```
'use strict';
const Alexa = require('ask-sdk');
const Util = require('./util');
const aplMainTemplate = require('./apl-main.json');
/***********
Data: Customize the data below as you please.
***********/


const SKILL_NAME = "Inner Beast Personality Quiz";
const HELP_MESSAGE_BEFORE_START = "Welcome to Inner Beast! Answer these few questions, and I will tell you what animal you have lurking within ready to escape! Are you ready to play?";
const HELP_MESSAGE_AFTER_START = "Just respond with yes or no and I'll give you the result in the end.";
const HELP_REPROMPT = "Your animal will be revealed after you answer all my yes or no questions.";
const STOP_MESSAGE = "Your Inner Beast will be waiting for you next time.";
const MISUNDERSTOOD_INSTRUCTIONS_ANSWER = "Please answer with either yes or no.";
const HINT_TEXT = `To play again, just say "Alexa, open ${SKILL_NAME}"`


const BACKGROUND_IMAGE_URL =
“https://docs.google.com/drawings/d/1ErGwcCO3HIMk67wwMBrSeIL9fJyEqLUObersZVQH2Bs/edit?usp=sharing”;
// const BACKGROUND_IMAGE_URL =  "default.jpg";
const BACKGROUND_GOODBYE_IMAGE_URL =
“https://docs.google.com/drawings/d/1a1eJiijg1iy2QoMXUh2-tOQr4fSLYO-qitq9ac67ao0/edit?usp=sharing”;
// const BACKGROUND_GOODBYE_IMAGE_URL = "goodbye.jpg";
const BACKGROUND_HELP_IMAGE_URL =
“https://docs.google.com/drawings/d/1WkW9koXB7tNy3sDYuKrF7GU9zX-FXZH9GEEO97LRBQk/edit?usp=sharing”
// const BACKGROUND_HELP_IMAGE_URL = "help.jpg";

const WELCOME_MESSAGE = "Hi! I can tell you what animal you're most like. All you have to do is answer a few questions with either yes or no. Are you ready to start?";
const INITIAL_QUESTION_INTROS = [
  "Great! Let's start!",
  "<say-as interpret-as='interjection'>Alrighty</say-as>! Here comes your first question!",
  "Ok lets go. <say-as interpret-as='interjection'>Ahem</say-as>.",
  "<say-as interpret-as='interjection'>well well</say-as>."
];
```

```
const QUESTION_INTROS = [
  "Oh dear.",
  "Okey Dokey",
  "You go, human!",
  "Sure thing.",
  "I would have said that too.",
  "Of course.",
  "I knew it.",
  "Totally agree.",
  "So true.",
  "I agree."
];
const UNDECISIVE_RESPONSES = [
  "<say-as interpret-as='interjection'>Honk</say-as>. I'll just choose for you.",
  "<say-as interpret-as='interjection'>Nanu Nanu</say-as>. I picked an answer for you.",
  "<say-as interpret-as='interjection'>Uh oh</say-as>... well nothing I can do about that.",
  "<say-as interpret-as='interjection'>Aha</say-as>. We will just move on then.",
  "<say-as interpret-as='interjection'>Aw man</say-as>. How about this question?",
];
const RESULT_MESSAGE = "Here comes the big reveal! You are "; // the name of the result is
inserted here.
const RESULT_MESSAGE_SHORT = "You are "; // the name of the result is inserted here.
