BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
and Communication

BACHELOR'S THESIS

Brno, 2018 Andrej Dobrík

BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS

ÚSTAV TELEKOMUNIKACÍ

DEVELOPMENT OF AN AMAZON ALEXA SKILL FOR AMAZON ECHO

VYTVOŘENÍ AMAZON ALEXA DOVEDNOSTI PRO AMAZON ECHO

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR — **Andrej Dobrík**
AUTOR PRÁCE

SUPERVISOR — **Ing. Jiří Pokorný**
VEDOUCÍ PRÁCE

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Teleinformatika**
Ústav telekomunikací

Student: Andrej Dobrík ***ID:*** 183628
Ročník: 3 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Vytvoření Amazon Alexa dovednosti pro Amazon Echo

POKYNY PRO VYPRACOVÁNÍ:

Cílem bakalářské práce bude analyzovat současné možnosti balíčku dovedností od Amazonu (Alexa Skills Kit) a identifikovat funkce nebo řešení, která nebyla doposud komerčně implementována. Následně student vytvoří dovednosti s chybějícími funkcemi. Bude vybíráno z těchto možných dovedností: vlastní dovednost, dovednost pro chytrou domácnost, informativní dovednost a video dovednost. Student si jednoznačně definuje parametry dovednosti dle vlastní volby. Výstupem bakalářské práce bude jedna nebo více dovedností v závislosti na jejich úrovni složitosti.

DOPORUČENÁ LITERATURA:

[1] DAVIDOFF, Scott, et al. Principles of smart home control. In: International Conference on Ubiquitous Computing. Springer, Berlin, Heidelberg, 2006. p. 19-34.

[2] HARPER, Richard (ed.). Inside the smart home. Springer Science & Business Media, 2006.

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 29.5.2018

Vedoucí práce: Ing. Jiří Pokorný
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRACT

In the past few years there has been a quite significant shift in how customers interact with their smart devices. For humans the most convenient way to communicate with these devices is through voice interaction. Many companies have realized this change in interaction and started to develop the software and hardware to accompany it.
This bachelor thesis will be devoted to such products, more specifically Amazon Echo products, their capabilities and requirements. Following by explanation of Echo communication. Explanation of Amazon Alexa Skills Kit and the creation and implementation of a new custom Amazon Alexa Skill for the Amazon Echo.

KEYWORDS

AWS, Amazon Echo, skill, custom skill, resource linking, Lambda function, Amazon developer console

ABSTRAKT

V posledných rokoch nastala významná zmena v spôsobe interakcie medzi používateľmi a inteligentnými zariadeniami. Pre ludí je najpraktickejší spôsob interakcie s týmito zariadeniami hlasová interakcia. Veľa firiem si túto zmenu v interakcií uvedomilo a začalo vytvárať software a príslušný hardware pre tento spôsob interakcie.
Táto bakalárska práca sa bude venovať týmto produktom, konkrétne produktom Amazon Echo, ich schopnostiam a požiadavkam. Nasledujúc vysvetlením komunikácie Echa. Priblížením Amazon Alexa Skills Kit a vytvorením a implementáciou nového vlastného Amazon Alexa Skillu pre Amazon Echo.

KLÍČOVÁ SLOVA

AWS, Amazon Echo, schopnosť, vlastná schopnosť, prepojenie zdrojov, Lambda funkcia, Amazon vývojová konzola

DOBRÍK, Andrej. *Creation of Amazon Alexa skill for the Amazon Echo*. Brno, 2017, 63 p. Bachelor's Thesis. Brno University of Technology, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Advised by Ing. Jiří Pokorný

ROZŠÍRENÝ ABSTRAKT

V posledných rokoch nastala významná zmena v spôsobe interakcie medzi používateľmi a inteligentnými zariadeniami. Pre ludí je najpraktickejší spôsob interakcie s týmito zariadeniami hlasová interakcia. Veľa firiem si túto zmenu v interakcií uvedomilo a začalo vytvárať software a príslušný hardware pre tento spôsob interakcie. Zmena v interakcií podporila vývoj technológií pre uľahčenie každodennej interakcie so správami, médiami alebo domácimi spotrebičmi. Pre túto interakciu boli vytvorení viacerí hlasový asistenti, z ktorích najpopulárneší sú Amazon Alexa, Google Home alebo Apple Siri.

Táto bakalárska práca sa bude venovať týmto produktom, konkrétne produktom Amazon Echo, ich schopnostiam a požiadavkam. Nasledujúc vysvetlením komunikácie Echa. Priblížením Amazon Alexa Skills Kit a vytvorením a implementáciou nového vlastného Amazon Alexa Skillu pre Amazon Echo. Pre túto prácu boli vytvorené dve Alexa schopnosti. V prvej teoretickej časti bola vytvorená schopnosť ktorá informovala používateľa o štatistikách jeho Youtubeového kanálu. V druhej časti bola vytvorená Alexa schopnosť pre windsurferov, ktorá na základe predpovede počasia určí vhodné vybavenie.

Cieľom tejto práce bolo preskúmať možnosti Amazon Alexa Skills balíčku, spraviť prieskum marketu Alexa skillov a následne vytvoriť skill, ktorý nebol doposiaľ implementovaný.

V teoretickej časti bakalárskej práce, sú preberané funkcie cloudového výpočtového servisu, ktoré ponúka Amazon Web Servis. Ďalej je priblížený Amazon Alexa Voice Servis ako hlasový assistent, ktorý bol predstavený spolu s Amazon Echo v novembri 2014. Amazon Voice Servis predstavil formu interaktívnej komunikácie a umožnil vývojárom vytvárať nespočetné množstvo Alexa schopností, ktoré môžu byť následne prepojené s rôznymi zariadeniami a produktmi.

Vďaka AVS (Amazon Voice Service), Amazon uľahčil vytváranie konverzačného rozhrania pre výrobcov zariadení. Táto skutočnosť uľahčuje vývojárom pridať Alexu a inteligentné hlasové ovládanie do nových produktov, čím prinášajú pohodlnú hlasovú interakciu používateľom týchto zariadení. Následne je v teoretickej časti popis samotného zariadenia Amazon Echo a jeho komunikácia s Amazon Web Servisom. V poslednej časti sú popísané vlastnenosti individuálnych skillov.

Amazon Alexa funguje na princípe podnetu na ktorý je vyvolaná odpoveď. Na rozdiel napríklad od Google Home je tento podnet nevyhnutný. Invokačné frázy, ktoré spúšťajú jednotlivé funkcie schopností, sú definované pri vytváraní skillu v Amazon Developer Portáli. Tento skill je určený pre windsurferov, ktorý s týmto športom začínajú a nemajú skúsenosti s výberom vhodnúho náčinia na základe predpovede počasia.

Táto schopnosť vyžadovala viac vrstvový interakčný model, pretože sa výmena informácií prebiehala vo viacerých komunikačných sekvenciách. Interakčný model tejto schopnosti prvotne začína vyzvaním používatela na zadanie konkrétneho miesta pre ktoré chce poznať predpoveď počasia, následne mu je táto predpoveď povedaná a používateľ dostane možnosť pre odporučenie vybavenia. Na ktoré môže odpovedať áno alebo nie. Ak odpovie áno tak je zavolaná funkcia ktorá na základe predpovede odporučí náradie. Ak odpovedá nie tak sa tento priebeh ukončí.

Vytváranie tohto skillu prebiehalo v dvoch častiach a to v Amazon Developer Portáli a v Lambda funkcií. V Amazon Developer Portály boli definované intenty inak povedané zámery, ktoré slúžia ako funcie tohoto skillu. K týmto zámerom boli pridané sample utterances alebo inak invokačné frázy, ktoré spúšťajú jednotlivé zámery.

Tieto frázy sú definované na predpoklade najpravdepodobnejších fráz vyslovených používatelom, doplenie chýbajúcich fráz je možné kedykoľvek. Existencia predefinovaných zámerov ako napríklad `AMAZON.YesIntent` uľahčuje proces vytvárania interakčného modelu, z toho dôvodu že invokačné frázy, ktoré spúšťajú tieto zámery, sú už implementované. V Lambda funkcií sa nachádza kód pre definované zámery. Tento kód je prepojený s Amazon Developer Portálom unikátnym Amazon Resource číslom, čo umožňuje real time testovanie tohoto kódu.

Testovanie tohoto skillu prebiehalo v dvoch prostrediach a to Amazon Service simulátore a v Lambda funkcií. V posledných mesiacoch Amazon developer portál dostal update, ktorý pridal možnosť testovania hlasom, čo bolo doposiaľ možné len s prítomnosťou fyzického echa alebo pomocou emulátoru Alexy. Testovanie prebieha zadaním invokačnej frázy, či už v hlasovej alebo v textovej forme. Táto invokačná fráza je premenená na JSON input ktorý sa referuje na Lambda funkciu a následne funkcia pošle odpoveď. Táto forma testovania bola používaná hlavne na konci vytvárania skillu. Lambda funkcia ponúka možnosť logov, čo umožnovalo v kóde izolovať určitú časť a sledovať jej priebeh. Spúšťanie tejto funkcie je možné pomocou test eventov, ktoré sú definované ako JSON input z Amazon Developer portálu. Tento JSON input bol kopírovaný z Amazon Developer Portálu a následne vložený do Lambda Test Eventu. Preto je možné spúšťať konkrétne funcie kódu. V tejto práci bol spravený prieskum trhu a následne boli vytvorené dva custom skilly. API endpointy tejto schopnosti boli úspešne vyvolané a dáta z nich boli spracované. Na základe ich hodnôt boli vytvorené odpovede. Tento skill bol testovaný rôznymi metódami na určenie správnej funkcionality. Tieto testy prebiehali v Amazon Developer konzole a v Lambda Test Eventoch. Alexa emulátor, ktorý bol obsiahnutý v Amazon Developer konzole nahradzoval potrebu prítonosti fyzického Echa pri testovaní. Chovanie tejto schopnosti odpovedalo jej kódu, s ojedinelými zlyhaniami rozpoznania reči.

DECLARATION

I declare that I have written the Bachelor's Thesis titled "Creation of Amazon Alexa skill for the Amazon Echo" independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the thesis and listed in the comprehensive bibliography at the end of the thesis.

As the author I furthermore declare that, with respect to the creation of this Bachelor's Thesis, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

Brno ....

author's signature

ACKNOWLEDGEMENT

Rád bych poděkoval vedoucímu bakalárské práce panu Ing. Jiřímu Pokornému za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

author's signature

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

INTRODUCTION

In the past few years there has been a quite significant shift in how customers interact with their smart devices. Ever since smart devices were introduced to the market, there was a strong emphasis on the convenience of the device being used. For humans the most convenient way to communicate is through voice interaction. Many companies have realized this change and started to develop software and hardware to accompany it.

This thesis will be devoted to such products, more specifically Amazon Echo products, their capabilities and requirements. Following by explanation of Echo communication. Explanation of Amazon Alexa Skills Kit and the creation and implementation of a new Amazon Alexa Skill for the Amazon Echo.

The first section of the thesis will focus on the evolution of the „smart home“ concept, the implementation of smart devices into the household, voice control and the shift to the voice interaction between the user and the device and how did the user experience changed over the years.

The second section will describe the Amazon Alexa Voice Service and the Amazon Echo itself, its structure, the communication between the Echo device and the Amazon Cloud Service. Following by the describtion of the Alexa Skills, Skill types, their abilities and their interaction schema.

The last section will be focused on the creation process of an Amazon Alexa Skill development and implementation. Following with the elements required for the skill to function and the subsequent test scenarios with the help of the Alexa Service Simulator, Lambda function and the Alexa Skill emulator.

1 A SMART HOME

The term „smart home“ has been around for quite a while. Yet, the definition had to go through many changes ever since it was first used in the early 1960s. For the most part, it had to do with the tremendous advancements in technology, but also with the misconception of how a „smart home“ family would function with these implementations [1].

A „smart home“ is not defined by the way it is build, nor how efficient it is when it comes to the power usage, however these traits may be included in the overall description. What is however perceived as „smart“ are the devices and technologies implemented in it. This partially automated home environment should serve as a time and energy saver for the consumer by increasing the comfort, security and the energy consumption efficiency of the household.

Fig. 1.1: Smart home features

1.1 Device implementation

Generally, houses have independent control panels that have to be installed separately and the goal was to centralize the controls of various devices found in the house such as lights, heating, audio setup or other appliances into a single operation unit, that would allow the consumer to program and control the whole environment from one place. Nowadays, many of these appliances can be controlled by user´s „smartphone“, which eliminates the need to actually go to the control panel. This brings us to the most crucial part, the determining factor of whether these implemented products will succeed, which is convenience.

1.2 User experience

When it comes to using a product, an app or a device, the more intuitive it is the more natural the interaction feels. In last years user experience seems to be a dominant discipline, since it stresses innovation of products, satisfaction of the user and overall usefulness. User experience could be divided into two sections: user interaction design and user interface design, which are both related.

