Vaasan yliopisto
UNIVERSITY OF VAASA

Topias Jokiniemi

Voice controlled audiobook reader software for visually impaired

The School of Technology and Innovations
Master`s Thesis in Software Engineering
Degree Programme in Information Technology

VAASA 2021

Contents

ABBREVIATIONS

AI	Artificial intelligence
AoG	Actions on Google, Platform to develop applications for Google Home device
HTTP	Hypertext, transport protocol, protocol for data transfer
HTML	Hypertext Markup Language, ruleset designed to be displayed in browser
IDE	Integrated development environment
NLP	Natural language processing, interaction with computer and human language
NLU	Natural-language understanding, AI field that handles reading comprehension
POC	Proof-of-concept
REST	Representational state transfer, stateless software architecture for webservices
SDK	Software development kit, collection of software development tools
SMIL	Synchronized Multimedia Integration Language, XML based language to define audio and text synchronization and navigation
SQL	Standard Query Language, Standard language for databases
SSML	Speech synthesis markup language, XML based speech synthesis language
VUI	Voice user interface
XML	Extensible Markup Language, ruleset for encoding documents
XHTML	Extensible Hypertext markup language, xml-based version of html

UNIVERSITY OF VAASA
The School of Technology and Innovations

Author:	Topias Jokiniemi
Topic of the thesis:	Voice controlled audiobook reader software for visually impaired
Degree:	Degree Programme in Information Technology
Master's Programme:	Software Engineering
Supervisor:	Jouni Lampinen
Year of entering the University:	2015
Year of completing the thesis:	2021
Number of pages:	62

ABSTRACT:

This thesis results in a functional proof-of-concept level application that will help Pratsam Oy Ab to determine if voice-controlled audiobook player developed for Google Home device has potential to be a quality product. This thesis describes the development and the research of functional system that could be used as a starting point, in case it would be developed into a full product.

Before committing to a product development, software companies might want to test if the product concept is viable. The concept viability can be verified with research into possible development problems and with proof-of-concept level software. The proof-of-concept can explore if the product can be built at all, and if the quality would be high enough, so that it results in a profitable product.

In this product concept the user would use voice to control and to play Daisy 2.02 format audiobooks, which is an audiobook format developed for visually impaired. The audiobook reading software would be developed for Google Home device without a visual user interface.

Main technologies of the thesis are voice user interface, speech recognition, text-to-speech technologies, Google cloud platform, cloud-based MySQL database, Java REST API backend and a Java console application to parse data from Daisy audiobook files into the database.

The proof-of-concept period did not fully prove the viability of the concept. Google however has given clues that the required features would be added later, which could make it worth it to start developing the concept further.

KEYWORDS voice user interface, Google Home Device, Google Assistant, smart speaker

1. INTRODUCTION

According to Finnish National Institute for Health and Welfare there are 55 000 visually impaired in Finland (Näkövammaisten liitto, 2019). Currently they have ways of accessing library books with help of Celia. Celia provides free audiobook service for any visually impaired that would like to listen to books in Finland. The audiobooks can be listened with help of computers, mobile phones or even specially designed devices (Celia, 2019).

As major tech companies have developed voice controllable Assistants (Amazon Alexa, Google Assistant, Apple Siri, and Microsoft Cortana) and there are not any Voice controllable audiobook readers for Celia books. It is a good time to try out if the systems are mature enough for product development.

1.1. Background

This project was made under salaried contract for Pratsam Oy Ab. The project is exploring with a proof-of-concept if voice controllable systems are mature enough for a stable and high-quality product. Pratsam is a software company specialized in accessible audiobook and newspaper. Pratsam has developed mobile app and online web reader keeping especially hearing impaired in mind. Pratsam has also systems in place to automate the process of converting to audiobook standards that are used globally around the world. The biggest customers for Pratsam are Celia and The Association of Swedish Talking Newspapers in Finland (Pratsam, 2021).

1.2. Research problems and the objective

This thesis done proof-of-concept research, which means the result is working software that proves that main features of audiobook player can be developed and could the quality of a product be high enough. The researched features were decided by Pratsam prior to starting this thesis.