const PLAY_AGAIN_REQUEST = "That was it. Do you want to play again?";

const resultList = {
  result1: {
    name: "An Australian Shepherd",
    display_name: "Australian Shepherd",
    audio_message: "Mans best friend and most loyal companion.",
    description: "Not only are you as loyal as they come. You protect yours and those being
picked on by the wolves of the world.. You're intelligent, flexible, and courageous. You always
put others first and give unyielding love and support no matter the situation. You are the best of
the best . The bravest of the brave. You wear your heart on your sleeve and don't hesitate to let
others know how you are feeling.",
    img: {
      largeImageUrl:
"https://docs.google.com/drawings/d/12tiSPgqDgrEMUvkJg7Znhi7p2WNJNDE5rFckibfnvQ0/edit
?usp=sharing"
      //largeImageUrl: "result1.jpg",
    }
  },
  result2: {
    name: "Lightning Bug",
    display_name: "Lightning Bug",
```

```
    audio_message: "Are you the center of attention because of your magic, or because you need to be?",
    description: "Always seeking the limelight. You’re special and as much as you try not to acknowledge it, you know it and you know other’s know it. You make up for being the center of attention by making sure everyone gets theirs.",
    img: {
      largeImageUrl: "https://docs.google.com/drawings/d/13uIprGH87N5B4ZY2VqoPHKf0arJMK0UWK88PFjpUt1U/edit?usp=sharing"
      //largeImageUrl: "result2.jpg",
    }
  },
  result3: {
    name: "tiger",
    display_name: "Tiger",
    audio_message: "Pure Muscle, carried with enviable prestige.",
    description: "While you use your stealth and observation of others to your advantage, you tend to walk your own path. You aren’t picky with what’s on your plate but you are picky about your muscle mass. You carry yourself with enviable prestige.",
    img: {
      largeImageUrl: “https://docs.google.com/drawings/d/1gz-IpfDZkotiazyfQ2E1Zev2HwEdBzCUdO1T4G-ha1U/edit?usp=sharing"
      //largeImageUrl: "result3.jpg",
    }
  },
  result4: {
    name: "A sea turtle",
    display_name: "Sea Turtle",
    audio_message: "You’re Wise, and not because you’re one hundred and one.",
    description: "While on occasion you may mistake a plastic bag for a jellyfish, you never mistake a foe for a loyal friend in your tribe. Others look to you for the best advice, because you are wise beyond your years, and you have an amazing sense of direction. Your chill, easy going vibe attract the best people to your circle.",
    img: {
      largeImageUrl: "https://docs.google.com/drawings/d/1M7mVDlaZKu9Z6_5nGn4nP-0GRFZBFYukQ9o67LvKGu4/edit?usp=sharing"
      //largeImageUrl: "result4.jpg",
    }
  },
  result5: {
    name: "Hippopotamus",
    display_name: "Hippopotamus",
```

```
    audio_message: "You’re a tank who loves the sun and the water and your tribe",
    description: "Your body requires a lot of sleep to be able to get your day to day tasks done. You prefer comfort over chaos but aren’t afraid to get your hands dirty if necessary. You are unyieldingly loyal to those you consider your own and prefer to keep them close where you can watch over them.You may come off as shy and quiet but you know just the right way to get to someone who has done you wrong and this catches them off guard every time!",
    img: {
      largeImageUrl: "https://docs.google.com/drawings/d/1wAYGWZTOILLpwlBasISMF97MWqP_AsdC37hRkgEx9II/edit?usp=sharing"
      //largeImageUrl: "result5.jpg",
    }
  }
};

const questions = [{
    question: "Have you ever performed in front of a crowd?",
    questionDisplay: "Have you ever performed in front of a crowd?",
    background: "https://docs.google.com/drawings/d/1KgspAKKEcdi6-RwHB9hK_4TS6D2RVnd9zNKO5M6nQ2c/edit?usp=sharing",
    //background: "question1.jpg",
    points: {
      result1: 4,
      result2: 0,
