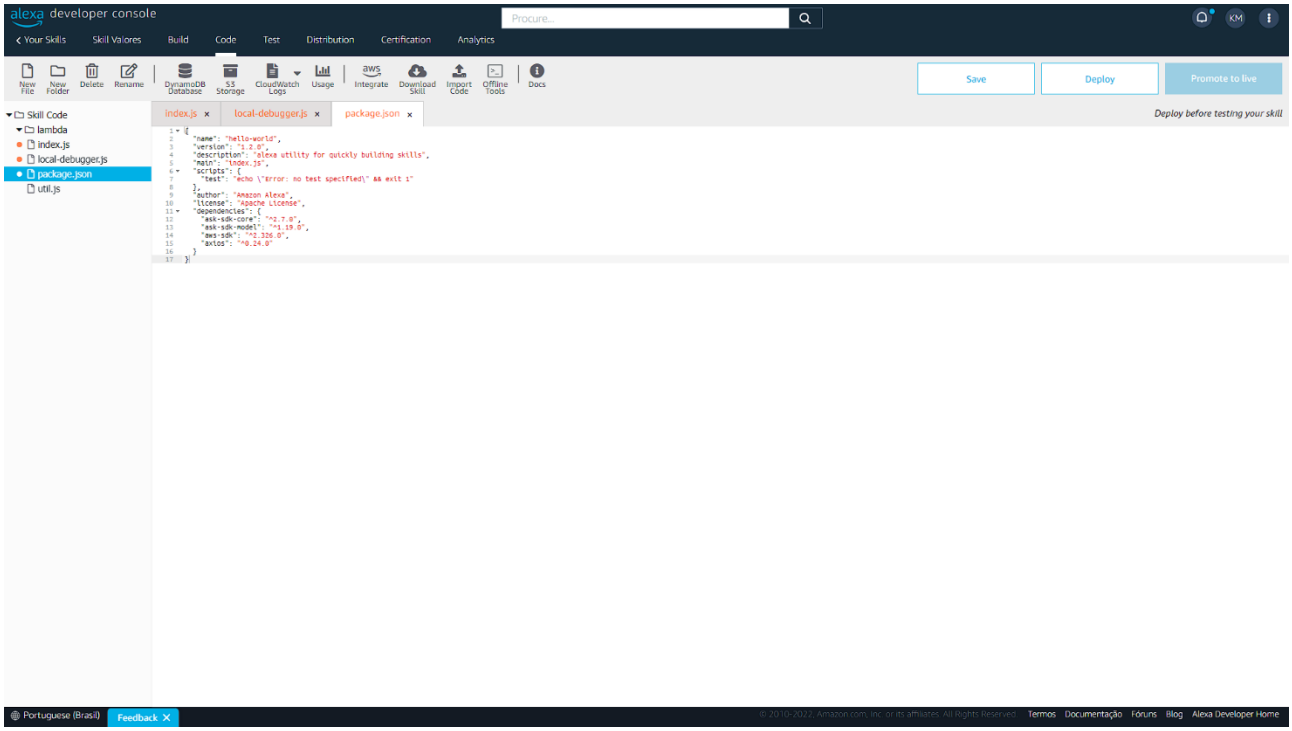
```
function getAndValidatePortNumber() {
  const portNumberArgument = Number(getArgument('portNumber', defaultPort));
  if (!Number.isInteger(portNumberArgument)) {
    throw new Error(`Port number has to be an integer - ${portNumberArgument}.`);
  }
  if (portNumberArgument < 0 || portNumberArgument > 65535) {
    throw new Error(`Port out of legal range: ${portNumberArgument}. The port number should be in the range [0, 65535]`);
  }
  if (portNumberArgument === 0) {
    console.log('The TCP server will listen on a port that is free.'
    + 'Check logs to find out what port number is being used');
  }
  return portNumberArgument;
}

/**
 * Gets the lambda handler name.
 * Defaults to "handler".
 */

function getLambdaHandlerName() {
  return getArgument('lambdaHandler', defaultHandlerName);
}

/**
 * Validates that the skill entry file exists on the path specified.
 * This is a required field.
 */
```

```
// eslint-disable-next-line consistent-return
function getAndValidateSkillInvokerFile() {
  const fileNameArgument = getArgument('skillEntryFile');
  if (!fs.existsSync(fileNameArgument)) {
    throw new Error(`File not found: ${fileNameArgument}`);
  }
  return fileNameArgument;
}

/**
 * Helper function to fetch the value for a given argument
 * @param {argumentName} argumentName name of the argument for which the value needs to be fetched
 * @param {defaultValue} defaultValue default value of the argument that is returned if the value doesn't exist
 */

function getArgument(argumentName, defaultValue) {
  const index = process.argv.indexOf(`--${argumentName}`);
  if (index === -1 || typeof process.argv[index + 1] === 'undefined') {
    if (defaultValue === undefined) {
      throw new Error(`Required argument - ${argumentName} not provided.`);
    } else {
      return defaultValue;
    }
  }
  return process.argv[index + 1];
}
```