User interface design is design of interface or system, which is in direct contact with the user. It could be defined as the designing process of how a user will be able to interact with such a system. It aims to further enhance the usability, visual appealability and technological qualities of an interface or system [2].

User interaction design, on the other hand, focuses on expanding the possibilities to create and further encourage the behavior between the system and user exchange. Furthermore, the interaction design defines the behaviors and structure of products and services, and user interactions with these products and services [3]. A good interaction design should be able to effectively communicate the system´s interactivity and functionality, define behaviors to user interaction, define simple and complex work–flows and inform the user about changes and avoid and prevent user errors.

For us humans, voice is the most natural form of interaction, since it is the way how we communicate. For user-system interaction however, this proposes quite a delicate problem, for speech is rather complex way of expressing, for system that is [4].

1.3 Speech recognition

Speech recognition could be traced back to the early 1950s, when „three Bell Labs“ developed a single–speaker recognition system [5], that worked by locating the *formants* [6], which could be defined as the spectral peaks of the sound spectrum, and had a very limited vocabulary.

Nowadays, modern systems use LSTM(Long short–term memory) that allows the system to learn „Deep learning“ tasks that require memories of events tracing back even thousands of discrete time steps ago [7]. In 2015 Google Voice reported significant performance jump by using CTC(connectionist temporal classification)–trained LSTM, which is available to all smartphone users. In the April of 2017 Google Voice was able to recognize multiple users which allowed a personalized response for everyone using it. This ability was later adopted by Amazon Alexa as well. Alexa users are asked to read 10 phrases aloud so the Alexa can set up a personalized profile [8].

2 AMAZON

Amazon.com, Inc. is an electronic commerce and cloud computing company based in Seattle Washington. Operating in 11 online market places (websites) with 109 centers worldwide, is currently placed as one of the biggest companies in the world. Founded on July 5, 1994 initially under the name Cadabra, Inc. later renamed to Amazon.com, Inc.. This online retailer store started with selling books, later expanded its products from the most basic stuff found in the household through various electronics to furniture.

Amazon is also the largest provider of cloud computing services such as IaaS (Infrastructure as a Service) and PaaS (Platform as a Service). IaaS provides the delivery of hardware in a form of servers, storage and network, and software associated with the hardware(operating systems virtualization technology, file system). The IaaS provider only keeps the data center–operational, this leaves the users to manage and deploy the software and services themselves. An example of an IaaS offering is AWS (Amazon Web Service) EC2 (Elastic Compute Cloud) [9]. The 2.1 illustrates how a virtual machine is build, uploaded, configured and eventually deployed into the IaaS environment.

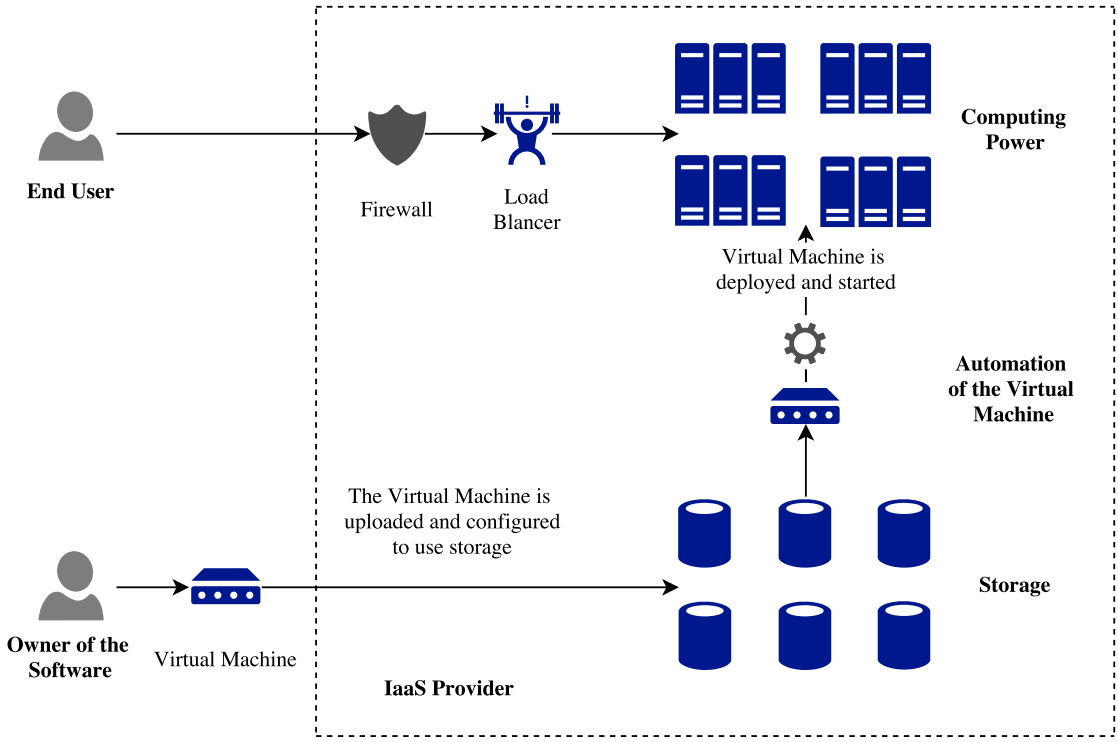

Fig. 2.1: Infrastructure as a Service

PaaS is a platform, delivered over the Web to developers similar to IaaS. However, it also provides a set of programming functions or databases as a foundation for developers to build their applications. An example of PaaS service is AWS Elastic Beanstalk. It allow the developer to deploy, scale web applications and services deployed with Java, .NET, PHP, Node.js, Python, Go and Docker on servers such as Apache, Nginx or Passenger [10].

Cloud computing could be defined as a use of shared computing resources, as an alternative to using local servers to handle applications. Cloud computing groups large numbers of servers together, creating a seamless computing platform. A diagram of this topology is shown in Fig. 2.2.

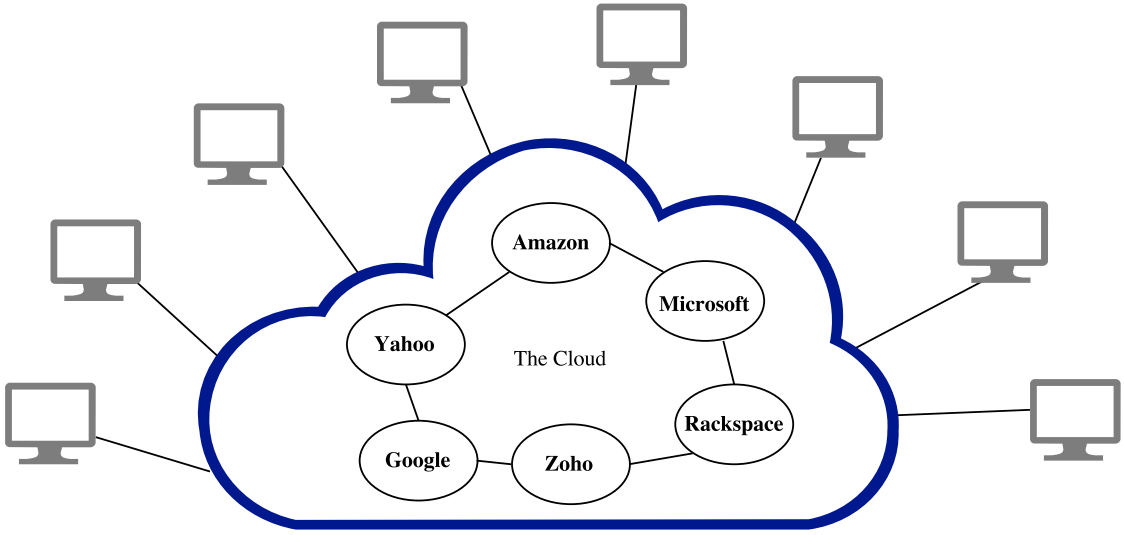

Fig. 2.2: Logical diagram of cloud computing

A typical cloud computing provider delivers these applications online, accessing them from a web service or software, while the actual are stored on servers. This type of service provides convenient, on–demand network access to resource pooling that can be rapidly provisioned and released with minimal management effort or interaction of the provider. Furthermore, this provides large benefits for companies and individuals. Such as the decrease in expenses, for the service is paid incrementally, no need for users and companies to worry about keeping the system up to date, employees can access information wherever they are or the elasticity of the infrastructure to allocate or de–allocate scalable resources to meet the desired requirements of users and companies.

2.1 Amazon Alexa Voice Service

The Amazon Alexa is the brain behind the Echo and million other Alexa-enabled devices, understanding and replying to questions and requests. The reason behind the name *Alexa* was the emphasis on the *X*, for it was easier and more precise to recognize a voice request from the user, by the device. Amazon Alexa Voice Service (AVS) was first introduced alongside the Amazon Echo in the November of 2014. It introduced new voice–forward experience and enabled developers to build countless new skills that were able to connect to other services and products. Through the Alexa Voice Service, Amazon has simplified the creation of conversational interfaces for device makers. This allows developers to add Alexa and intelligent voice control to new products, ranging from mobile devices, smart home appliances to cars, bringing convenient voice experience to customers. The Alexa Voice Service is stored in the cloud, more specifically in Amazon Web Service. It is always getting smarter through constant API updates, new features and new Alexa Skills. Amazon Voice Service offers comprehensive set of development tools and resources, from technical documentation and development kits to prototyping projects and APIs. With the introduction of the Amazon Voice Service SDK, which is fully featured and extensible free to use development toolkit for commercial applications, Amazon made it easier to create and deploy Alexa–enabled products.

2.2 Amazon Echo Devices

Amazon has its own section of voice-controlled products powered by the intelligent voice assistant called Alexa. The first generation of Amazon Echo was announced in 2014 and first became available in 2015, at first only purchasable by Amazon Prime members. It later hit the European market in the second half of 2016 [11].

If a tear down of this 23.5 centimeters tall cylinder would be performed, first to be found would be serial number under the foot of the cylinder (E.g. SK70SDI). Continuing upwards, first parts of the hardware can be found, the power and speaker driverboard, with Step Down Regulator with Integrated Switch, Ultra Low Power Stereo Audio Codec and 15W Filter–Free Class D Stereo Amplifier, all manufactured by Texas Instruments. By removing the outer metal shell of the Echo a 2.0" tweeter and a 2.5" woofer are exposed. Tweeters produce high audio frequencies while the woofers on the other hand produce low audio frequencies. On the side of the exposed Echo a motherboard can be found with the main components of the Echo. A Digital Media Processor and Integrated Power Management IC, both manufactured by Texas Instruments. Following by a slot of 256 MB of LPDDR1 RAM made by SAMSUNG. 4 GB iNAND Ultra Flash Memory made by SanDisk.

Last but not least a Wi–Fi and Bluetooth modules made by Qualcomm Atheros can be found on the motherboard. At the top of the Echo device two buttons can be found, a muting button and the action button. All 7 microphones that listen for the user input are also located in the top part of the cylinder, from which 6 are located on the edges and one is located in the center of the top part. A LED light ring is located around the top that changes the intensity, of the blue light it emits, when Alexa acknowledges a voice command. The top wheel–like part of the Echo is placed on gears, if spinned by the user it changes the volume of the device based on the value which the encoder recieves by the movement of the wheel. By removing the head of the device and stripping of the plastic foam which it is covered in, another driverboard with components is found. Four Programmable 9–Output LED Drivers, four 92 dB SNR Low–Power Stereo ADC located around the middle of the driverboard and a Dual Positive–Edge–Triggered D–Type Flip–Flops, all manufactured by Texas Instruments. By far the most spacious of the Echo internals is a saxophone shaped reflex port, also known as a bass reflex system which boosts the bass while minimizing the distortion of the sound. It allow the Echo to be heard from further away. A remote controller is also included with the Echo. It has basic functions such as mute, play, pause, rewind, volume up and volume down [12]. A more detailed list of the Echo components is listed in Tab. 2.1.

This device would continuously listen to all speech, waiting for the triggering keyword to be spoken. This function could have been disabled by a mute button at the top of the cylinder. In order for the Echo to be functional, a wireless internet connection is needed. A dual-band Wi-Fi 802.11/a/b/g/n was used for this and a Bluetooth with A2DP (Advanced Audio Distribution Profile) for audio to be streamed and AVRCP (Audio/Video Remote Control Profile) for voice control of mobile devices that were connected to the Echo. It met with positive claims from users and developers, for they have realized the vast possibilities of implementing this device.

In March of 2016 Amazon introduced the Echo Dot, the first descendant of the Amazon Echo First Generation. For the most part the functionality did not change, however it came in a smaller format and was intended to be connected to external speakers. In the October of 2016 the second generation of the Echo Dot was introduced, with improved voice recognition and the ESP(Echo Spacial Perception) that enabled several Echos and Dots to be connected together.