      result3: 5,
      result4: 3,
      result5: 1
    }
  },
  {
    question: "Do you workout daily?",
    questionDisplay: "Is working out a part of your daily routine?",
    background: "https://docs.google.com/drawings/d/10dOXIDXMqJnmNme7s_6oSd7s1LxwtgOPrZBGrluxe2M/edit?usp=sharing",
    points: {
      result1: 4,
      result2: 1,
      result3: 2,
      result4: 3,
      result5: 5
    }
  },
```

```
  {
    question: "Do you enjoy reading a good book more than going out to a party?",
    questionDisplay: "Do you enjoy a book more than a party?",
    background: "https://s3.amazonaws.com/coach-courses-us/public/courses/voice/2.7/q3.jpg",
    //background: "question3.jpg",
    points: {
      result1: 0,
      result2: 5,
      result3: 1,
      result4: 3,
      result5: 4
    }
  },
  {
    question: "Are you up all night?",
    questionDisplay: "Are you a night owl?",
    background:
"https://docs.google.com/drawings/d/1IHmkNtfGMH86Uqpm2lxGOHWMNGJJfYwiWjhcC4IBh54
/edit?usp=sharing",
    points: {
      result1: 2,
      result2: 3,
      result3: 4,
      result4: 4,
      result5: 5
    }
  },
  {
    question: "Do you sleep for more than 9 hours a day?",
    questionDisplay: "Do you sleep for 9 plus hours a day?",
    background:
"https://docs.google.com/drawings/d/1seeCt6wQvRYs5I5JcHkVmHzy5eEncuof9G4rqWmco_U/
edit?usp=sharing",
    points: {
      result1: 0,
      result2: 5,
      result3: 3,
      result4: 4,
      result5: 5
    }
  }
];

/**********
```

Execution Code: Avoid editing the code below if you don't know JavaScript.
**********/

// Private methods (this is the actual code logic behind the app)

```
const newSessionHandler = {
  canHandle(handlerInput) {
    const request = handlerInput.requestEnvelope.request;
    const attributesManager = handlerInput.attributesManager;
    const sessionAttributes = attributesManager.getSessionAttributes();
    var isCurrentlyPlayingOrGettingResult = false;
    if (sessionAttributes.state) {
      isCurrentlyPlayingOrGettingResult = true;
    }


    return handlerInput.requestEnvelope.request.type === `LaunchRequest` ||
(!isCurrentlyPlayingOrGettingResult && request.type === 'IntentRequest' &&
(request.intent.name === 'AMAZON.YesIntent' || request.intent.name === 'AMAZON.NoIntent'));
  },
  handle(handlerInput) {

    console.log('------------------------------------ New session')
    const request = handlerInput.requestEnvelope.request;
    const attributesManager = handlerInput.attributesManager;
    const sessionAttributes = attributesManager.getSessionAttributes();
    if (handlerInput.requestEnvelope.request.type === `LaunchRequest`) {
      _initializeApp(sessionAttributes);
      return buildResponse(handlerInput, WELCOME_MESSAGE, WELCOME_MESSAGE,
SKILL_NAME);
    }
    if (request.type === 'IntentRequest' && request.intent.name === 'AMAZON.YesIntent') {

      sessionAttributes.state = states.QUIZMODE;

      const systemSpeak = _nextQuestionOrResult(handlerInput);
      return buildResponse(handlerInput, systemSpeak.prompt, systemSpeak.reprompt,
SKILL_NAME, systemSpeak.background,systemSpeak.displayText);
    }
    if (request.type === 'IntentRequest' && request.intent.name === 'AMAZON.NoIntent') {
      console.log('------------------------------------ Exit session')
      const attributesManager = handlerInput.attributesManager;
```

```
      const sessionAttributes = attributesManager.getSessionAttributes();
      sessionAttributes.state = '';
      return buildResponse(handlerInput, STOP_MESSAGE, '', SKILL_NAME,
BACKGROUND_GOODBYE_IMAGE_URL,STOP_MESSAGE);
    }
    if (request.type === 'IntentRequest' && request.intent.name === 'UndecisiveIntent') {
      const outputSpeech = MISUNDERSTOOD_INSTRUCTIONS_ANSWER;
      return buildResponse(handlerInput, outputSpeech, outputSpeech, SKILL_NAME,