1.3. Structure of the thesis

The second chapter goes through the research relevant to voice controllable systems. The third chapter describes all the major technologies used in the development. The fourth chapter gives feature descriptions with examples. The fifth chapter goes through all the requirements and results in a plan for all the features and describes the system architecture as whole and the design of individual components. The sixth chapter describes how the implementation of the components and all features. The seventh chapter is about describes the results of the proof-of-concept and goes trough of all elements that could affect the decision Pratsam if the product is viable enough. The seventh and final chapter is my personal conclusion of the project.

1.4. Current state of audiobook services for visually impaired

Currently in Finland visually impaired can get Celia audiobooks for free if they sign up for it in their local library. Celia belongs to the administrative sector of the Ministry of Education and Culture. Celia is a non-profit organization that converts books into audiobook format. Celia also has their own systems to keep track of current audiobooks the user has loaned from the library. Celia audiobooks can be listened with use of computer browser, mobile apps or even with physical reader specifically designed for visually impaired (Celia 2019).

Audiobooks are converted into Daisy 2.02 standard, which is a globally used specification designed especially for visually impaired. While Daisy 3.0 standard was released in 2005, the predecessor 2.02 is still more used in Finland. Both standards are used to define structure of the book. They define navigation elements and have the text of the book synchronized with audio (Daisy: 2.02, 2001, Daisy: 3, 2005).

2. RELEVANT RESEARCH

2.1. Developing Voice user interfaces

Conversational voice user interfaces can be classified into three main types: Finite state-based systems, frame-based systems; and agent-based systems. Finite state-based system would guide the user through pre-determined path and asked single information questions until the user intent is resolved. Frame-based system is similar, but the path is more flexible and variable. User can answer with longer sentences containing more information of the user intent and the system will try to correct errors and understand more complete commands. Agent-based systems provide the most human like experience with AI and by not constraining to a flow of conversation but instead the user try to have conversation as he would have with human. More evolved agent-based systems even have a name such as Alexa, Siri, and Cortana (McTear, 2002).

Users are most commonly trying to resolve error situations in voice user interfaces by articulating sentences slower, more clearly and louder. This strategy was used on average 40.48% at the time. This strategy was the most common for all different kinds of errors. Other common strategies were simplification of the phrase, trying to word the intent differently, and using more information. Which all were uses 12% of the time (Myers, Furqan, Nebolsky, Caro & Zhu, 2018).

Designing Voice user interfaces should not be only translating visual user interfaces into voice user interfaces. Just replacing buttons with voice commands will not result in a great experience. For example, having visual reference of a list of data does not use the working memory the same way as listing the same data with voice user interfaces. Keeping lists short, maximum of seven items, preferably even shorter, makes it possible to remember all the options without stretching the limitations of the working memory (Gamm & Haeb-Umbach, 1995).

2.2. Accessibility

Children are using speech interfaces differently from adults. Children have harder time to reformulate their questions in case of an error within system. The younger the child the more hints they required to be able to formulate their intents into understandable form. Children are more often repeating their question instead of formulating it in a different way (Yuan et al., 2019). To design a speech interface to help users to effectively reformulate their questions, the system should be *"providing feedback on what was heard, asking for the missing context, clarify what is known, specifying formulations that are difficult for the system, and allowing context to be provided as a statement."* (Yuan et al., 2019 p. 86)

Elderly people prefer voice control over normal graphical user interfaces. 75% found the Voice user interface to be pleasant or neutral. For graphical user interfaces only 37,5% found them to be pleasant on neutral. The difference could be effect of declining eyesight and fine motor skills. Seniors also preferred female voice over male voices as female voice is more understandable. For elderly, a good error handling is extremely important in satisfaction as the usage can become tedious if the user gets stuck in voice user interface (Schlögl, Chollet, Garschall, Tscheligi, & Legouverneur, 2013).

2.3. Privacy

Users are more likely to use their voice assistants if they trust that their voice assistant device does not collect unnecessary information without user consent. Users are especially concerned about private conversations being listened without notice and location being tracked. Users expect to be in control of what data is being collected and want have possibility to erase the collected data. Most users want to have power off or at least mute button on their voice assistant devices (Yeasmin, Das, & Backstrom, 2020).