Amazon later expanded on their Alexa-enabled devices with Amazon Tap which was a portable, battery-charged version of Echo. Echo Look with a built-in camera, that could in addition to classic Alexa function, take photos and record video. Echo Show, a 17 centimeters LCD screen media playing device. Echo Plus, that improved upon the first generation Echo and served as a smart home hub for most home-

component	model	manufacturer
bottom part driverboard		
3A Step-Down Regulator with Integrated Switcher	TPS53312	Texas Instruments
Ultra Low Power Stereo Audio Codec	TLV320DAC3203	Texas Instruments
15W Filter-Free Class D Stereo Amplifier	TPA3110D2	Texas Instruments
main motherboard		
Digital Media Processor	DM3725CUS100	Texas Instruments
4 GB iNAND Ultra Flash Memory	SDIN7DP2-4G	SanDisk
Wi-Fi and Bluetooth Module	QCA6234X-AM2D	Qualcomm Atheros
Integrated Power Management IC	TPS65910A1	Texas Instruments
top part driverboard		
Programmable 9-Output LED Driver (x4)	LP55231	Texas Instruments
92dB SNR Low-Power Stereo ADC (x4)	TLV320ADC3101	Texas Instruments
Dual Positive-Edge-Triggered D-Type Flip-Flops	SN74LVC74A	Texas Instruments
V6 Microphone (x7)	S1053 0090	SiSonic

Tab. 2.1: Amazon Echo Components

connected smart devices, it also included considerably better microphones. Echo Connect that allowed users to connect their phone line to the Echo and make calls through this device [13].

2.3 Echo Communication

A cloud service is hosted by Amazon called the Amazon Alexa Service that stores the data, metadata and references required to run a skill. As a developer you register your skill through Amazon Developer Portal by defining the metadata about the created skill, such as name of the skill, logo and keywords required to activate the request of this skill or the kind of specific queries the developer wants it to respond to. The metadata is deployed and stored to the Amazon Alexa Service.

However, the actual custom code created by the developer, that is used to implement the skill, is not deployed on the Amazon Alexa Service itself, but it is deployed separately. There is however, a reference that includes the destination of where the implementation lives, as a section of the metadata that is deployed onto implementation on the Amazon Alexa Service. This enables the developer to update the implementation whenever he or she desires to do so, without the need to re-sync it to the hardware every time. This enables the Echo device to serve as a live test-drive to test the implemented changes with.

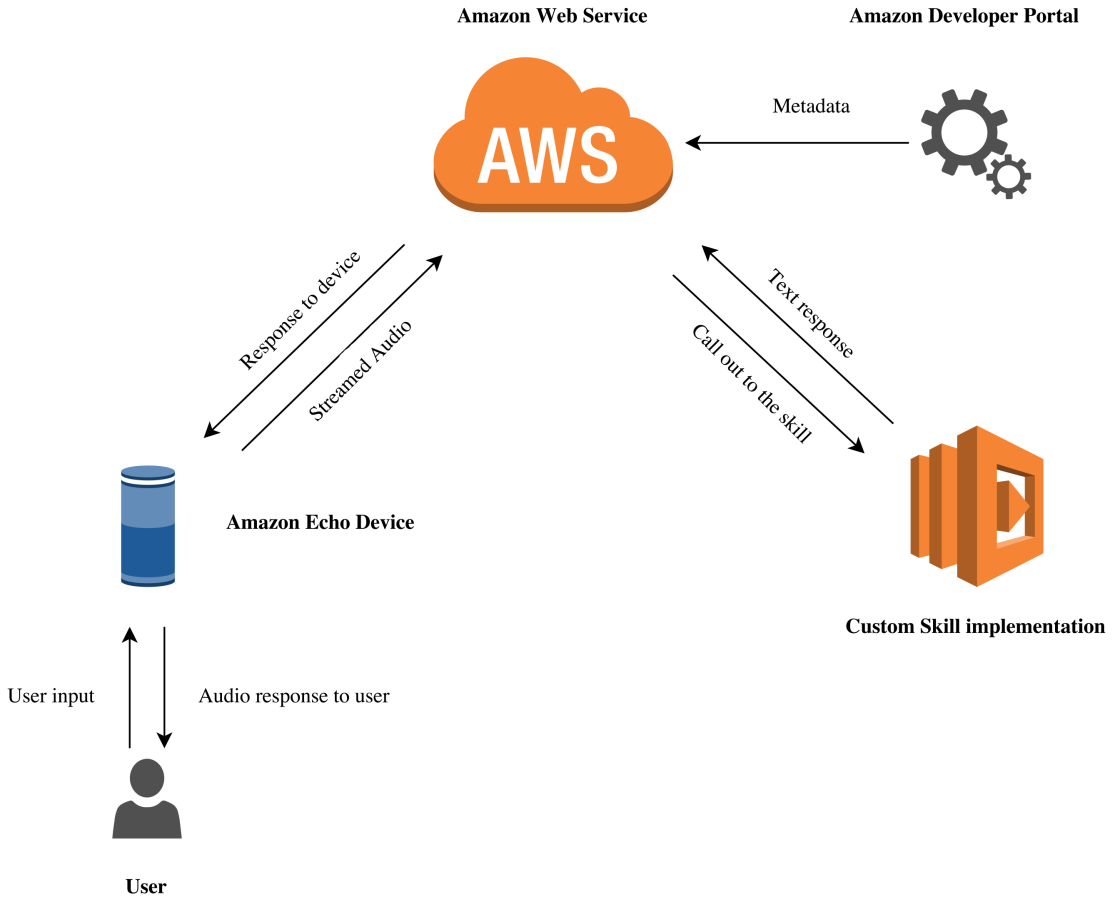

Fig. 2.3: Communication Process

The simplest way to code the implementation for a skill is within the Amazon Web Services Lambda function. Lambda is an „Amazon Web Services“ service that enables the developer to deploy functions without need to create a run-time environment for the code to be executed in. Simply put, a developer writes a function and Amazon will scale up a run-time environment, executes the function, then scales the run-time environment down. This is an excellent use case for an ephemeral process such as making a one-off web request in response to a voice

command by the user [14].

The process starts with the user speaking to the device, the audio is streamed to the Amazon Alexa Service. Based on the metadata, that was specified by developer´s skill implementation, a skill is looked up, then a reference to the skill implementation is found and it is called out to it. The called out skill will respond with a text response to the Amazon Alexa Service. From there a response is sent back to the Echo device. This is illustrated in Fig. 2.3.

2.4 Amazon Alexa Skills

Amazon Alexa provides users with a set of built-in abilities, also referred to as skills. These skill vary from playing music from different providers such as Spotify or Apple Music, answering questions or providing weather forecast.

The Alexa Skills Kit allows developers to teach Alexa-enabled devices new skills. These skills can be then accessed by asking questions or making requests. All Alexa Skills could be sorted into five categories, briefly mentioned in Tab. 2.2, these individual categories will be expanded on later in Skill Types section.

Custom Skills	A set of skills that can order the customer a pizza or call an Uber
Smart Home Skills	By using SMART HOME skill API, the user can control smart devices in the house
Flash Briefing Skills	The user will be provided with brief list of news, based on user´s predefined sources, E.g. BBC, NPR
Video Skills	By using VIDEO skill API, user can change stations or play requested movie on their smart TV
List Skills	Users can adjust their lists, change their contents and add to them

Tab. 2.2: Brief list of Skill types

Depending on which type of the skill developer choses to opt for, the functionality of the skill will be determined by its type. As mentioned before in Echo Communication 2.3, the developer creates a cloud-based service which handles the skill types of the request. Which are then routed to appropriate services.

When developing a skill, the first step is to decide on what type of skill is going to be developed. As mentioned before, these skill types determine the attributes of skills and require certain steps for them to work.

2.4.1 Custom Skills

The Interaction Model of a Custom Skill consists of three parts: the Intent Schema, Slot Types definitions and Sample Utterances. The Interaction Model can be found on the Amazon Developer web page, where the skill is registered, programmed and deployed.

In the intent schema, the developer defines the intents. The intents are programmed in *JSON* (JavaScript Object Notation) data structure [15], which is a lightweight data–interchange format, based on a sub–net of the JavaScript programming language. It's text format is language independent which means that the code would work across various programming languages. However, it uses conventions similar to the programming languages such as *C*, *C#* or *Java*.

Intents represent the actions a skill can perform. The intent schema is a statically defined interface that specifies how many different actions a skill can perform, what their names are and what arguments they take. The Alexa Skills Set provides developers with a library that includes built-in intents. These intents can be used to add functionality to the skill without providing any Sample Utterances. A Sample Utterance is basically the words users say to invoke the request. Every-single built-in intent has an intent signature defined to it. An intent signature starts with the action and sets the type of the action´s properties to the entity types it acts on or uses with the following syntax.

2.4.2 Smart Home Skill

Smart Home Skills provide a simple way for developers and users to enable Alexa Voice interaction to control and monitor the status of cloud–connected devices. When it comes to the Smart Home Skill interaction model and development of the skill, it is simplified for its complexity of the voice interactions are handled by Amazon. Sample Utterances are interpreted by Alexa and messages are sent to the skill that communicates these requests. In addition, the Smart Home Skill API enables developers and users to display the up–to–date status of the devices that are configured in the Alexa app.

The way the Smart Home Skill API works is that the customer enables a Smart Home Skill in the Alexa app, links it to their corresponding account with a device cloud and assigns the discovered devices associated with the account. When an utterance is spoken by the user such as, „Alexa, dim the livingroom lights to 50%“, or the user adjusts the lights in the app itself, Alexa recognizes the intent of the customer to change the settings on the specified device. Alexa uses this information to create a message called *directive*. The *directive* includes customer's authentication information, the device identifier and the value of the new setting. This message is

then sent by Alexa to the Smart Home Skill that controls the lights. The developer of the Smart Home Skill receives and parses this message in code, hosted in AWS Lambda and passes it to the specified device in customer's device cloud. The developer responds with a message called *event* and indicates whether the request was successful or not successful. There is the option of sending the event synchronously from the Lambda function or asynchronously from the device cloud. Alexa then uses this *event* information to respond to the customer's request. This process is illustrated in Fig. 2.4.

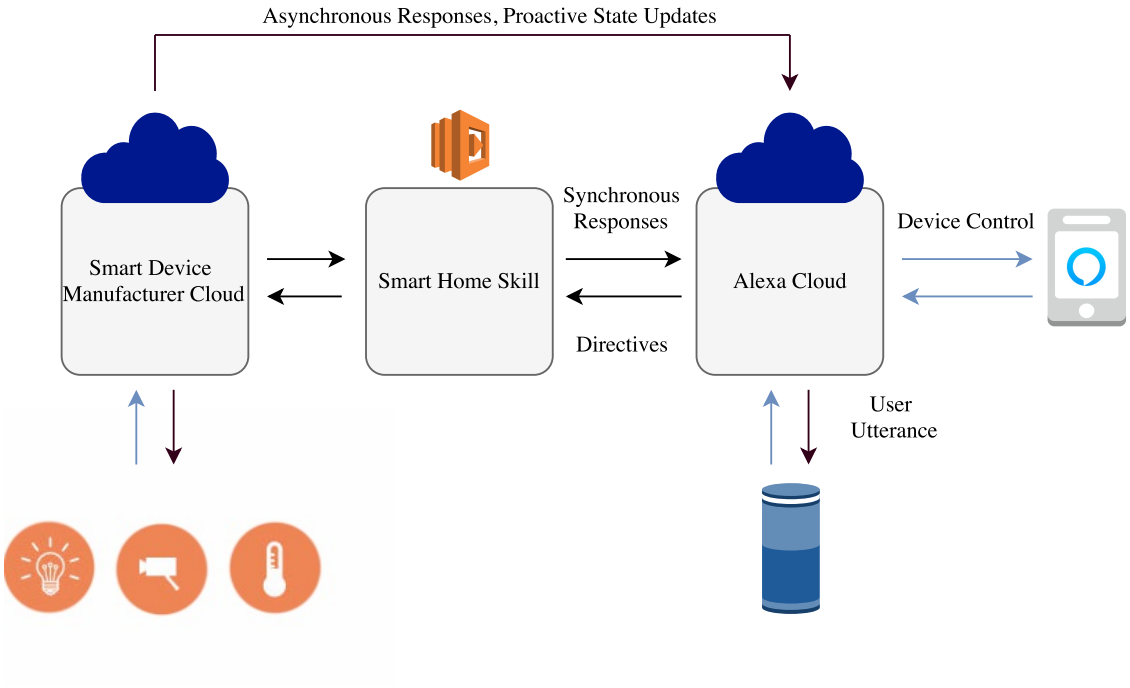

Fig. 2.4: Smart Home Skill Communication Process

In general, Smart Home Skills development can be divided into two categories. Developers working for device manufacturers that produce cloud–connected devices, that want to enable their devices with voice interaction capabilities for the customers to interact with. And independent developers who want to control their own cloud–connected devices through Alexa Skills for their private or general public use. However, when it comes to general public use, developers are required to acquire certification for such use.