Copie o seguinte código e cole em **package.json.**

```
{
  "name": "hello-world",
  "version": "1.2.0",
  "description": "alexa utility for quickly building skills",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Amazon Alexa",
  "license": "Apache License",
  "dependencies": {
    "ask-sdk-core": "^2.7.0",
    "ask-sdk-model": "^1.19.0",
    "aws-sdk": "^2.326.0",
    "axios": "^0.24.0"
  }
}
```



Copie o seguinte código e cole em **útil.js.**

const AWS = require('aws-sdk');

const s3SigV4Client = new AWS.S3({
signatureVersion: 'v4',
region: process.env.S3_PERSISTENCE_REGION
});

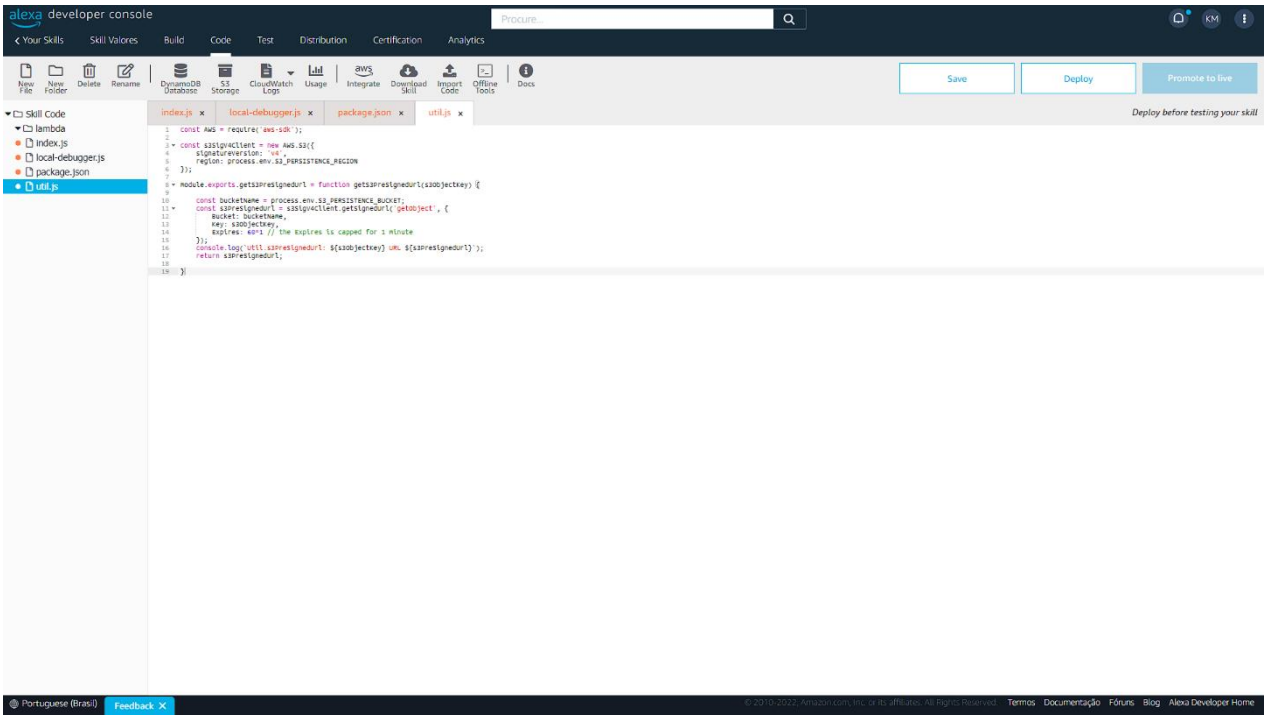
module.exports.getS3PreSignedUrl = function getS3PreSignedUrl(s3ObjectKey) {