BACKGROUND_IMAGE_URL,"");
    }
  },
};

const nextQuestionIntent = (handlerInput, prependMessage) => {
  const attributesManager = handlerInput.attributesManager;
  const sessionAttributes = attributesManager.getSessionAttributes();
  sessionAttributes.questionProgress++;
  var currentQuestion = questions[sessionAttributes.questionProgress].question;
  return {
    prompt: `${prependMessage} ${_randomQuestionIntro(sessionAttributes)}
${currentQuestion}`,
    reprompt: HELP_MESSAGE_AFTER_START,
    displayText: questions[sessionAttributes.questionProgress].questionDisplay,
    background: questions[sessionAttributes.questionProgress].background
  };
}

const resultIntent = (handlerInput, prependMessage) => {
  const attributesManager = handlerInput.attributesManager;
  const sessionAttributes = attributesManager.getSessionAttributes();
  const resultPoints = sessionAttributes.resultPoints;
  const result = Object.keys(resultPoints).reduce((o, i) => resultPoints[o] > resultPoints[i] ? o : i);
  const resultMessage = `${prependMessage} ${RESULT_MESSAGE}
${resultList[result].name}. ${resultList[result].audio_message}. ${PLAY_AGAIN_REQUEST}`;
  return {
    prompt: resultMessage,
    reprompt: PLAY_AGAIN_REQUEST,
    displayText: `${RESULT_MESSAGE_SHORT} ${resultList[result].name}`,
    background: resultList[result].img.largeImageUrl
  }

  // this.emit(':askWithCard', resultMessage, PLAY_AGAIN_REQUEST,
resultList[result].display_name, resultList[result].description, resultList[result].img);
```

```
  //                    ^speechOutput  ^repromptSpeech     ^cardTitle                    ^cardContent
^imageObj
}

const RepeatIntentHandler = {
  canHandle(handlerInput) {
   return Alexa.getRequestType(handlerInput.requestEnvelope) ===   'IntentRequest' &&
Alexa.getIntentName(handlerInput.requestEnvelope) === 'AMAZON.RepeatIntent';
   },
handle(handlerInput) {
    // Get the session attributes.
    const sessionAttributes =
    handlerInput.attributesManager.getSessionAttributes();
    console.log('Repeat');
    console.log(sessionAttributes.lastPrompt);
   return          buildResponse (handlerInput, sessionAttributes.lastPrompt,
sessionAttributes.lastReprompt, sessionAttributes.lastTitle, sessionAttributes.lastImage_url,
sessionAttributes.lastDisplayText, sessionAttributes.lastDisplay_type)
  }
};

const _randomIndexOfArray = (array) => Math.floor(Math.random() * array.length);
const _randomOfArray = (array) => array[_randomIndexOfArray(array)];
const _adder = (a, b) => a + b;
const _subtracter = (a, b) => a - b;

// Handle user input and intents:

const states = {
  QUIZMODE: "_QUIZMODE",
  RESULTMODE: "_RESULTMODE"
}



const quizModeHandler = {
  canHandle(handlerInput) {

    const attributesManager = handlerInput.attributesManager;
    const sessionAttributes = attributesManager.getSessionAttributes();
    var isCurrentlyPlaying = false;
    if (sessionAttributes.state && sessionAttributes.state === states.QUIZMODE) {
      isCurrentlyPlaying = true;
    }
```

```
    return isCurrentlyPlaying;
  },
  handle(handlerInput) {
    console.log('---------------------------------------Quiz Mode')
    const request = handlerInput.requestEnvelope.request;
    const attributesManager = handlerInput.attributesManager;
    const sessionAttributes = attributesManager.getSessionAttributes();
    var prependMessage = '';
    if (request.type === 'IntentRequest' && request.intent.name === 'AMAZON.NextIntent') {
      const systemSpeak = nextQuestionIntent(handlerInput, prependMessage);
      return buildResponse(handlerInput, systemSpeak.prompt, systemSpeak.reprompt,
SKILL_NAME, systemSpeak.background,systemSpeak.displayText);
    }

    if (request.type === 'IntentRequest' && request.intent.name === 'AMAZON.YesIntent') {
      _applyresultPoints(sessionAttributes, _adder);