2.4.3 Flash Briefing Skill

Flash Briefing provides an overview of news and other content, that the customer enables in skill section of the Alexa app. The customer triggers their Flash Briefing by asking the Alexa–enabled device phrases such as „Alexa, tell me the news“, or

„Alexa, give me my Flash Briefing“. Alexa then reads out the text content or plays audio that is provided by the enabled skill. Flash Briefing can also include a daily joke.

A Flash Briefing Skill provides the content for customer's Flash Briefing, so when the developer is creating a skill, he or she has the opportunity for providing original content to reach customers on daily basis. Furthermore, there is the need to update every so often, for the content to stay relevant. Also, the content should be provided in either text or audio format.

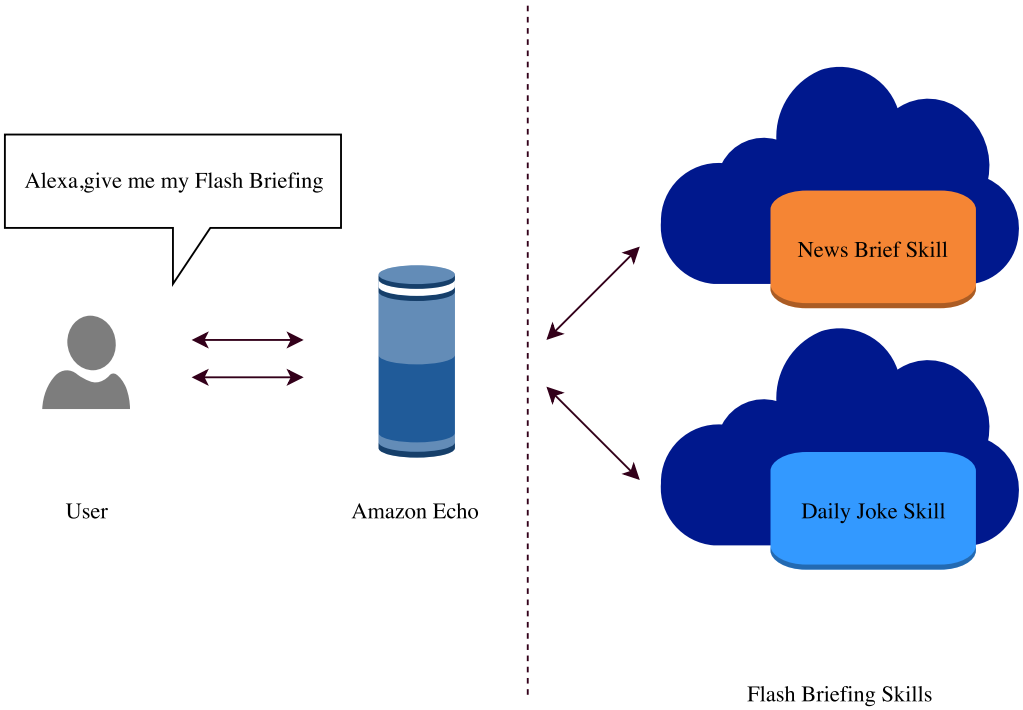

Fig. 2.5: Flash Briefing Skill Communication Process

The Flash Briefing Skill API defines the format of the content and allows developers to create the Flash Briefing Skills. The developer uses the developer portal to define the name, logo and other information for the skill to be displayed in. Moreover, a configuration of the feeds populating the Flash Briefing Skill is required. The developer can only create skills with content he or she has the right to. An understanding of web technologies and HTTPS, RSS or JSON configuration abilities are required. The communication process is illustrated in Fig. 2.5.

2.4.4 Video Skill

A rather significant difference distinguishing Video Skill from other types of skills, in contrast with a custom skill for example, i that the Video Skill API enables

customers to easily find and consume video content without invoking a specific skill. Moreover, Alexas awareness of user's devices and subscription services provides this functionality and allows voice control over them. Provider produces new content, a new television series could serve this example. If the Video Skill API is used to create this skill, the series can be assigned to the Alexa's content catalog. Alexa then recognizes the content assigned to certain provider. Customers can then say the name of the series, E.g. „Alexa, play Game of Thrones", Alexa identifies the provider and helps the customer to enable the skill needed to play such series. If the Video Skill API was not used the customer would be required to remember the providers name and enable specific skills manually. Developing a Video Skill is simplified, for the interaction is defined by Amazon. The communication process of Video Skill is illustrated in Fig. 2.6.

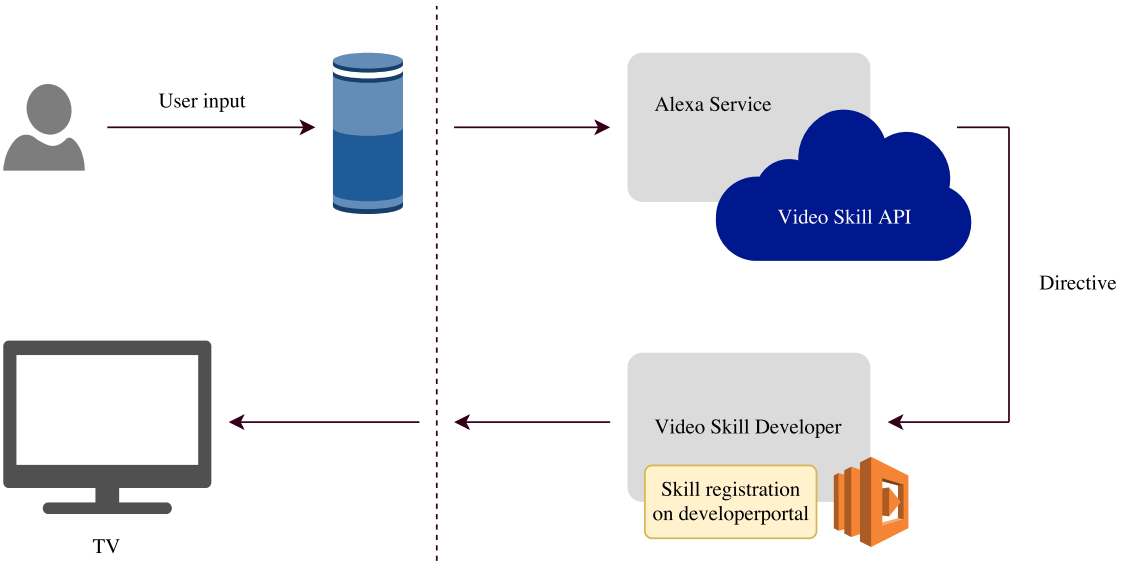

Fig. 2.6: Video Skill Communication Process

2.4.5 List Skill

A List Skill is created with the ASK CLI (Alexa Skills Kit Command Line Interface). This type of skill makes use of list events in the skill service. Moreover, the skill can understand and react to changes and adjustments that occur to lists from top–level utterances on Alexa. A List Skill can include custom component that can include any abilities and features accessible for Custom Skill. The List Skill API defines the intents and user interactions such as: adding ore removing items from list, updating the already existing item, endpoints to receive list events and handlers for processing and responding to events.

3 CREATION OF A CUSTOM YOUTUBE SKILL

For the Amazon Alexa, with its Alexa–enabled devices, has been on market for few years and during its existence amounted over 20000 skills either created by manufacturers or individuals, it is significantly more difficult to create a new and original skill. With this approach, the decision was to create a skill for personal use that could however be implemented for other users. This skill should be able to inform the owner of the Youtube channel about their primary analytics such as current video view count and current subscriber count, to eliminate the need of checking them manually in he Youtube creator studio.

An interaction instance for the „GetVideoViewCount“ and „GetSubscriberCount“ intents is illustrated in Fig. 3.1. The user should also be able to pass a date parameter utterance and receive these analytics based on the time period requested. The requested analytics however, can not be extracted directly from Youtube. Moreover, Google developer account will be required to create the Application Programming Interface (API) that includes these analytics. The JavaScript code that will power the intents of this skill will reside in the AWS Lambda function, surce-linked to the Amazon developer console.

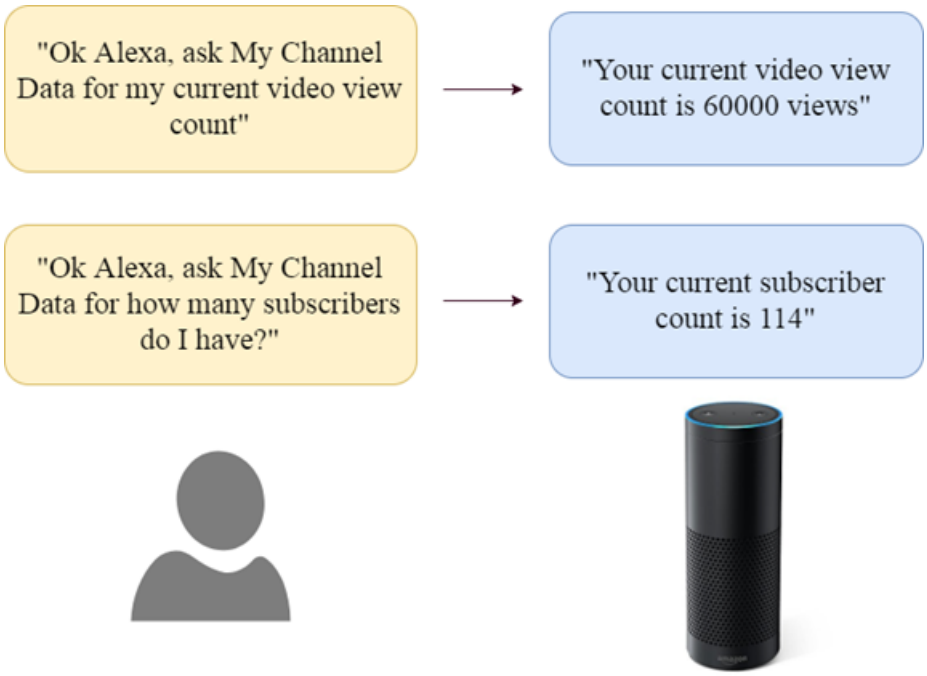

Fig. 3.1: User interaction case

The functionality process of this skill would start with the invocation phrase „Hey, Alexa“, that triggers the start of a new session. This session is started in the background while the Echo device listens to the intent invocation utterances. Based in the utterance spoken, three scenarios could occur. The launch request utterance was recognized and the skill introduces itself to the user. The utterance is not recognized or not valid and the session will end with the Echo audio output. Or, the utterance invocation succeeds and follows with given intent request. An HTTP request is sent out to the API URL endpoint with a string data response that is consequently parsed and transformed from string to JavaScript object. This phase could result in either context.succeed of context.fail. If failed, a response is generated and the session is ended If succeeded, a response is generated and built. This success response is the main objective of this skill. This process is illustrated in Fig. 3.2 diagram.

3.1 Amazon Developer Portal

First step to create the skill was to create an Amazon developer account, for a basic Amazon account does not provide any development capabilities of the Amazon Developer Console. For this skill an Amazon Alexa Skill Kit was used. There is another option for the user to bring voice capabilities to Alexa–enabled devices, called Alexa Voice Service. The skill registration registration includes 6 steps, or rather sections to be filled out to further specify the skill.

Skill Information section defines the skill type, where a custom skill was chosen, language of the skill, generated ID of the skill, name under which the skill will be represented in the Alexa app and the invocation name that is used to trigger the skill.

Next section consists of the Interaction Model where intents, slots and sample utterances are defined, mentioned in Section 2.4.1. The intents were specified in the JSON data structure as an argument `„intents"` which takes a vector of the different intents. For the intent to be defined a key value needed to be paired with the intent, one of which was intent of pulling current video view count of the youtube channel, defined as `"intent" : "GetVideoViewCount"`. This simple intent does not require any slots.

```
{
  "intents": [
    {
      "intent": "GetSubscriberCount"
    },
    {
      "intent": "GetVideoViewCount"
    },
    {
      "slots": [
        {
          "name": "SinceDate",
          "type": "AMAZON.DATE"
        }
      ],
      "intent": "GetVideoViewCountSinceDate"
    }
  ]
}
```

Listing 3.1: Intent Schema

Slot, a statically typed argument passed into the intent, will however be required in another intent that will pass a date attribute for the view count for example. Furthermore, the skill will be able to take in an additional utterance from the user. The slot type used in `"GetVideoViewCountSinceDate"` intent is provided and defined by Amazon itself as a default slot type, moreover there will be no need to map any natural language phrases for this slot, such as „this month“ or „this year“ for they are included in this default slot type. Most default slot types are sufficient enough for the majority of skills. The intent schema used for this skill is shown in Fig. 3.1.