const bucketName = process.env.S3_PERSISTENCE_BUCKET;
const s3PreSignedUrl = s3SigV4Client.getSignedUrl('getObject', {
Bucket: bucketName,
Key: s3ObjectKey,
Expires: 60*1 // the Expires is capped for 1 minute
});

console.log(`Util.s3PreSignedUrl: ${s3ObjectKey} URL ${s3PreSignedUrl}`);

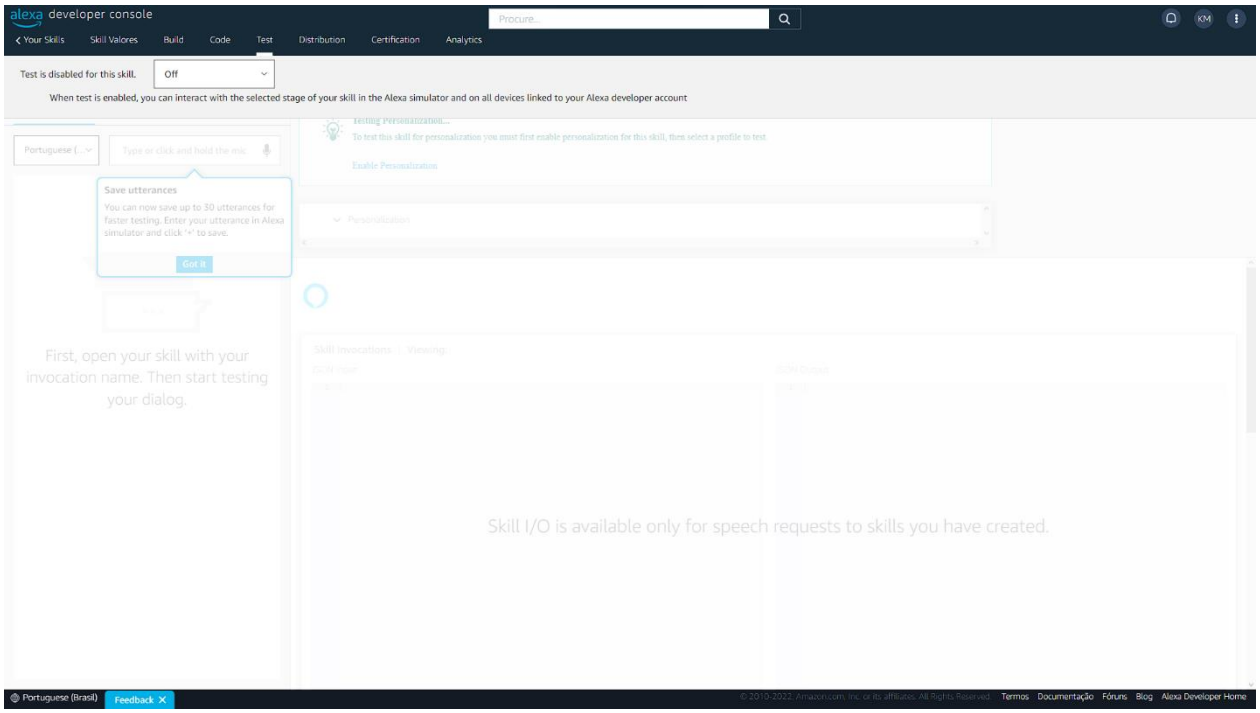
return s3PreSignedUrl;