      sessionAttributes.state = states.QUIZMODE;
      const systemSpeak = _nextQuestionOrResult(handlerInput);
      return buildResponse(handlerInput, systemSpeak.prompt, systemSpeak.reprompt,
SKILL_NAME, systemSpeak.background,systemSpeak.displayText);
    }

    if (request.type === 'IntentRequest' && request.intent.name === 'AMAZON.NoIntent') {
      _applyresultPoints(sessionAttributes, _subtracter);
      sessionAttributes.state = states.QUIZMODE;
      const systemSpeak = _nextQuestionOrResult(handlerInput);
      return buildResponse(handlerInput, systemSpeak.prompt, systemSpeak.reprompt,
SKILL_NAME, systemSpeak.background,systemSpeak.displayText);
    }

    if (request.type === 'IntentRequest' && request.intent.name === 'UndecisiveIntent') {
      Math.round(Math.random()) ? _applyresultPoints(sessionAttributes, _adder) :
_applyresultPoints(sessionAttributes, _subtracter);
      const systemSpeak = _nextQuestionOrResult(handlerInput,
_randomOfArray(UNDECISIVE_RESPONSES));
      return buildResponse(handlerInput, systemSpeak.prompt, systemSpeak.reprompt,
SKILL_NAME, systemSpeak.background,systemSpeak.displayText);
    }

  if (request.type === 'IntentRequest' && request.intent.name === 'AMAZON.RepeatIntent') {
        console.log('Repeat');
           console.log(sessionAttributes.lastPrompt);
```

```
            return          buildResponse (handlerInput, sessionAttributes.lastPrompt,
sessionAttributes.lastReprompt, sessionAttributes.lastTitle, sessionAttributes.lastImage_url,
sessionAttributes.lastDisplayText, sessionAttributes.lastDisplay_type)
          }


  },
};

const resultModeHandler = {
  canHandle(handlerInput) {
    const request = handlerInput.requestEnvelope.request;
    const attributesManager = handlerInput.attributesManager;
    const sessionAttributes = attributesManager.getSessionAttributes();
    var isCurrentlyPlaying = false;
    if (sessionAttributes.state &&
      sessionAttributes.state === states.QUIZMODE) {
      isCurrentlyPlaying = true;
    }

    return !isCurrentlyPlaying && request.type === 'IntentRequest' && sessionAttributes.state
=== states.RESULTMODE;
  },
  handle(handlerInput) {
    console.log('------------------------------------ Result mode')
    const request = handlerInput.requestEnvelope.request;
    const attributesManager = handlerInput.attributesManager;
    const sessionAttributes = attributesManager.getSessionAttributes();

    if (request.type === 'IntentRequest' && request.intent.name === 'AMAZON.YesIntent') {
      _initializeApp(sessionAttributes);
      sessionAttributes.state = states.QUIZMODE;
      const systemSpeak = _nextQuestionOrResult(handlerInput);
      return buildResponse(handlerInput, systemSpeak.prompt, systemSpeak.reprompt,
SKILL_NAME, systemSpeak.background, systemSpeak.displayText);
    }
    if (request.type === 'IntentRequest' && request.intent.name === 'AMAZON.NoIntent') {
      const attributesManager = handlerInput.attributesManager;
      const sessionAttributes = attributesManager.getSessionAttributes();
      sessionAttributes.state = '';
      return buildResponse(handlerInput, STOP_MESSAGE, '', SKILL_NAME,
BACKGROUND_GOODBYE_IMAGE_URL,STOP_MESSAGE);

    }
```

```
  if (request.type === 'IntentRequest' && request.intent.name === 'AMAZON.RepeatIntent') {
        console.log('Repeat');
           console.log(sessionAttributes.lastPrompt);
              return          buildResponse (handlerInput, sessionAttributes.lastPrompt,
sessionAttributes.lastReprompt, sessionAttributes.lastTitle, sessionAttributes.lastImage_url,
sessionAttributes.lastDisplayText, sessionAttributes.lastDisplay_type)
           }



  },
};


const ExitHandler = {
  canHandle(handlerInput) {
    const request = handlerInput.requestEnvelope.request;
    return request.type === 'IntentRequest' &&