These intents however require mappings from a natural language query to them, in a form of sample utterances. For each intent a number of sample utterances was added. The higher the number of different phrases that is added to the sample utterances field, the smaller the probability of an error. If a slot type is defined in the intent, it has to be included in the corresponding utterance, E.g. `how many video views do I have from {SinceDate}`. A number of sample utterances is shown in Listing 3.2.

```
GetSubscriberCount current subscriber count
GetSubscriberCount subscriber count
GetSubscriberCount how many subscribers do I have
GetSubscriberCount number of subscribers
GetVideoViewCount current video view count
GetVideoViewCount video view count
GetVideoViewCount how many video views do I have
GetVideoViewCount number of video views
GetVideoViewCountSinceDate current video view count {SinceDate}
GetVideoViewCountSinceDate video view count {SinceDate}
GetVideoViewCountSinceDate how many video views do I have {SinceDate}
GetVideoViewCountSinceDate how many video views do I have for {SinceDate}
GetVideoViewCountSinceDate number of video views {SinceDate}
```

Listing 3.2: List of Sample Utterances

In the Configuration field an endpoint needs to be defined to where the code to run the skill is implemented. This was done by the Lambda ARN (Amazon Resource Name). The Lambda ARN is an unique ID for an Amazon resource and it is obtained when a Lambda function is created. This resource linking allowed a real time testing of the Lambda function from the Amazon Developer Console.

3.2 AWS Lambda Function

To create a Lambda function, first an AWS account had to be created. When creating a Lambda function two options are presented. To either write the code from scratch or build it from a blueprint. Blueprint is essentially a template of prewritten code. The decision however, was to write the function code from scratch. Next step was to select a trigger on which the Lambda function would be automatically invoked. An Alexa Skills Kit trigger was selected. Following onto the Configuration tab, the ARN (Amazon Resource Name), Amazon CloudWatch logs and a inline text editor are presented. For this function a Node.js 6.10 runtime was selected. With this configuration, the JavaScript code could be written and executed.

Whenever any function is written on Lambda, it needs to be wrapped inside a function handler with a signature `exports.handler = (event, context) {}`,

which is a default signature for the Lambda function. The custom skill itself, or rather any skill in particular, consists of four separate events in the whole life cycle of the skill, which are:

- New Session
- Launch Request
- Intent Request
- Session End Request

Whenever a skill is executed a new session is created and the event will be fired. When a skill is invoked by only its invocation name without any additional utterances, a launch request is called. For this custom skill, if „my channel data“ was said, that would call the launch request and the Amazon Echo would respond with „Welcome to a custom Alexa skill, this skill is a part of the Amazon Echo semestral thesis“. Moreover if a phrase „current subscriber count“ was added to the initial invocation name, that would trigger the intent request for the particular `GetSubscriberCount` intent.

Furthermore, session ended request is called when the skill has ended. The control flow the code starts with an `if` statement to check whether there is a new session event. Once inside a new session a `switch` statement with `(event.request.type)` parameters can be implemented for launch request, each intent request and session end request.

Helpers are required to generate the actual data structure returned from this Lambda function onto the Amazon Alexa Service to pass to the Echo device, `generateResponse` helper will serve this purpose. The `response:` data structure, which is the `speechletResponse` generated from the `buildSpeechletResponse` helper. The `speechletResponse` is a data structure that specifies the output text for the device to speak, as well as whether or not should end the session or leave the session open for more requests. Additional data can be implemented, such as the reprompt text as what the Amazon Echo says when it does not receive an input for a period of time. For a value to be returned from each request a call out to `context.succeed` and `generateResponse` passed to it from the helper is used. A simple way to demonstrate this is in the launch Request, where the `context.succeed` includes the `generateResponse` to pass into it and `buildSpeechletResponse` with the text output and consequently end the session. This code can be tested on either the Lambda console or through the ARN in the Test section.

For the intent request a `switch` statement with `(event.request.intent.name)` parameters is used to execute the domain logic for each intent within. As an example of this logic will be the domain `GetVideoViewCount`, illustrated in Listing 3.3. The endpoint to which the function will call out is in a URL form of Youtube Data API,

more on which in 3.3 Section. Then make an HTTP request, get the response, parse it, get the video views from Youtube API, following by the `content.succeed` call out and generate the text response with the number of video views. This process is illustrated in Fig. 3.2 diagram.

```
case "IntentRequest":
        // Intent Request
        console.log(`INTENT REQUEST`)

        switch(event.request.intent.name) {
          case "GetSubscriberCount": //intent reference
            var endpoint = "https://www.googleapis.com/youtube/v3/channels?part=snippet,statistics&id=UChJFEa7pTnzWdu9v8GuomcQ&key=AIzaSyAIfOhEQiAB4dwmI0RHI8BJjY1z0neHErc" //endpoint reference in this case call out to the Youtube API
            var body = ""
            https.get(endpoint, (response) => { //HTTP request
              response.on('data', (chunk) => { body += chunk }) //response
              response.on('end', () => {
                var data = JSON.parse(body) //data parse
                var subscriberCount = data.items[0].statistics.subscriberCount //data acquisition
                context.succeed( //in case of success
                  generateResponse(
                    buildSpeechletResponse(`Current subscriber count is ${subscriberCount}`, true),
                    {}
                  )
                )
              })
            })
            break;
```

Listing 3.3: GetVideoViews code logic

The same approach is used in the `GetSubscriberCount`. After the creation of the Lambda function, the Amazon Resource Number is generated. This ARN was copied and pasted into the configuration section in the Amazon developer console. After this step, the Lambda function was linked to the Alexa skill. Every saved change in the code of the Lambda function will be automatically updated to the Alexa skill.

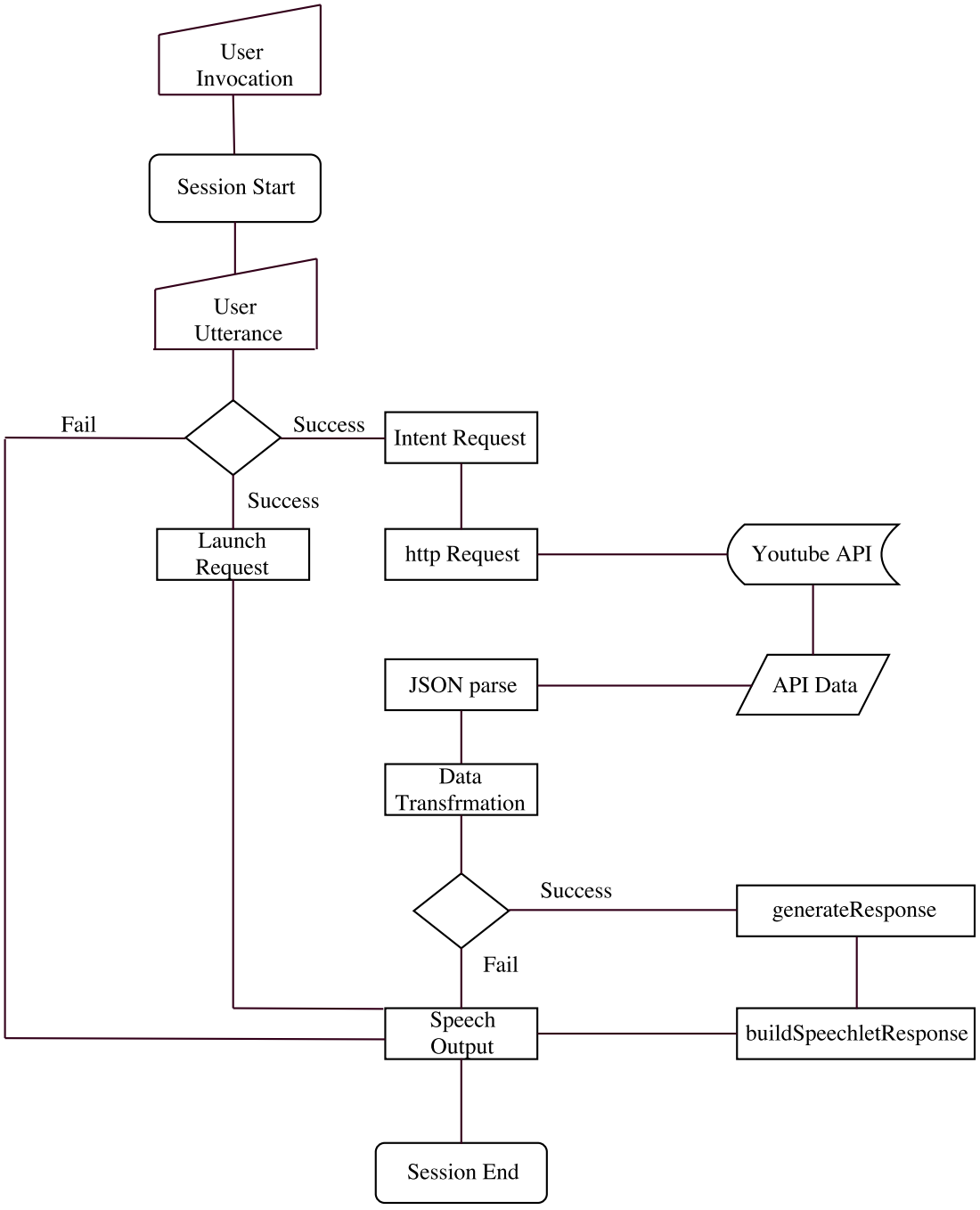

Fig. 3.2: Control Flow Diagram

3.3 Application Programming interface (API)

For this particular skill, various data needed to be extracted from Youtube API service in an URL form. More specifically, the Youtube DATA API and Youtube Analytics and Reporting API. For Youtube is owned by the Google, each of these services required a Google developer account and authorisation.

After creating the google developer account and API needed to be aquired. An API key is a generated string that is unique for every account. This API key is used for the authorisation and is included in the URL of the API.

Youtube API can either be created manually or by an API generator provided by Google. For intents `GetSubscriberCount` and `GetVideoViewCount` a Youtube DATA API was used, this API provides both parameters with additional information.

In the API generator for a channel description API two parameters were required, ID of the channel and a part parameter. ID of the channel is an unique string generated upon registration of the corresponding channel. The part parameter included snippet and statistics, for these two attributes provide sufficient data for the corresponding intents. Executing these settings triggers an immediate API outcome response, whether successful or not.

However, this response does not include the actual API URL. To obtain this URL an APIs explorer had to be loaded, where more parameters were available, such as the maximum number of responses that the API should return or a boolean „mine“ value. This „mine“ value determines whether the channel should instruct the API to only return channels owned by the authenticated user. The authorisation scopes were selected and executed, a sign in into the google account was issued and confirmed.

A request URL was generated: `https://www.googleapis.com/youtube/v3/channels?part=snippet,statistics&id={CHANNEL_ID}&key={YOURA_PI_KEY}` where the `id={CHANNEL_ID}` parameter was replaced with appropriate channel identification number and the `key={YOUR_API_KEY}` parameter by unique API key. This API would serve as an endpoint to which the intent will call out to.

A similar process was followed while constructing the API for the `GetVideoViewContSinceDate` intent. Moreover, the Youtube Analytics and Reporting API was chosen. Process of authentication of this API is done either with the Google OAuth 2.0 or an API key, OAuth 2.0 was chosen for this API for it requires an authorised access.

```
{
 "kind": "youtubeAnalytics#resultTable",
 "columnHeaders": [
  {
   "name": "day",
   "columnType": "DIMENSION",
   "dataType": "STRING"
  },
  {
   "name": "views",
   "columnType": "METRIC",
   "dataType": "INTEGER"
  }
 ],
 "rows": [
  [
   "2013-04-19",
   187.0
  ],
  [
   "2013-04-20",
   53.0
  ],
  [
   "2013-04-21",
   26.0
  ],

//analytics continue to the end date parameter
```

Listing 3.4: Youtube Analytics and Reporting API Response

This API requires more parameters to function, for instance the ID of the channel, start and end date of the searched parameter, metrics that describe what should be accessed, in this case views and the dimensions parameter disclosing the date range of the metrics. These parameters are mandatory for the API to be functional. Following the authentication and execution, an URL is generated with a response illustrated in Listing 3.4.

3.4 Testing of the Skill

As mentioned in Section 2.3 and illustrated in Fig. 2.3, real time testing is available via the linking of Amazon Web Service with the deployed Lambda function. There are four convenient ways of testing of the skill, each with its respective advantages:

- Amazon developer console service simulator
- Lambda function test events
- Directly through interaction with the Amazon Echo
- Alexa Skill testing tool emulator

3.4.1 Amazon Developer Console Testing

The service simulator resides in the Test section of the Amazon developer portal. Moreover, based on the text input in a sample utterance form, the service simulator provides a Service Request and Service Response, either successful or not. As an utterance example will be used current subscriber count intent utterance. For this type of testing, there is no need to provide a full sentence for the service simulator to work. A simple sample utterance will trigger the test. After entering the „current subscriber count“ utterance, Service Request and Service Response were generated, see Listing 3.5 and Listing 3.6.