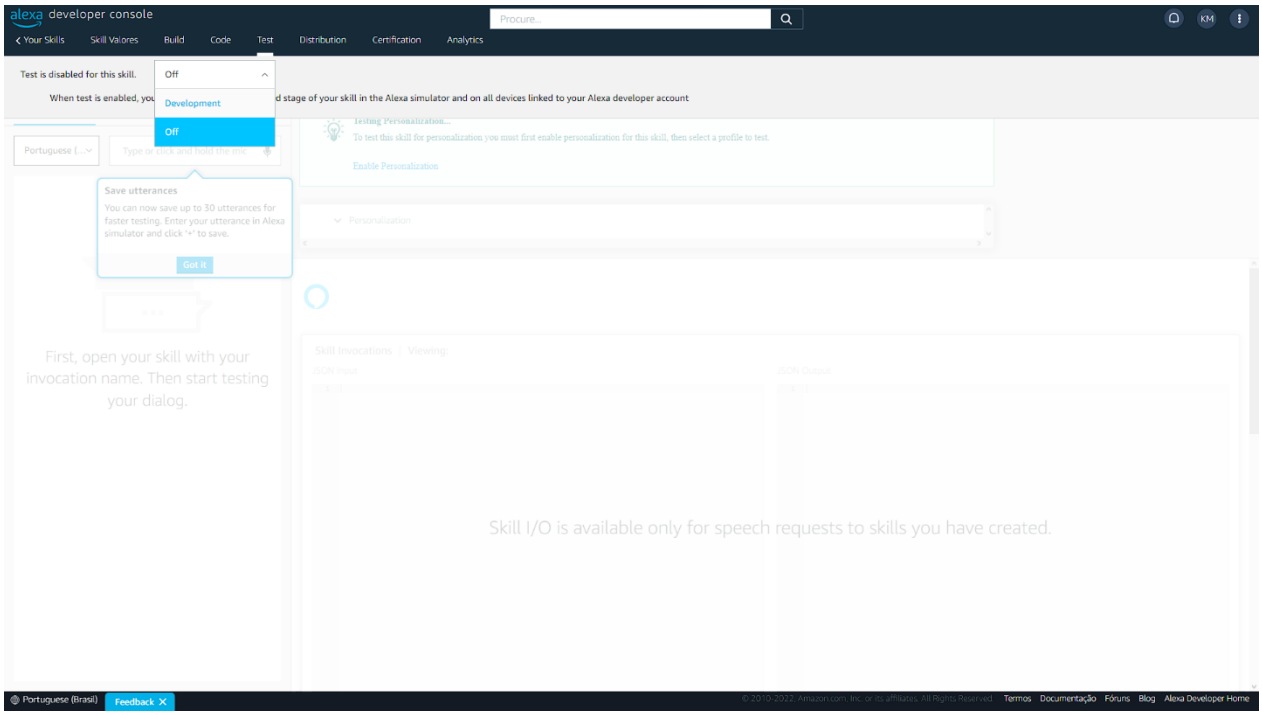
}

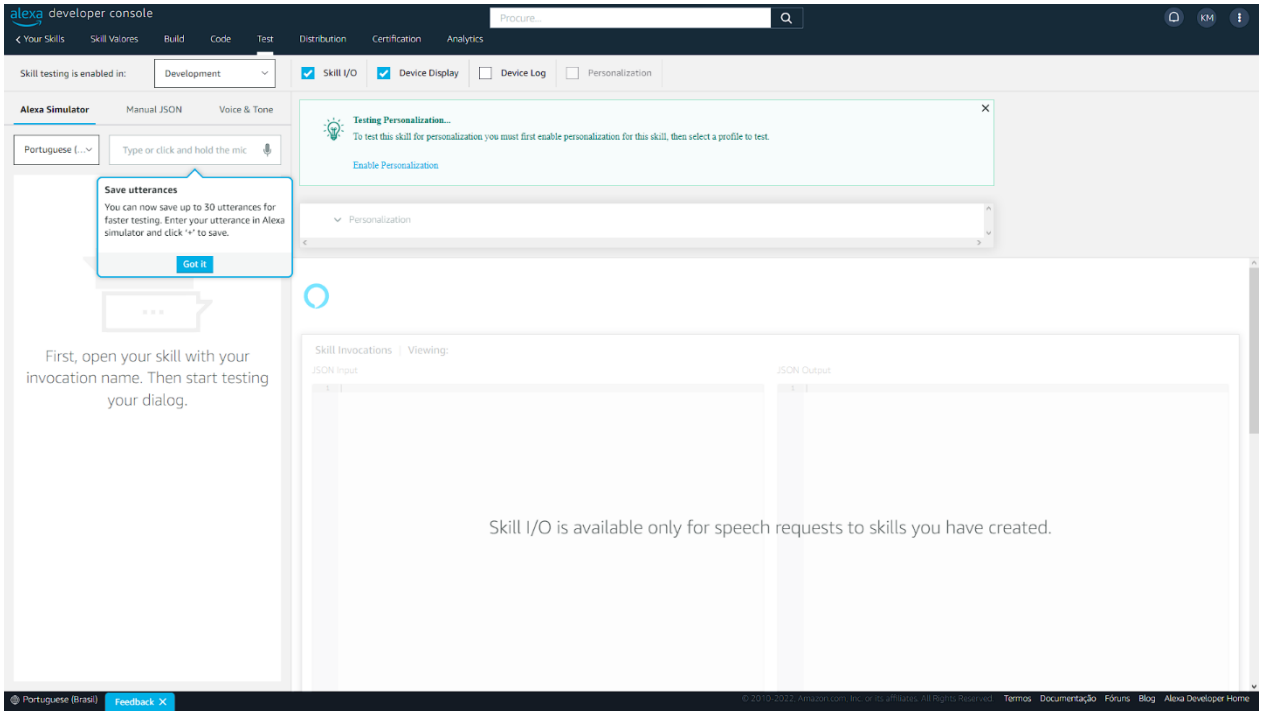


Após todos códigos inseridos salve todos em cada aba no botão **Save** e logo após **Deploy**.

Após isso a skill já pode ser testada acessando a aba **Test**.