      (request.intent.name === 'AMAZON.CancelIntent' ||
        request.intent.name === 'AMAZON.StopIntent');
  },
  handle(handlerInput) {
    console.log('------------------------------------ Exit session')
    const attributesManager = handlerInput.attributesManager;
    const sessionAttributes = attributesManager.getSessionAttributes();
    sessionAttributes.state = '';
    return buildResponse(handlerInput, STOP_MESSAGE, '', SKILL_NAME,
BACKGROUND_GOODBYE_IMAGE_URL,STOP_MESSAGE);
    //---------------------------------------------
  },
};

const HelpIntentHandler = {
  canHandle(handlerInput) {
    const request = handlerInput.requestEnvelope.request;
    return request.type === 'IntentRequest' &&
      request.intent.name === 'AMAZON.HelpIntent';
  },
  handle(handlerInput) {
    console.log('------------------------------------ Help')
    const attributesManager = handlerInput.attributesManager;
    var speechOutput = '';
    const sessionAttributes = attributesManager.getSessionAttributes();
```

```
    if (sessionAttributes.state === states.QUIZMODE) {
      speechOutput = HELP_MESSAGE_AFTER_START;
    } else {
      speechOutput = HELP_MESSAGE_BEFORE_START;
    }
    const reprompt = HELP_REPROMPT;
    return buildResponse(handlerInput, speechOutput, reprompt, SKILL_NAME,
BACKGROUND_HELP_IMAGE_URL);
  },
};
const UnhandledHandler = {
  canHandle() {
    return true;
  },
  handle(handlerInput) {
    console.log('--------------------------------------Unhandled')
    const outputSpeech = MISUNDERSTOOD_INSTRUCTIONS_ANSWER;
    return buildResponse(handlerInput, outputSpeech, outputSpeech, SKILL_NAME);
  },
};

const SessionEndedRequestHandler = {
  canHandle(handlerInput) {
    console.log("Inside SessionEndedRequestHandler");
    return handlerInput.requestEnvelope.request.type === 'SessionEndedRequest';
  },
  handle(handlerInput) {
    console.log(`Session ended with reason: ${JSON.stringify(handlerInput.requestEnvelope)}`);
    return handlerInput.responseBuilder.getResponse();
  },
};

// Skill brain

const _initializeApp = handler => {
  // Set the progress to -1 one in the beginning
  handler.questionProgress = -1;
  // Assign 0 points to each animal
  var initialPoints = {};
  Object.keys(resultList).forEach(result => initialPoints[result] = 0);
  handler.resultPoints = initialPoints;
};

const _nextQuestionOrResult = (handlerInput, prependMessage = '') => {
```

```
  const attributesManager = handlerInput.attributesManager;
  const sessionAttributes = attributesManager.getSessionAttributes();
  if (sessionAttributes.questionProgress >= (questions.length - 1)) {

    sessionAttributes.state = states.RESULTMODE;
    return resultIntent(handlerInput, prependMessage)


  } else {
    return nextQuestionIntent(handlerInput, prependMessage);
  }
};

const _applyresultPoints = (handler, calculate) => {
  const currentPoints = handler.resultPoints;
  const pointsToAdd = questions[handler.questionProgress].points;

  handler.resultPoints = Object.keys(currentPoints).reduce((newPoints, result) => {
    newPoints[result] = calculate(currentPoints[result], pointsToAdd[result]);
    return newPoints;
  }, currentPoints);
};

const _randomQuestionIntro = handler => {
  if (handler.questionProgress === 0) {
    // return random initial question intro if it's the first question:
    return _randomOfArray(INITIAL_QUESTION_INTROS);
  } else {
    // Assign all question intros to remainingQuestionIntros on the first execution:
    var remainingQuestionIntros = remainingQuestionIntros || QUESTION_INTROS;
    // randomQuestion will return 0 if the remainingQuestionIntros are empty:
    let randomQuestion =
remainingQuestionIntros.splice(_randomIndexOfArray(remainingQuestionIntros), 1);
    // Remove random Question from rameining question intros and return the removed question.
