---------- Forwarded message ---------  
From: **Munish Kanchan** <munishkanchan40@gmail.com>  
Date: Fri, 2 Oct, 2020, 9:43 pm  
Subject: index.json for Ashish  
To: ASHISH KANCHAN <ashishkanchanriachitput@gmail.com>

/\* \*  
 \* 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 persistenceAdapter = require('ask-sdk-s3-persistence-adapter');  
  
const launchDocument = require('./documents/launchDocument.json');  
  
const util = require('./util');  
const birthdayDocument = require('./documents/birthdayDocument.json');  
  
  
  
const LaunchRequestHandler = {  
    canHandle(handlerInput) {  
        return Alexa.getRequestType(handlerInput.requestEnvelope) === 'LaunchRequest';  
    },  
    handle(handlerInput) {  
  
  
        const speakOutput = 'Hello! Welcome to Caketime with Ashish. What is your birthday?';  
        const repromptText = 'I was born November sixth, two thousand fourteen. When were you born?';  
  
         
        if (Alexa.getSupportedInterfaces(handlerInput.requestEnvelope)['Alexa.Presentation.APL']) {  
        // Create Render Directive.  
        handlerInput.responseBuilder.addDirective({  
        type: 'Alexa.Presentation.APL.RenderDocument',  
        document: launchDocument,  
         
        datasources: {  
        text: {  
        type: 'object',  
        start: "Welcome",  
        middle: "to",  
        end: "Cake Time!"  
         },  
        assets: {  
        cake: util.getS3PreSignedUrl('Media/alexaCake\_960x960.png'),  
        backgroundURL: getBackgroundURL(handlerInput, "lights")  
         }  
}  
});  
  
  
        }  
         
         
    
        return handlerInput.responseBuilder  
            .speak(speakOutput)  
            .reprompt(repromptText)  
            .getResponse();  
    }  
};  
  
  
  
    function getBackgroundURL(handlerInput, fileNamePrefix) {  
          
          
    const viewportProfile = Alexa.getViewportProfile(handlerInput.requestEnvelope);  
    const backgroundKey = viewportProfile === 'TV-LANDSCAPE-XLARGE' ? "Media/"+fileNamePrefix+"\_1920x1080.png" : "Media/"+fileNamePrefix+"\_1280x800.png";  
    return util.getS3PreSignedUrl(backgroundKey);  
}  
  
  
    const HasBirthdayLaunchRequestHandler = {  
    canHandle(handlerInput) {  
  
        const attributesManager = handlerInput.attributesManager;  
        const sessionAttributes = attributesManager.getSessionAttributes() || {};  
  
        const year = sessionAttributes.hasOwnProperty('year') ? sessionAttributes.year : 0;  
        const month = sessionAttributes.hasOwnProperty('month') ? sessionAttributes.month : 0;  
        const day = sessionAttributes.hasOwnProperty('day') ? sessionAttributes.day : 0;  
         
         
         
        return Alexa.getRequestType(handlerInput.requestEnvelope) === 'LaunchRequest'  
  
            && year  
  
            && month  
  
            && day;  
    },  
        async handle(handlerInput) {  
  
        const serviceClientFactory = handlerInput.serviceClientFactory;  
         
        const deviceId = handlerInput.requestEnvelope.context.System.device.deviceId;  
         
        const attributesManager = handlerInput.attributesManager;  
        const sessionAttributes = attributesManager.getSessionAttributes() || {};  
  
        const year = sessionAttributes.hasOwnProperty('year') ? sessionAttributes.year : 0;  
        const month = sessionAttributes.hasOwnProperty('month') ? sessionAttributes.month : 0;  
        const day = sessionAttributes.hasOwnProperty('day') ? sessionAttributes.day : 0;  
         
        let userTimeZone;  
  
    try {  
    const upsServiceClient = serviceClientFactory.getUpsServiceClient();  
    userTimeZone = await upsServiceClient.getSystemTimeZone(deviceId);  
    } catch (error) {  
  
    if ([error.name](http://error.name/) !== 'ServiceError') {  
        return handlerInput.responseBuilder.speak("There was a problem connecting to the service.").getResponse();  
    }  
    console.log('error', error.message);  
}  
  
        // TODO:: Use the settings API to get current date and then compute how many days until user's birthday  
        // TODO:: Say Happy birthday on the user's birthday  
         
        // getting the current date with the time  
        const currentDateTime = new Date(new Date().toLocaleString("en-US", {timeZone: userTimeZone}));  
         
        // removing the time from the date because it affects our difference calculation  
        const currentDate = new Date(currentDateTime.getFullYear(), currentDateTime.getMonth(), currentDateTime.getDate());  
        const currentYear = currentDate.getFullYear();  
         
        // getting the next birthday  
        let nextBirthday = Date.parse(`${month} ${day}, ${currentYear}`);  
  
        // adjust the nextBirthday by one year if the current date is after their birthday  
        if (currentDate.getTime() > nextBirthday) {  
        nextBirthday = Date.parse(`${month} ${day}, ${currentYear + 1}`);  
}  
         
        const oneDay = 24\*60\*60\*1000;  
  