The Service Request indicates the start of a new session assigned to the users Amazon ID. A request type is chosen, in this case based on the `GetSubscriberCount` intent. For the service response `outputSpeech` is issued with the `plainText` type and the`BuildSpeechletRespond` outputs the phrase „Current subscriber count is 114.“, and ends the session.

```
{
  "session": {
    "new": true,
    "sessionId": "SessionId.e47df305-c6ca-4da7-ae9a-45e2
   fdfd0409",
    "application": {
      "applicationId": "amzn1.ask.skill.9fec1436-4566-49
   dc-9001-c3163e499c70"
    },
    "attributes": {},
    "user": {
      "userId": "amzn1.ask.account.AGMAM6LBXWMFGJ"
    }
  },
  "request": {
    "type": "IntentRequest",
    "requestId": "EdwRequestId.c8d42726-9d1f-404c-ab73-56
   d8be403a82",
    "intent": {
      "name": "GetSubscriberCount",
      "slots": {}
    },
  "context": {
    "AudioPlayer": {
      "playerActivity": "IDLE"
    },
       "device": {
        "supportedInterfaces": {}
      }
    }
  },
  "version": "1.0"
}
```

Listing 3.5: Service Request

```
{
  "version": "1.0",
  "response": {
    "outputSpeech": {
      "text": "Current subscriber count is 114",
      "type": "PlainText"
    },
    "speechletResponse": {
      "outputSpeech": {
        "text": "Current subscriber count is 114"
      },
      "shouldEndSession": true
    }
  },
  "sessionAttributes": {}
}
```

Listing 3.6: Service Response

During the test of the `GetVideoViewCuntSinceDate` the `SinceDate` parameter was replaced by „this month“ utterance. The intent request of the Service Request is illustrated in Listing 3.7. It is noticeable how intent request calls out to the slot type parameter of `SinceDate` with the value translated from „this month“ utterance to `"value" : "2017/12"`. This test was conducted on various `SinceDate` utterances such as today, this year, last year, last month, from January, from January of 2016. Each of these respected utterances was translated to their descriptive values. The Service Response however, responded with an error. This had to do with the non-functioning Youtube Analytics and Reporting API, mentioned in Section 3.3.

```
"request": {
  "type": "IntentRequest",
  "requestId": "EdwRequestId.e0a86429-9621-472d-9381-22
297661caf6",
  "intent": {
    "name": "GetVideoViewCountSinceDate",
    "slots": {
      "SinceDate": {
        "name": "SinceDate",
        "value": "2017-12"
      }
    }
  },
```

Listing 3.7: Service Response of the Intent Request with {SinceDate} slot

3.4.2 Lambda Function Console Testing

An easier way of testing the code, with more descriptive details about what events are triggered or where an error has occured, is the Lambda function console under test event type which can be configured for appropriate event type.

A test event is a sample piece of JSON data that is passed into the Lambda function, to see what the result would have been, had it been called by the specific intent. Example templates are available for this type of testing provided by the Lambda console, one of which is the Alexa Start Session template, which mocks out the data the actual cloud service would call.

Furthermore, despite the templates being available, the Service Request from the Amazon developer console can be copied to test specific event. This JSON data is copied from the Service Request and pasted into the event configuration in the Lambda console. The test event is saved and consequently triggered. The execution results are generated in a form of log window outputs and summary of the action. To demonstrate this type of testing the `GetVideoViewCount` intent service request was pasted to the event configuration window and the function was tested. The log output is illustrated in the Listing 4.8. It is noticeable how the session is executed and followed by the Intent Request call out. Further information of the execution are displayed in the log output such as duration, billed duration, memory size and memory used.

```
START RequestId: c894bbb8-de80-11e7-8953-290071d53a44
   Version: $LATEST
2017-12-11T14:37:16.334Z  c894bbb8-de80-11e7-8953-290071d
   53a44  NEW SESSION
2017-12-11T14:37:16.337Z  c894bbb8-de80-11e7-8953-290071d
   53a44  INTENT REQUEST
END RequestId: c894bbb8-de80-11e7-8953-290071d53a44
REPORT RequestId: c894bbb8-de80-11e7-8953-290071d53a44
   Duration: 585.09 ms Billed Duration: 600 ms   Memory
   Size: 128 MB Max Memory Used: 23 MB
```

Listing 3.8: Lambda function log output

3.4.3 Amazon Echo device and Alexa Skill testing tool emulator

During the development and testing of the skill, the Amazon Echo device was not available to use. However, this issue was resolved by an Alexa Skill testing emulator provided by „Echoism.io“, that allows the developer to link their Amazon developer account with this website and test any skill assigned to their account [16]. This type of test was the most convenient one, for the interaction between the user and the device is enclosed and the developer can construct the communication accordingly. While testing the interaction, a few sample utterances were added to enrich the conversation with more realistic feeling.

4 CREATION OF A CUSTOM WEATHER SKILL

With the considerable amount of new Alexa skills being developed everyday, it is essential to consider the requirements for the respective skills such as originality, helpfulness, responsiveness and for the most part personal or commercial implementability.

The Amazon Alexa Skill Store provides listings of the most popular skills on the marker divided into various categories. Ranging from Business and Finance skills that inform users about the fluctuations of different currencies and commodities, through Games, Trivia and Entertainment skills providing interactive for of entertainment such as guessing games or the "Joke of the day" skill, to Smart Home skills that allow a responsive control over Alexa enabled devices such as lights, door locks and air conditioning.

A research was conducted to determine which skill types could be beneficial for the user. The trending page of the Amazon Alexa Store shows that the most popular Alexa skills were either informational such as news reports, or leisure Alexa skills such as music playing skills. For this particular reason the decision was to create an informational skill that would gather real–time weather forecast data and would inform the user and suggest the appropriate windsurfing equipment for the current weather forecast conditions.

This skill would be aimed at windsurfers, in particular windsurfers that are in the initial process of learning the sport and could potentially have trouble with determining the right equipment for the wind conditions. In comparison to the previous skill, see Section 3, this skill will require a more complex interaction model, for the interaction will consist of higher number of communication layers. For instance, the invocation phrase will include the respective location followed by the response of the weather forecast details with the possibility offered to the user, whether the user wishes to receive the suggestions for the suitable equipment.

The weather forecast data requires extraction from a weather forecast source. The decision was to opt for an API source website OpenWeatherMap.org. This API service offers 5 subscription plans: free, startup, developer, professional and enterprise. The distinguishing element between these subscription plans is the API request calls per minute, ranging from 60 calls per minute to 200000 calls per minute. For the purpose of this Alexa skill, 60 calls per minute will suffice. This API, besides the current weather forecast data, also provides a 5 day/3 hour update forecast which suits this skill exceptionally.

Furthermore, this API provides the access to current weather data for any location including over 200000 cities, which data is frequently updated from more than 40000 weather stations. The API data is available in JSON, XML or HTML format and do not require any payment. This applies to 5 day/3 hour update forecast as well. For the API to be accessible however, an account needs to be registered to acquire the API key [17], more on which mentioned in Section 4.4.

4.1 Interaction model of the skill

The user invokes the skill with a predefined invocation phrase and Alexa triggers the Launch Request with a response to the user, asking him/her to specify the particular place for which the weather forecast should be looked up. User then responds with an utterance specifying the windsurfing spot location.

Based on the utterance an Intent Request is triggered and an API call is issued. After parsing the data pulled from the API request a response is structured, including the forecast data, followed by a question whether the user wishes to receive a recommendation for an appropriate windsurfing equipment. For the additional input it is essential for the session to remain opened, because of the parsed data are stored in memory and will be consequently used in following part. If the user responds with "NO" utterance, the session ends. If the response is "YES" the application responds with a predefined data based on the weather conditions.

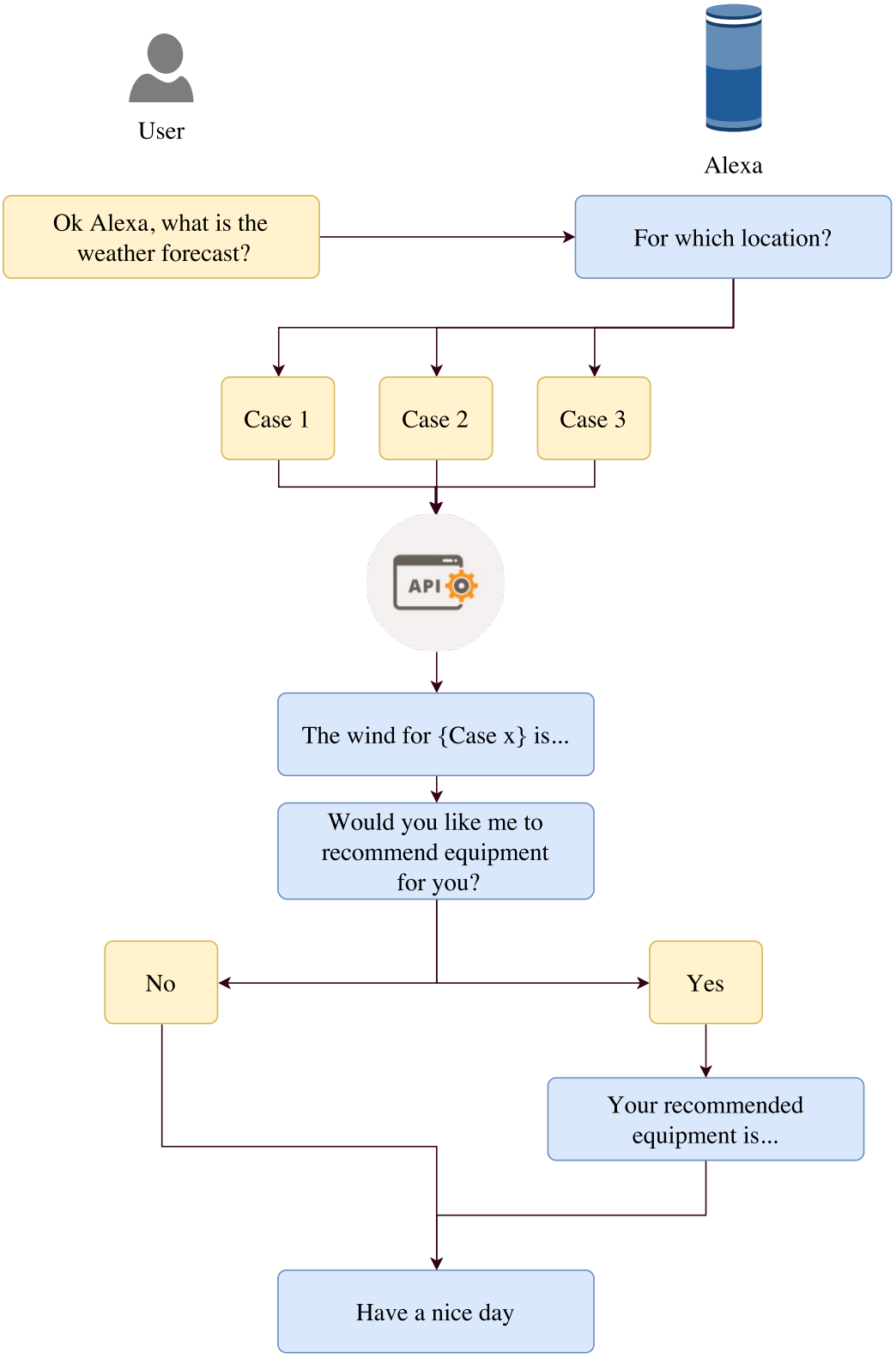

Fig. 4.1: Weather skill interaction model

4.2 Amazon Developer Portal

During the development of the weather skill the Amazon Developer Portal underwent two reconstructions that mainly increased the simplicity of use. As mentioned in Section 3.1, the interaction model consists of three elements, the intent schema, slots and sample utterances.

The rework of the Amazon Developer Portal merged these parts into one intuitive configuration console, this allows the user to define the whole interaction schema in few easy continual steps. The first update of the Amazon Developer Portal allowed for switching to the old console for an easier adjustment transition for the developers. The second update got rid of the old developer console completely.

Upon creation of the new skill, the console is presented with the options to configure the Interaction Model, Interfaces, Endpoint and the Account Linking. Intents are defined starting with the Intent name followed by sample utterances. Slots can be defined immediately just by selecting a section in the sample utterance. With this custom skill, built-in intents were used such as `AMAZON.YesInetent`, `AMAZON.NoIntent` and `AMAZON.HelpIntent`. These built-in intents have predefined invocation utterances that simplify the development process.

Custom intents were defined as well, the intent schema is illustrated in Listing 4.1. The weather forecast will be based on the location that the user defines. This can be done by defining a custom slot `{City}` that will recognize the city utterance the user specifies, the Lambda function would however require the code to accompany this, more on which in Section 4.2.1. This code should be able to pull a specific `CITY_ID` from the API response. The second option is to define custom locations that the user would be most interested in.

Two ways of defining the intents is presented. Definition by the amazon developer editor or definition by an in-line JSON editor. Regardless of the choice the developer makes while defining the intent schema, these editors update with every change that is done in either one of them.