If the remainingQuestions are empty return the first question:
    return randomQuestion ? randomQuestion : QUESTION_INTROS[0];
  }
};

// Utilities
```

```
let buildResponse = (handlerInput, prompt, reprompt, title = SKILL_NAME, image_url =
BACKGROUND_IMAGE_URL,  displayText = prompt.replace(/(<([^>]+)>)/gi, ""), display_type =
"BodyTemplate7" ) => {
  console.log(title);
        const sessionAttributes = handlerInput.attributesManager.getSessionAttributes();
        sessionAttributes.lastPrompt = prompt;
        sessionAttributes.lastReprompt = reprompt;
        sessionAttributes.lastTitle = title;
        sessionAttributes.lastImage_url = image_url;
        sessionAttributes.lastDisplayText = displayText;
        sessionAttributes.lastDisplay_type = display_type;
  if (reprompt) {
    handlerInput.responseBuilder.reprompt(reprompt);
  } else {
    handlerInput.responseBuilder.withShouldEndSession(true);
  }
   var response ;
  if (Alexa.getSupportedInterfaces(handlerInput.requestEnvelope)['Alexa.Presentation.APL']) {
     response = getDisplay(handlerInput, title,  prompt, image_url, display_type).responseBuilder;
  } else {
     response = handlerInput.responseBuilder.speak(prompt)

  }
  return response.withSimpleCard(title, displayText).getResponse();
}
function supportsDisplay(handlerInput) {
  var hasDisplay =
    handlerInput.requestEnvelope.context &&
    handlerInput.requestEnvelope.context.System &&
    handlerInput.requestEnvelope.context.System.device &&
    handlerInput.requestEnvelope.context.System.device.supportedInterfaces &&
    handlerInput.requestEnvelope.context.System.device.supportedInterfaces.Display
  return hasDisplay;
}


function getDisplay(handlerInput, title, displayText, image_url, display_type){
        if (!image_url.includes('https://')) {
                image_url=Util.getS3PreSignedUrl("Media/"+image_url);
        }


        console.log("the display type is => " + display_type);
    console.log(image_url);
```

```
var VISUAL_TOKEN = 'showthis';
        // Create Render Directive

        handlerInput.responseBuilder.addDirective({
           type: 'Alexa.Presentation.APL.RenderDocument',
           datasources: {
                 "headlineTemplateData": {
                    "type": "object",
                    "objectId": "headlineSample",
                    "properties": {
                       "backgroundImage": {
                          "contentDescription": null,
                          "smallSourceUrl": null,
                          "largeSourceUrl": null,
                          "sources": [
                             {
                                "url": image_url,
                                "size": "large"
                             }
                          ]
                       },
                       "textContent": {
                          "primaryText": {
                             "type": "PlainText",
                             "text": displayText.replace(/(<([^>]+)>)/gi, "")
                          }
                       },

                       "hintText": HINT_TEXT,
                       "welcomeSpeechSSML": `<speak>${displayText}</speak>`
                    },
                    "transformers": [
                       {
                          "inputPath": "welcomeSpeechSSML",
                          "transformer": "ssmlToSpeech",
                          "outputName": "welcomeSpeech"
                       }
                    ]
                 }
              },
           token: VISUAL_TOKEN,
           document: aplMainTemplate
        });
```

```
        return handlerInput;
}

// Init

  const skillBuilder = Alexa.SkillBuilders.custom();
  exports.handler = skillBuilder
    .addRequestHandlers(
      SessionEndedRequestHandler, HelpIntentHandler, ExitHandler, newSessionHandler,
quizModeHandler, resultModeHandler, RepeatIntentHandler,  UnhandledHandler
    )
    //.addErrorHandlers(ErrorHandler)
    .lambda();
```