Users are usually placing their smart speakers in the most central location in their house. Without being concerned of the privacy. Privacy and security concerns are one of the major factors for not adopting smart speakers. Even if companies such as Amazon or Google do define what data they are collecting and how they are using it. Privacy concerned users are not trusting that the companies are following their own privacy agreements or expect them to change the agreements as they please. Interestingly the more user is expecting to benefit from the voice assistants the more they are also able ignore the privacy concerns (Lau, Zimmerman, & Schaub, 2018).

3. TECHNOLOGY

To develop for Google Home device, you are developing for Google Assistant. Assistant applications are done into platform called Actions on Google (AoG). Actions on Google platform allows developers to develop a conversational user interface also known as voice user interface (VUI). Structure of the VUI has similarities to simple console UIs back in the days before graphical user interfaces. These interfaces are menus of that guide user and help them select an action they want. AoG sends the text comprehension of what user said and asks backend system for an answer (Google: Assistant 2021).

Dialogflow is a Googles preferred way to handle voice user interfaces. Dialogflow allows developer to visually develop the conversation. Dialogflow uses ever learning AI to make sense of incomplete or incorrect sentences just as humans can understand multiple ways of saying the same thing. Dialogflow gives the developer a prediction of what was the intent of the users. Back-end systems such as REST API can be used to answer to the user with voice line guiding the conversation with user until the intent is found (Google: Dialogflow 2021).

Google Home has possibility to answer with text to speech Speech synthesis using SSML standards. Additionally, direct audio can be played which can be used to play recordings of books.

3.1. Google Home Device

Google Home is a smart speaker developed by Google. The device has Google Assistant that allows using the speaker with voice.

3.1.1. Google Assistant

Google Assistant is an AI powered virtual assistant developed by Google. Google assistant was initially introduced in May 2016 in Google Home smart speakers. At December 2016 Google launched Actions on Google Developer platform. Allowing third party developers to develop for Google Assistant (Verge 2016). In May 2017, the Assistant was deployed to Android and iOS mobile devices. Google has published 3rd party SDK in April 2019 to allow developers to develop for the smart speakers with Google assistant. After release of the SDK, it has been integrated to many systems including cars, fridges, ovens, and Raspberry Pi (Google: Assistant, 2021).

3.1.2. Devices

Google has developed multiple sizes for the speaker. All the speakers have the same Assistant features, but they differ in audio quality. At October 2019 Google published 2nd generation of Home speakers with name Google Nest. Nest has the same features as 1st generation of their smart speakers, but with better sound quality and added machine learning chip that handles commonly used commands locally (Techradar, 2019).

3.2. Speech recognition

Google assistant uses machine learning technology to convert speech into text. The text will be sent to Dialogflow, which has natural language processing artificial intelligence to determine user intent (Google: Dialogflow, 2021).

3.2.1. Dialogflow

Dialogflow is a platform used together with Actions on Google to manage conversional interfaces. Dialogflow gets users voice command from Actions on Google as text. The text is then analyzed by Dialogflows AI for user intent. All the intents need to be configured manually with use of example phrases for the AI to determine what was the user intent. The intent sends a webhook as HTTP POST request to a Developer made API that can describe to response back to the user. The response can be SSML format text or media file such as an audiobook audio file (Google: Dialogflow, 2021).

3.3. Speech synthesis and Speech Synthesis Markup Language SSML

Speech synthesis is artificial production of human speech. Google assistant supports Speech Synthesis Markup Language (SSML) for the configuration of phrases. SSML is an XML based language designed to help describe how synthase the sentence. SSML is great as it allows developer to customize sentences to be more human like with features like pauses after important phrase, emphasis sound levels, change of speed and possibility to describe how to interpret an element of a sentence (Google: SSML, 2021).

3.4. Google Cloud Platform

Google Cloud Platform is a set of cloud computing services provided by Google. Google has more than 90 different products under the brand name. While everything in Google Cloud comes with a cost. Google provides trial period worth for maximum of 300$ for new cloud developer to try out the products. The cloud platform can be used for web services, databases, to store data, machine learning, cloud computing and so on (Google: Cloud, 2021).

