```
const Alexa = require('ask-sdk-core');
var https = require('https');
const LaunchRequestHandler = {
  canHandle(handlerInput) {
    return handlerInput.requestEnvelope.request.type === 'LaunchRequest';
  },
async handle(handlerInput) {
    const response = await httpGet();
    console.log(response);
    var blood = round(response[0].sgv / 18,1);

    var blooddec = round((blood % 1) * 10,0) ;
    var bloodunit = blood - (blood % 1);
    var direction = response[0].direction;
    var directionText = ''
    switch(direction) {
      case 'Flat':
        directionText = 'estable'
        break;
      case 'FortyFiveDown':
        directionText = 'bajando lentamente'
        break;
      case 'FortyFiveUp':
        directionText = 'subiendo lentamente'
        break;
     case 'SingleDown':
        directionText = 'bajando'
        break;
      case 'SingleUp':
        directionText = 'subiendo'
        break;
      case 'DoubleUp':
        directionText = 'subiendo rápidamente'
        break;
      case 'DoubleDown':
        directionText = 'bajando rápidamente'
        break;
      default:
        directionText = direction
      }
    console.log("Direction " + direction);
    return handlerInput.responseBuilder
      .speak('Tienes ' + response[0].sgv  + ' y está ' +  directionText    )
      .getResponse();
  }
};
const HelloWorldIntentHandler = {
  canHandle(handlerInput) {
    return handlerInput.requestEnvelope.request.type === 'IntentRequest'
      && handlerInput.requestEnvelope.request.intent.name === 'HelloWorldIntent';
  },
  async handle(handlerInput) {
    const response = await httpGet();
    console.log(response);
    var blood = round(response[0].sgv / 18,1);

    var blooddec = round((blood % 1) * 10,0) ;
    var bloodunit = blood - (blood % 1);
    var direction = response[0].direction;

    console.log(blood);
    return handlerInput.responseBuilder
      .speak( response[0].sgv + '    ' +  direction)
      .getResponse();
  }
};
const CancelAndStopIntentHandler = {
  canHandle(handlerInput) {
    return handlerInput.requestEnvelope.request.type === 'IntentRequest'
      && (handlerInput.requestEnvelope.request.intent.name === 'AMAZON.CancelIntent'
        || handlerInput.requestEnvelope.request.intent.name === 'AMAZON.StopIntent');
  },
  handle(handlerInput) {
    const speechText = 'Hasta luego!';
    return handlerInput.responseBuilder
      .speak(speechText)
      .getResponse();
  }
};
const SessionEndedRequestHandler = {
  canHandle(handlerInput) {
    return handlerInput.requestEnvelope.request.type === 'SessionEndedRequest';
  },
  handle(handlerInput) {
    // Any cleanup logic goes here.
    return handlerInput.responseBuilder.getResponse();
  }
};

// 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 handlerInput.requestEnvelope.request.type === 'IntentRequest';
  },
  handle(handlerInput) {
    const intentName = handlerInput.requestEnvelope.request.intent.name;
    const speechText = `You just triggered ${intentName}`;

    return handlerInput.responseBuilder
      .speak(speechText)
      //.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) {
    console.log(`~~~~ Error handled: ${error.message}`);
    const speechText = `Lo siento, no he podido entender lo que has dicho. Por favor, inténtalo de nuevo.`;

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

function round(value, precision) {
  var multiplier = Math.pow(10, precision || 0);
  return Math.round(value * multiplier) / multiplier;
}
function httpGet() {
    return new Promise(((resolve, reject) => {
      var options = {
      host: 'misitio.herokuapp.com',
        path: '/api/v1/entries/current.json',
        method: 'GET',
    };
  const request = https.request(options, (response) => {
    response.setEncoding('utf8');
    let returnData = '';

    response.on('data', (chunk) => {
      returnData += chunk;
    });

    response.on('end', () => {
      resolve(JSON.parse(returnData));
    });

    response.on('error', (error) => {
      reject(error);
    });
  });
  request.end();
}));
}

// 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()
  .addRequestHandlers(
    LaunchRequestHandler,
    HelloWorldIntentHandler,
    CancelAndStopIntentHandler,
    SessionEndedRequestHandler,
    IntentReflectorHandler) // make sure IntentReflectorHandler is last so it doesn't override your custom intent handlers
  .addErrorHandlers(
    ErrorHandler)
  .lambda();
```