Moving onto the next part of the Developer portal, the Interfaces. The Interfaces tab provides the developer with additional built-in interfaces that, if enabled, will generate intents required for the custom skill to work. Presented interfaces are following:

- The AudioPlayer interface that provides directives and requests for streaming audio and monitoring playback progression
- Display Interface for the Echo Show that allows skill developers to create skills for Alexa that use both screen and voice interaction
- The VideoApp interface that provides the VideoApp.Launch directive for streaming native video files in Echo Show
- Alexa Gadget for skills using the Gadget Controller Directives or Game Engine Inputs, Gadget Controller for the control over the Alexa Gadget and Game Engine for receiving input from the Alexa Gadget (this interface was in BETA development during the development of the skill)

The Endpoint tab includes the slots for the Amazon Resource Numbers if the developer wishes to accommodate different Lambda functions for different regions. Another option is to accommodate a user defined HTTPS web service endpoint that will be managed by the user. This section also includes The Skill ID which is used to configure the Lambda trigger configuration. This Skill ID and the Amazon Resource Number were required for the source linking of the Amazon Developer Portal with the Lambda function.

The Amazon Developer Portal also includes the Test, Launch and Measure tabs. The Test tab received the most significant upgrade from the previous version in a form of an Alexa Simulator. This eliminated the need of the Echoism.io Alexa emulator. This part will be described in detail in the 4.5.1 Section.

4.2.1 Actions, Entity Types and Properties

The intents in the built-in intent library are defined by a set of actions, entity types, and properties. Similar to a programming language, the actions and entity types are classes that have properties. An intent signature is assembled by these building blocks into a name that can be used in the intent schema.

An `action` represents the action that the user defines, such as searching for information. For instance, `SearchAction` represents the "searching for information" action. An intent defined with `SearchAction` indicates that the user wants search or look up information, such as finding a weather forecast. An intent defined with `AddAction` indicates that the user wants to add information somewhere, such as adding an event to a calendar or adding the title a book to a reading list.

Most actions operate on other objects or use other objects. A search action needs to identify the information to search for. An add action needs to identify the item to add and possibly also the list to which the item should be added to. These items are specified using the properties of the action.

For example, `SearchAction` has an object property that identifies the type of information the user wants to find. `AddAction` has a `targetCollection` property to identify the type of the target list and an `object` property to identify the type object to add to the list.

An entity type represents the objects that the actions can act on or use. Entity types also have attributes, which are represented by properties. For example, `WeatherForecast` is an entity type that defines a weather forecast. It has properties such as location, duration, and temperature (among others).

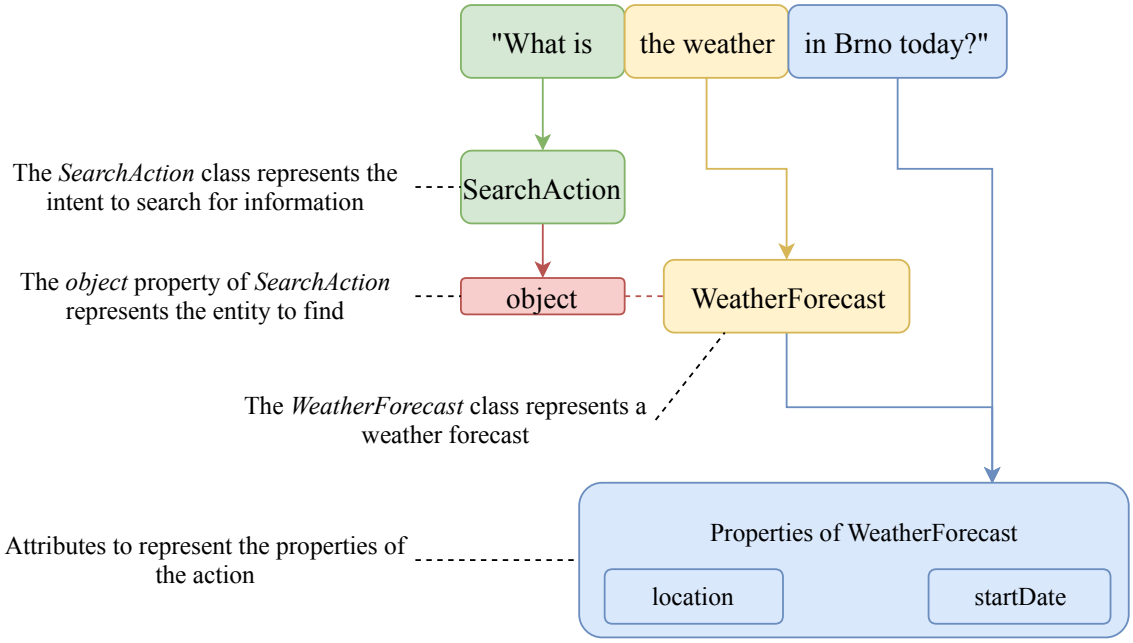

Fig. 4.2: Weather forecast invocation utterance

- The `SearchAction` action represents the user's intent to search for information
- The `object` property of `SearchAction` represents the entity type to find
- The `WeatherForecast` entity type represents a weather forecast
- The `WeatherForecast` entity type has properties that represent possible attributes of a weather forecast, such as the location and date (`location` and `startDate`)

4.2.2 Property Values Passed as Slot Values

Entity types such as `WeatherForecast` have their own properties that further define its attributes, such as `startDate`, `location` and `duration`. When users say utterances that invoke an intent defined with `WeatherForecast`, they may mention values that correspond to these attributes, such as the `date` and `location` of the weather forecast they want. These property values are conveyed to the skill as slots.

Unlike with custom intents, the user does not declare these slots in the intent schema. Alexa recognizes the slots from the properties of the objects included in the intent signature. If the user mentions any possible values for those slots, those values are provided in the request as slot values. The property name is used as the slot name.

For example, in the utterance "what is the weather today," the word "today" corresponds to a possible value for the `WeatherForecast.startDate` property. Assuming `AMAZON.SearchAction<object@WeatherForecast>` is included in the intent schema, this utterance sends a request to the skill.

```
{
  "languageModel": {
   "intents": [
     {
      "name": "AMAZON.CancelIntent",
       "samples": []
     },
     {
       "name": "AMAZON.HelpIntent",
       "samples": []
     },
     {
       "name": "AMAZON.NoIntent",
       "samples": []
     },
     {
       "name": "AMAZON.StopIntent",
       "samples": []
     },
     {
       "name": "AMAZON.YesIntent",
       "samples": []
     },
     {
       "name": "GetCityData",
       "samples": [
         "what is the wind forecast for {City}",
         "current weather for {City}",
         "current weather forecast for {City}",
         "current wind forecast for {City}",
         "{City}"
       ],
       "slots": [],
    "invocationName": "my windsurf data"
  }
```

Listing 4.1: Intent Schema of the Wind Skill

4.3 AWS Lambda Function

Based on the previous work on the custom Youtube skill, see 3.2 Section, a similar approach was followed during the development of the custom Alexa Weather skill. A small part of the Youtube skill code laid the foundation and served as a stepping stone for the following development.

As mentioned in Section 3.2, the Lambda function was wrapped inside a function handler with a signature `exports.handler = (event, context) {}`. A `switch` statement is run for different request types such as launch request, intent request and session ended request. The `LAUNCH REQUEST` is triggered by an invocation phrase „my windsurf data“, with a direct response informing about the skill capabilities „Welcome to weather forecast skill. This skill can inform you about wind forecast for Banska Bystrica, Bratislava and Brno, which location would you like to get the weather forecast for?“. This request leaves the session open for additional input.

Helpers are again required to generate the data structure returned from the Lambda function onto the Amazon Alexa Service to pass to the Echo device or to the simulated device. For the different intent types another `switch` statement is used with `(event.request.intent.name)` parameters.

Even though, built–in intents have predefined sample utterances, they have to be defined in the Lambda function. `AMAZON.NoIntent`, `AMAZON.HelpIntent` and the `AMAZON.CancelIntent` respond with a predefined responses.

For the `AMAZON.YesIntent` a function was created that generates the response based on the `windSpeed` parameter pulled from the API response.

The code logic for receiving the data from the API and handling them is following. Whenever a new intent is invoked an http endpoint is defined with variables used for handling the response. Then an HTTP request, get the response, parse it and get the `data.wind.speed` and `data.wind.deg` parameters from the `api.openweathermap.org` API. At first the skill returned the wind direction parameter as a number, this however might confuse users so an `if` statement was implemented. This `if` statement takes the `windDeg` value, compares it with conditions illustrated in Listing 4.2 and passes it to the `windDegString` variable.

Moreover, the response does no longer include a number but a Cardinal direction. Following the `if` statement, if successful the `content.succeed` generates the text response with the `windSpeed` and `windDegString` parameters. This process is illustrated in Listing 4.2.

Test events, generated from the Amazon Developer Portal, allow for a real time testing of the code inside the Lambda Function, more on which in 4.5.2.

```
case "IntentRequest":
console.log(‘INTENT REQUEST‘)
  switch(event.request.intent.name) {
     case "GetCaseOneData":
     var endpoint = "http://api.openweathermap.org/data
   /2.5/weather?id=CITYID&APPID=APIKEY&units=metric"
     var body = ""
     https.get(endpoint, (response) => {
        response.on('data', (chunk) => { body += chunk })
        response.on('end', () => {
           var data = JSON.parse(body)
           var windSpeed = data.wind.speed
           var windDeg = data.wind.deg
           var windDegString
            if(windDeg >= 316 && windDeg <= 45){
                windDeg = windDegString;
                windDegString = "North";
            }else if(windDeg >= 46 && windDeg <= 135){
                windDeg = windDegString;
                windDegString = "East";
            }else if(windDeg >= 136 && windDeg <= 225){
                windDeg = windDegString;
                windDegString = "South";
            }else if(windDeg >= 226 && windDeg <= 315){
                windDeg = windDegString;
                windDegString = "West";
            }
               context.succeed( //in case of success
               generateResponse(buildSpeechletResponse(‘
   The weather forecast for Banska Bystrica is ${
   windSpeed} meters per second with ${windDegString}
   direction. Would you like me to recommend some
   equipment for you? ‘, false)
             ))
        })})
     break;
```

Listing 4.2: Weather data code logic

4.4 Weather forecast API

The particular API source for the weather forecast gathering used in this skill provides API calls examples: `api.openweathermap.org/data/2.5/weather?q=London`.

This API link provides a syntax structure that includes data available for the developer to work with. If however, the developer requires an API call for another city, and the London string is changed to Brno for example, the URL response indicates an error of an invalid API key, for the API link was classified as a sample API.

Openweathermap.org used to be one of the examples that did not require an API key and would provide the weather forecast data anyway without the need of an authentication. This however is no longer the case, for the companies that open up and provide their data, to developers or other companies, require the identification of the users when requesting the data. An account needs to be created for the individual API key to be generated. That way the company can track the frequency of data pulls, what the data is used for and ultimately to keep track of this paid service.

The Openweathermap.org can be however used free of charge, as mentioned in 4 Section, if however an application is developed for a significant amount of users that query the weather data, a fee needs to be paid to the provider, in this case Openweathermap.org to facilitate the queries.

Moreover, an account was created for the API to be accessible and functional. The `http` request includes the city ID element and the API key element: `http://api.openweathermap.org/data/2.5/weather?id=CITY&APPID=APIkey`.

This now functioning API call responds with JSON data structure that includes all necessary weather forecast data. This chunk of JSON data was rather illegible for there was no proper formatting for the developer to clearly read out the data. For the JSON structure to be more legible to the developer an extension for the chrome browser was downloaded and installed.

This extension formatted the chunk data into the root elements of the API response that include the individual data provided by the API, illustrated in Listing 4.3.

After reviewing the documentation and the API call response an additional adjustment needed to be done to the URL which was the change of the unit system that the data is represented to the metric system by adding `&units=metric` to the URL.

```
{
"coord": {
"lon": 16.61,
"lat": 49.2
},
"weather": [
{
"id": 800,
"main": "Clear",
"description": "clear sky",
"icon": "01d"
}
],
"base": "stations",
"main": {
"temp": 0,
"pressure": 1011,
"humidity": 66,
"temp_min": 0,
"temp_max": 0
},
"visibility": 10000,
"wind": {
"speed": 5.7, //requested parameter
"deg": 320 //requested parameter
},
"clouds": {},
"dt": 1521549000,
"sys": {},
"id": 3078610, //List of the City IDs was included in the
    documentation
"name": "Brno",
"cod": 200
}
```

Listing 4.3: JSON structure of the API response for Brno

4.5 Testing of the Skill

APIs of the weather skill were successfully returned and their data parsed. Based on their values a response was structured. This skill was subjected to different testing scenarios to determine its flaws and functionality, enclosing the development process in Amazon developer portal, source linking of the Lambda function, APIs reference and tests of the developed skill with the help of the Amazon developer console and Lambda test events.