3.4.1. Actions on Google

Actions on Google is the main service that handles new applications used by Google Home devices. Actions on Google service allows user to change what voice commands to use to launch the application. In Actions on Google applications, all the users voice commands are converted to text and directed to Dialogflow for user intent analysis.

3.4.2. App Engine

App Engine is one of the services provided in Google Cloud. App Engine was designed for scalable web applications such as REST APIs. App Engine supports languages Go, PHP, Java, Python, Node.js, .NET, Ruby, and custom environment for more. Being in cloud provides possibility for scaling on demand if the request count for API increases (Google: App Engine, 2021).

3.4.3. Google Cloud Storage

Google Cloud storage cloud-based file hosting service that is easily accessed by other projects in a single Google Cloud project. File access can be defined as broad and as limited as is needed (Google: Storage, 2021).

3.4.4. Cloud SQL Database

Cloud SQL is a service that provides cloud-based MySQL, PostgreSQL, or SQL server. The database is automatically connected to App Engine based web service and access can be configured as needed. Google Cloud SQL has built in storage capacity manager and automatic backup systems to help with database management (Google: Cloud SQL, 2021).

3.5. Audiobook standard Daisy 2.02

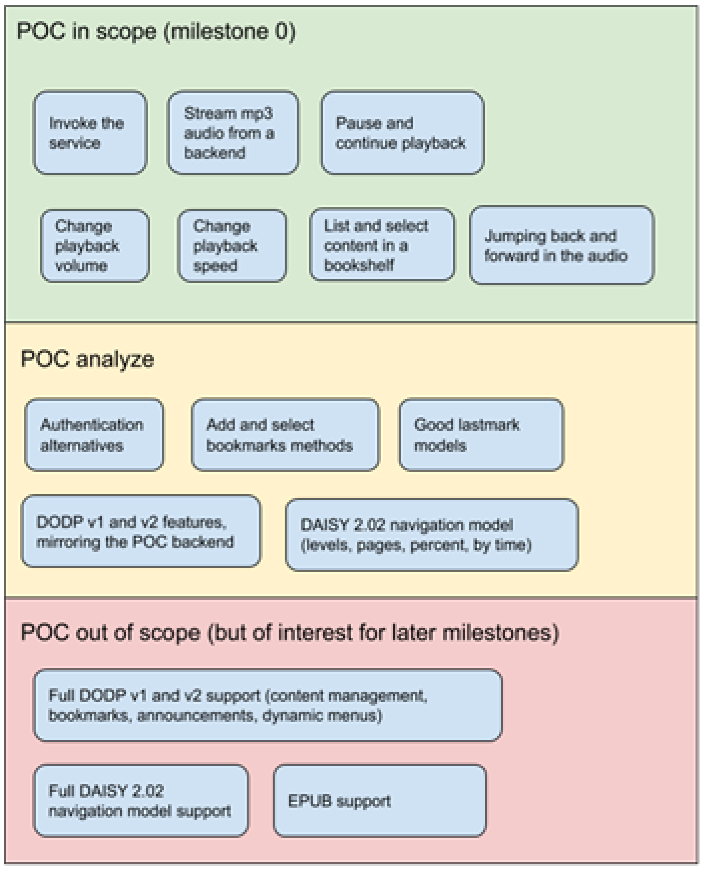
Daisy 2.02 is audiobook standard used by Celia. Defines how audiobook is structured and how user can navigate through the book. Daisy designed especially for visually impaired that require precise navigation and speech-audio synchronization. Daisy 2.02 is XML based markup language having its structure defined in NCC.html file (Daisy: 2.02, 2001).

4. REQUIREMENTS

Pratsam wants to develop proof-of-concept for reading audiobooks with Google Assistant using only voice commands. The picture presents what features are wanted for the resulting software. The first block in green “POC in scope (milestone 0)” has all the features required for Pratsam to decide if the product can be viable if fully developed.