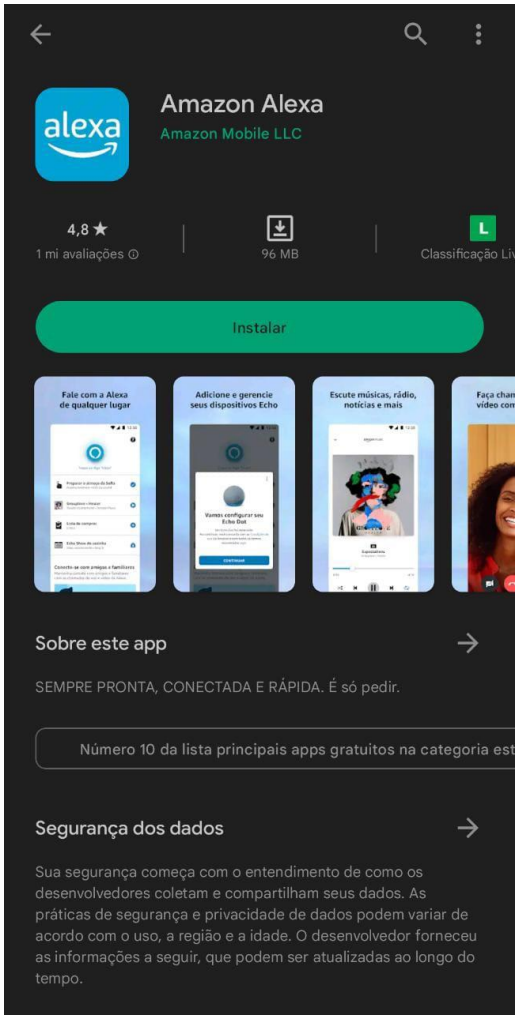




3. UTILIZANDO NA ALEXA

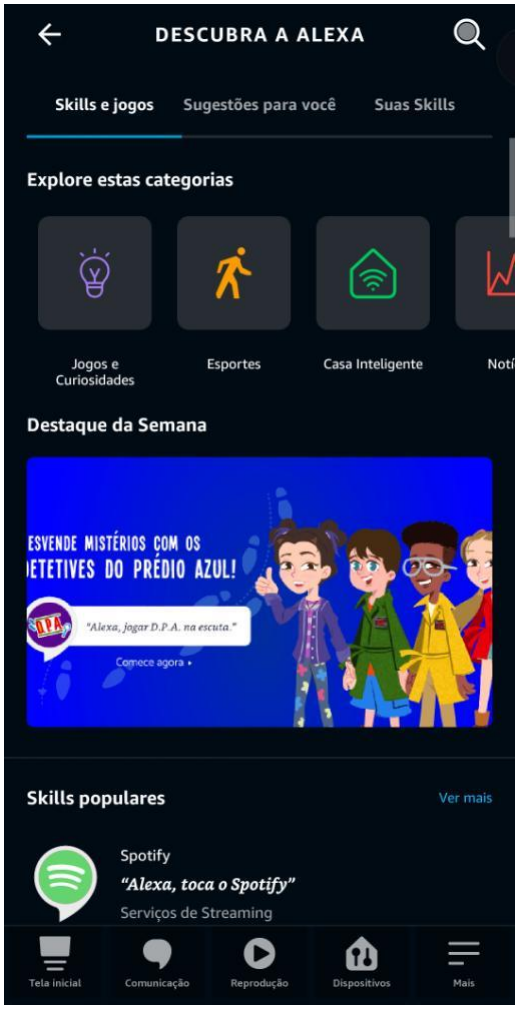
Caso tenha uma Alexa em casa e queira testa a *skill* nela siga as seguintes instruções: (*Vale lembrar que o e-mail cadastrado no console tem que ser o mesmo que usado na sua Alexa.)*

Primeiramente abra o aplicativo **Amazon Alexa**.

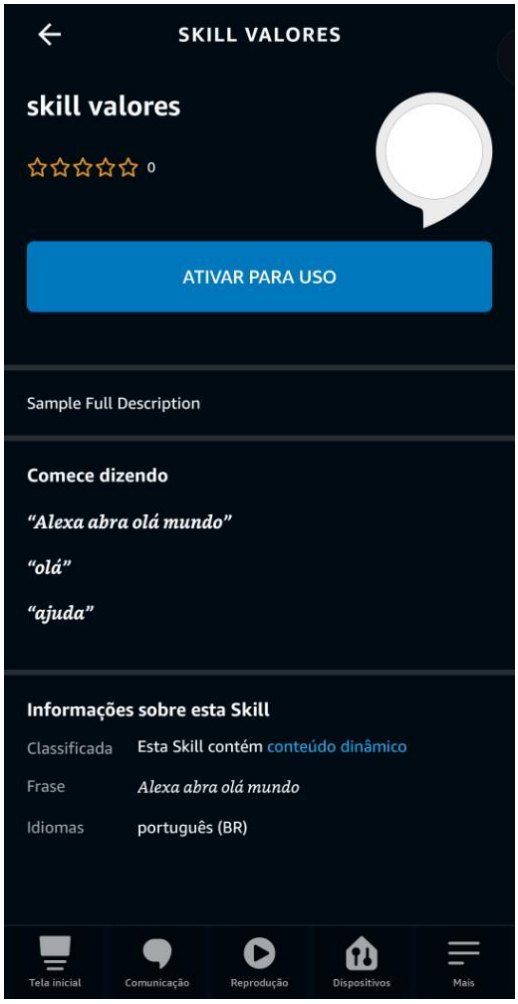


Siga para a aba **Mais**.

Aperte em **Skills e jogos**, logo após em **Suas skills**.



desenv. E aperte na **skill valores**.



E aperte em **ATIVAR PARA O USO**.