        // setting the default speakOutput to Happy xth Birthday!  
        // Don't worry about when to use st, th, rd--Alexa will automatically correct the ordinal for you.  
        let speakOutput = `Happy ${currentYear - year}th birthday!`;  
         
         
        if (currentDate.getTime() !== nextBirthday) {  
        const diffDays = Math.round(Math.abs((currentDate.getTime() - nextBirthday)/oneDay));  
         
         
        speakOutput = `Welcome back. It looks like there are ${diffDays} days until your ${currentYear - year}th birthday.`  
  
        }  
  
   
   
         // Add APL directive to response  
        const diffDays = Math.round(Math.abs((currentDate.getTime() - nextBirthday)/oneDay));  
  
        const numberDaysString = diffDays === 1 ? "1 day": diffDays + " days";  
     
    if (Alexa.getSupportedInterfaces(handlerInput.requestEnvelope)['Alexa.Presentation.APL']) {  
        // Create Render Directive  
          
          
        if (currentDate.getTime() !== nextBirthday) {  
    //TODO Move the old directive here.  
    handlerInput.responseBuilder.addDirective({  
    type: 'Alexa.Presentation.APL.RenderDocument',  
    document: launchDocument,  
    datasources: {  
        text: {  
            type: 'object',  
            start: "Your Birthday",  
            middle: "is in",  
            end: numberDaysString  
        },  
        assets: {  
            cake: util.getS3PreSignedUrl('Media/alexaCake\_960x960.png'),  
            backgroundURL: getBackgroundURL(handlerInput, "lights")  
        }  
    }  
});  
      
      
    } else {  
    //TODO Write a birthday specific directive here.  
    // Create Render Directive  
      
      
    handlerInput.responseBuilder.addDirective({  
    type: 'Alexa.Presentation.APL.RenderDocument',  
    document: birthdayDocument,  
    datasources: {  
        text: {  
            type: 'object',  
            start: "Happy Birthday!",  
            middle: "From,",  
            end: "Alexa <3"  
        },  
        assets: {  
            video: "<https://public-pics-muoio.s3.amazonaws.com/video/Amazon_Cake.mp4>",  
            backgroundURL: getBackgroundURL(handlerInput, "confetti")  
        }  
    }  
}).addDirective({  
    type: "Alexa.Presentation.APL.ExecuteCommands",  
    token: "birthdayToken",  
    commands: [{  
        type: "ControlMedia",  
        componentId: "birthdayVideo",  
        command: "play"  
    }]  
});  
      
    }  
          
          
          
         
         
    }  
  
   
  
        return handlerInput.responseBuilder  
            .speak(speakOutput)  
            .getResponse();  
    }  
};  
  
  
const CaptureBirthdayIntentHandler = {  
    canHandle(handlerInput) {  
        return Alexa.getRequestType(handlerInput.requestEnvelope) === 'IntentRequest'  
            && Alexa.getIntentName(handlerInput.requestEnvelope) === 'CaptureBirthdayIntent';  
    },  
    async handle(handlerInput) {  
        const year = handlerInput.requestEnvelope.request.intent.slots.year.value;  
        const month = handlerInput.requestEnvelope.request.intent.slots.month.value;  
        const day = handlerInput.requestEnvelope.request.intent.slots.day.value;  
         
        const attributesManager = handlerInput.attributesManager;  
         
        const birthdayAttributes = {  
        "year" : year,  
        "month" : month,  
        "day" : day  
        };  
         
        attributesManager.setPersistentAttributes(birthdayAttributes);  
        await attributesManager.savePersistentAttributes();  
         
        const speakOutput = `Thanks, I'll remember that you were born ${month} ${day} ${year}.`;  
  
        return handlerInput.responseBuilder  
            .speak(speakOutput)  
            //.reprompt('add a reprompt if you want to keep the session open for the user to respond')  
            .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();  
    }  
};  
  
  
        const LoadBirthdayInterceptor = {  
        async process(handlerInput) {  
        const attributesManager = handlerInput.attributesManager;  
        const sessionAttributes = await attributesManager.getPersistentAttributes() || {};  
  
        const year = sessionAttributes.hasOwnProperty('year') ? sessionAttributes.year : 0;  
        const month = sessionAttributes.hasOwnProperty('month') ? sessionAttributes.month : 0;  
        const day = sessionAttributes.hasOwnProperty('day') ? sessionAttributes.day : 0;  
  
        if (year && month && day) {  
            attributesManager.setSessionAttributes(sessionAttributes);  
        }  
    }  
};  
  
  
  
/\*\*  
 \* This handler acts as the entry point for your skill, routing all request and response  
 \* payloads to the handlers above. Make sure any new handlers or interceptors you've  
 \* defined are included below. The order matters - they're processed top to bottom  
 \* \*/  
exports.handler = Alexa.SkillBuilders.custom()  
  
.withApiClient(new Alexa.DefaultApiClient())  
  
.withPersistenceAdapter(  
new persistenceAdapter.S3PersistenceAdapter({bucketName:process.env.S3\_PERSISTENCE\_BUCKET})  
)  
    .addRequestHandlers(  
        HasBirthdayLaunchRequestHandler,  
        LaunchRequestHandler,  
        CaptureBirthdayIntentHandler,  
        HelpIntentHandler,  
        CancelAndStopIntentHandler,  
        FallbackIntentHandler,  
        SessionEndedRequestHandler,  
        IntentReflectorHandler)  
        .addRequestInterceptors(  
    LoadBirthdayInterceptor  
)  
    .addErrorHandlers(  
        ErrorHandler)  
    .withCustomUserAgent('sample/hello-world/v1.2')  
    .lambda();