The second block in yellow “POC analyze” are features that are not required, but if the analysis finds out that the features can be implemented in short amount of time Pratsam can decide to include them in proof-of-concept implementation.

The last block in red “POC out of scope (but interest for later milestones)” contains features that will be important after proof-of-concept period and will not be researched in this thesis.


POC in scope (milestone 0)
Invoke the service
Stream mp3 audio from a backend
Pause and continue playback
Change playback volume
Change playback speed
List and select content in a bookshelf
Jumping back and forward in the audio
POC analyze
Authentication alternatives
Add and select bookmarks methods
Good lastmark models
DODP v1 and v2 features, mirroring the POC backend
DAISY 2.02 navigation model (levels, pages, percent, by time)
POC out of scope (but of interest for later milestones)
Full DODP v1 and v2 support (content management, bookmarks, announcements, dynamic menus)
Full DAISY 2.02 navigation model support
EPUB support

Picture 1 Requirements

4.1. Feature Descriptions and Examples

Invoking the service using voice commands. Example of this would be *"Hey Google, start the audiobook reader"*
Stream audio from backend, user can start playing audiobook using voice commands.
Pause and continue playback using voice commands. Examples "Hey Google, pause the audio" and "Hey Google, continue"
Change playback volume using voice commands. Examples: "Hey Google, volume up", "Hey Google, Change the volume to 100%", "Hey Google, lower the volume".
Change playback speed of the audiobook. Hearing impaired are used to listening with great speeds, some even use double the normal speeds for everything from listening the news to listening books. For that reason, audiobooks should have possibility for up to double the speed. Examples: "Hey Google, speed up", "Hey Google, set speed to 200%", "Hey google, lower the speed".
List and select contents from a virtual bookshelf. Celia for example has library services in place for every user to borrowing audiobooks for limited time. All borrowed audiobooks are stored in a virtual bookshelf that user has access to. User can browse the current audiobooks and select one for listening. It is not required to have full implementation connected to library services. Instead, idea is to have similar proof-of-concept level implementation. User can browse and select a book from virtual bookshelf. Examples: "Hey Google, open my bookshelf", "Hey Google, select book <book name>".
Jumping back and forward in the audio. Users want to listen a part of the book again or just scroll through the book with use of voice commands. Common uses for this are when user did not hear a sentence or just wants to listen the exciting part of the audiobook again. User might also want to skip forward. Examples: "Hey google, go back 30 seconds", "Hey google, go to the previous chapter", "Hey google, jump forwards for 10 minutes".
Analyze authentication alternatives, how users can be identified and what authentication methods are available when using Google Home device.

Add and select bookmark Users want to save their favorite moments from a book. Each bookmark needs to be user based and should be usable even after closing the application. If possible, bookmarks should be available for all multiple Google home devices where user has authenticated. While reading a book user can add a bookmark and continue reading. Bookmarks need to be browsable and selecting a bookmark would result in audiobook continuing form bookmarks position. Examples: “Hey Google, Add bookmark”, “Hey Google, get my bookmarks for book <book name>”, “Hey Google, select bookmark one”.

Lastmark is the last position where user left when he closed the book. Like bookmark, user wants to easily continue the book he was reading last time from the position where he left. Examples “Hey Google, continue reading <book name>”, “Hey Google, read <book name> from the lastmark”.

DODP v1 and v2 features The Daisy Online Delivery Protocol is the universal protocol for distributing online books. Analyze how its features could be implemented in future.

Daisy 2.02 navigation model: levels pages percent, time

Analyze a way to implement Daisy 2.02 navigation features.

5. PLANNING

5.1. Research plan

The research is done using proof-of-concept research methodology. Where we implement a working software that tries to implement all major elements of an audiobook reader. To prove that it is possible to implement everything for more complete product. Software companies use proof-of-concept methodology to find out if a product idea is possible to implement with feasible cost and timeframe.