The skill behaved according to the programming with a few speech recognition failures. Considering that the Alexa can translate the speaker output to Slovak language but has trouble analyzing some speech elements, such as Slovak city names in this case, some errors are expected to occur.

4.5.1 Amazon Developer Console Testing

As mentioned in the 4.2 Section, the rework of the Amazon Developer Portal opened new and more convenient way of testing of the Weather skill. For the most part, it was caused by the implementation of the Alexa voice emulator to the Amazon Developer Portal that eliminated the need of a third party Alexa voice emulator, `Echoism.io`. There are four options available for testing in the Alexa simulator:

- Alexa Simulator: Voice input
- Alexa Simulator: Text input
- Manual JSON input
- Voice & Tone input

The Alexa Simulator with option of a voice or text input allows for the most convenient testing from options listed above, with a direct response from the simulated device logs. For this type of testing, `IntentRequest` for the city Bratislava (`GetCaseTwoData`) was chosen. This test is divided into two parts, starting with the `GetCaseTwoData`, see Listing 4.4 with a response, see Listing 4.5. Following onto the `AMAZON.YesIntent`, see Listing 4.6 with a session ending response from the simulated device, see Listing 4.7.

After inserting the phrase „Ask my windsurf data for current wind forecast for Bratislava“ or invoking it with an utterance, JSON input is generated. If successful, JSON output is generated as well, including the response from the simulated device „The weather forecast for Bratislava is 5.1 meters per second with South direction. Would you like me to recommend some equipment for you?“.

```
{ "version": "1.0",
  "session": {
    "new": true,
    "sessionId": "amzn1.echo-api.session.57e0db6e",
    "application": {
      "applicationId": "amzn1.ask.skill.711c0d9d"
    }
  },
  "context": {"AudioPlayer": {"playerActivity": "IDLE"
    },
      "user": {"userId": "amzn1.ask.account.AFOQQQK"
      },
      "device": {"deviceId": "amzn1.ask.device.AHJQG",
        "supportedInterfaces": {
          "AudioPlayer": {},
          "Display": {
            "templateVersion": "1.0",
            "markupVersion": "1.0"
          }
        }
      },
      "apiEndpoint": "https://api.amazonalexa.com",
      "apiAccessToken": "eyJ0eXAiOiJ"
    }
  },
  "request": {
    "type": "IntentRequest",
    "requestId": "amzn1.echo-api.request.a4b92bb2",
    "timestamp": "2018-04-04T10:38:45Z",
    "locale": "en-US",
    "intent": {
      "name": "GetCaseTwoData",
      "confirmationStatus": "NONE"
    }
  }
}
```

Listing 4.4: JSON input for GetCaseTwo Intent

```
{
  "body": {
    "version": "1.0",
    "response": {
      "outputSpeech": {
        "type": "PlainText",
        "text": "The weather forecast for Bratislava is 5
   .1 meters per second with South direction. Would you
   like me to recommend some equipment for you? "
      },
      "shouldEndSession": false //session remains open
    },
    "sessionAttributes": {}
  }
}
```

Listing 4.5: JSON output for GetCaseTwo Intent

After this response, the session remains open and awaits a follow–up input. An assuring element of whether the session remains open is `"shouldEndSession": false` parameter illustrated in Listing 4.5.

Furthermore, `AMAZON.YesIntent` with the invocation phrase „yes“ was inserted with a positive response, see Listing 4.7, „A bigger board is recommended, around 150 liters with a 10 square meters sail. Perfect conditions to get used to pulling the sail from the water“. This response also ends the session, see Listing 4.7.

During the voice input testing an error that was occurring was caused by the Alexa's inability to analyze some speech elements, Slovak city names in this case, this was caused also due to the developer's accent.

The Manual JSON does not maintain sessions. Each operation is a single turn operation or one-off exchange. Moreover, this type of testing was not used.

The Voice & Tone can implement Alexa's response output and personality. This text to speech simulator supports SSML (Speech Synthesis Markup Language) using Alexa's voice. As an example, `<amazon:effect name="whispered">` changes Alexa's voice to a soft whisper tone. This however, was not needed nor desired considering this particular skill and was not used.

```
{ "version": "1.0",
  "session": {
    "new": false,
    "sessionId": "amzn1.echo-api.session.315b674e",
    "application": {
      "applicationId": "amzn1.ask.skill.711c0d9d"
    }
  },
  "context": {"AudioPlayer": {"playerActivity": "IDLE"
    },
      "user": {"userId": "amzn1.ask.account.AFOQQQK7"
      },
      "device": {"deviceId": "amzn1.ask.device.AHJQGK6",
        "supportedInterfaces": {
          "AudioPlayer": {},
          "Display": {
            "templateVersion": "1.0",
            "markupVersion": "1.0"
          }
        }
      },
      "apiEndpoint": "https://api.amazonalexa.com",
      "apiAccessToken": "eyJ0eXAiOi"
    }
  },
  "request": {
    "type": "IntentRequest",
    "requestId": "amzn1.echo-api.request.c84e8389",
    "timestamp": "2018-04-11T10:39:34Z",
    "locale": "en-US",
    "intent": {
      "name": "AMAZON.YesIntent",
      "confirmationStatus": "NONE"
    }
  }
}
```

Listing 4.6: JSON input for AMAZON.YesIntent

```
{
  "body": {
    "version": "1.0",
    "response": {
      "outputSpeech": {
        "type": "PlainText",
        "text": "A bigger board is recommended, around 150 liters with a 10 square meters sail. Perfect conditions to get used to pulling the sail from the water"
      },
      "shouldEndSession": true //session ends
    }
  }
}
```

Listing 4.7: JSON output for AMAZON.YesIntent

4.5.2 Lambda Function Console Testing

The majority of testing was done in the Lambda Function Test Environment. Mostly due to the ease of use with the help of the Service Requests generated in the Amazon Developer Portal. With every implemented function, a `console.log('action')` was added to see if the `action` was successful. Following the steps from previous testing, see 3.4.2 Section, a JSON input was copied from the Amazon Developer Portal to the test event section in the Lambda function. After triggering the test event a log output was generated with a summary of the action that took place. This log is illustrated in Listing 4.8 with a `GetCaseTwoData` for the location Bratislava.

```
START RequestId: 2aeb85bd-3fc7-11e8-abd5-cf7dd649f637 Version: $LATEST
2018-04-14T09:35:28.917Z  NEW SESSION
2018-04-14T09:35:28.917Z  INTENT REQUEST
END RequestId: 2aeb85bd-3fc7-11e8-abd5-cf7dd649f637
REPORT RequestId: 2aeb85bd-3fc7-11e8-abd5-cf7dd649f637 Duration: 311.66 ms Billed Duration: 400 ms  Memory Size: 128 MB Max Memory Used: 21 MB
```

Listing 4.8: Lambda function log output

5 CONCLUSION

The main focus of this bachelor thesis was to explore the possibilities and capabilities of the Amazon Alexa Skills Kit and identify functions not commercially implemented.

The theoretical part discussed the history of the „smart home“ concept, the shift in smart device implementation. Furthermore, it separated the elements of the user experience and briefed on techniques used in speech recognition. Its main focus however, was to describe the Amazon Web Service, more specifically its cloud computing functionality, the Amazon Echo device and the Amazon Alexa Skills focusing on various skill types and their capabilities.

In the first part of practical section, new custom skill was developed and subsequently subjected to different testing scenarios to determine its flaws and functionality. The function of this skill was to inform about the user's Youtube channel analytics. This section enclosed the development process in Amazon developer portal, source linking of the Lambda function, Google APIs reference and tests of the developed skill with the help of the Amazon developer console, Lambda test events and the `Echosim.io` Alexa emulator.

The resulting skill includes introduction of itself upon the launch request and can return current video view count and current subscriber count of the Youtube channel. These two values can not be however requested based on a time period parameter, as was originally intended.

During the development of the custom Youtube skill an understanding of the processes required for the skill to function was achieved. A basic knowledge of the JSON data structure and JavaScript programming language was gained and this skill laid the training ground foundations for further improvements and expansion of this skill and development of the following skill.

This particular skill could be expanded exponentially to inform the user about their Youtube channel statistics, popularity of videos residing on the channel or comments on their videos. The dysfunctional intent functions, as stated in the Section 3.4.1, were caused by insufficient understanding of the API structures and requirements.

The second part of the practical section was focused on the development of the second, commercially not implemented Amazon Alexa Skill.

After conducting the research of the Amazon Skill Marketplace, the decision was to develop a custom weather Alexa skill that would inform users about current wind forecast for a certain locations and, if desired, would recommend suitable windsurfing equipment for the forecast.

This weather forecast data had to be pulled from a source, the decision was to use a web service `http://openweathermap.org/` that was free for this type of research use. With all necessary resources an interaction model of the skill was created to clear out the functionalities of the skill, see Fig. 4.1.

The skill was registered on the Amazon Developer Portal, authorized and source linked with the Lambda Function where the JavaScript code resided.

Furthermore, the endpoints were implemented to the Lambda code and the skill was consequently tested. Tests were conducted inside the Amazon Developer Portal Test Console and the Lambda Function Test Event Console. Each of these test utilities has qualities suitable for different types of tests. The skill behaved according to the programming with a few speech recognition failures.

Considering the functionality and the purpose of the skill, improvements could be made to the interaction dialogue generated from the device, this however is a minor retouch that could be changed. Additional data could be pulled from the `http://openweathermap.org/`, to provide a more detailed review of the weather conditions for the particular locations.

BIBLIOGRAPHY

[1] R. Harper, *Inside the smart home*. Springer Science & Business Media, 2006.

[2] S. Davidoff, M. K. Lee, C. Yiu, J. Zimmerman, and A. K. Dey, "Principles of smart home control," in *International conference on ubiquitous computing*. Springer, 2006, pp. 19–34.

[3] B. S. Gardner, "Responsive web design: Enriching the user experience," *Sigma Journal: Inside the Digital Ecosystem*, vol. 11, no. 1, pp. 13–19, 2011.

[4] D. Leonard-Barton and D. K. Sinha, "Developer-user interaction and user satisfaction in internal technology transfer," *Academy of Management Journal*, vol. 36, no. 5, pp. 1125–1139, 1993.

[5] B.-H. Juang and L. R. Rabiner, "Automatic speech recognition–a brief history of the technology development," *Georgia Institute of Technology. Atlanta Rutgers University and the University of California. Santa Barbara*, vol. 1, p. 67, 2005.

[6] H. Irving, "Science and music," 1938.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[8] R. Dale, "The return of the chatbots," *Natural Language Engineering*, vol. 22, no. 5, pp. 811–817, 2016.

[9] S. Bhardwaj, L. Jain, and S. Jain, "Cloud computing: A study of infrastructure as a service (iaas)," *International Journal of engineering and information Technology*, vol. 2, no. 1, pp. 60–63, 2010.

[10] Amazon.com, "Aws elastic beanstalk," 2017.

[11] A. Weber and A. Jones, "Amazon echo: The best user guide to learn amazon echo and amazon prime membership (amazon prime, users guide, web services, digital media, amazon... movie, prime music)," 2016.

[12] IFIXIT, "Amazon echo teardown," 2014. [Online]. Available: https://www.ifixit.com/Teardown/Amazon+Echo+Teardown/33953

[13] Amazon.com, "Echo & alexa devices," 2017.

[14] "Amazon alexa skills kit documentation." [Online]. Available: https://developer.amazon.com/alexa-skills-kit

[15] T. Bray, “The javascript object notation (json) data interchange format,” 2014.

[16] “Omnet,” 2017. [Online]. Available: https://echosim.io/

[17] “Current weather and forecast.” [Online]. Available: http://openweathermap.org/

LIST OF SYMBOLS, PHYSICAL CONSTANTS AND ABBREVIATIONS

API	Application Programming Interface
ASK CLI	Alexa Skills Kit Command Line Interface
ARN	Amazon Resource Number
AWS	Amazon Web Service
AVS	Alexa Voice Service
CTC	Connectionist Temporal Classification
EC2	Elastic Compute Cloud
ESP	Echo Spacial Perception
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
JSON	JavaScript Object Notation
LSTM	Long short–term memory
PaaS	Platform as a Service
RAM	Random Access Memory
RSS	Rich Site Summary
SDK	Software Development Kit
SSML	Speech Synthesis Markup Language
URL	Uniform Resource Locator
XML	Extensible Markup Language

A CONTENT OF ENCLOSED CD

Directories present on the CD include Source files of the Lambda function in the JavaScript `srclambda.js` file and the Intent schema in the JSON structure `intentschema.js` file. Furthermore, the CD includes the master file of the Bachelor's thesis in `.pdf` format.

```
/ ..... Root directory of the included CD
├── Source Files ..... Source Files
│   ├── srclambda.js
│   └── intentschema.js
└── BPDobrík.pdf ..... Bachelor's thesis
```