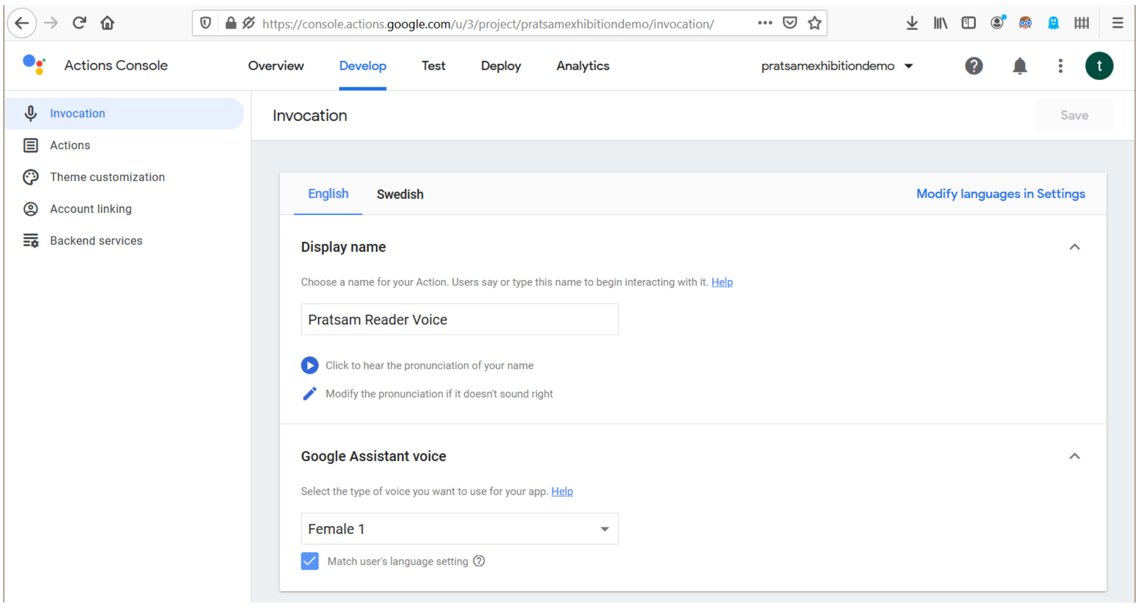
"The evidence that a product, technology or an information system is viable and capable of solving an organization's particular problem. A proof-of-concept is often developed for new products that have not yet come to market." (YourDictionary, 2021).

In this proof-of-concept period Pratsam is the most interested in the audio playback features of Google Home device. Secondary features such as bookmark and bookshelf are tested after the main features of Google Home device are developed. The main source of information will be the Google Assistant developer documentation and Google Cloud documentation.

During the proof-of-concept period the project will not be connected to actual library services or to any other products that Pratsam has developed. Instead, everything will be kept simple while keeping in mind that most of the functionality will be used if the project will be developed into a full product.

5.2. Research for the features

Invoking the service feature requires configuring the project on Actions on Google developer console. All Assistant applications are initially created in Actions on Google website. The website allows developer to make the Google Cloud project that will be used for the voice application. The invocation name can be set as soon as Actions on Google project is made and connected to a natural language processing (NLP) platform that handles the voice user interface. Google suggests using Dialogflow as it is Googles owned NLP platform. After the cloud project is connected to Dialogflow it can already answer to some voice commands such as Hello and answer back with a simple statement. After the invoking phrase is set it can be used on all devices where user has logged on with the same Google user account as the project was made with. Invoking the application with Google Home device can be done by the command “Hey Google, Talk to <invocation name>”. Invocation name can be changed at any time and it will work on all Google Assistant devices that that support media playback. Before the Assistant action is published to public, only the manually added test users can invoke the action.


https://console.actions.google.com/u/3/project/pratsamexhibitiondemo/invocation/

Actions Console | Overview | Develop | Test | Deploy | Analytics | pratsamexhibitiondemo

Invocation | Actions | Theme customization | Account linking | Backend services

Invocation | Save

English | Swedish | Modify languages in Settings

Display name

Choose a name for your Action. Users say or type this name to begin interacting with it. Help

Pratsam Reader Voice

Click to hear the pronunciation of your name

Modify the pronunciation if it doesn't sound right

Google Assistant voice

Select the type of voice you want to use for your app. Help

Female 1

☑ Match user's language setting

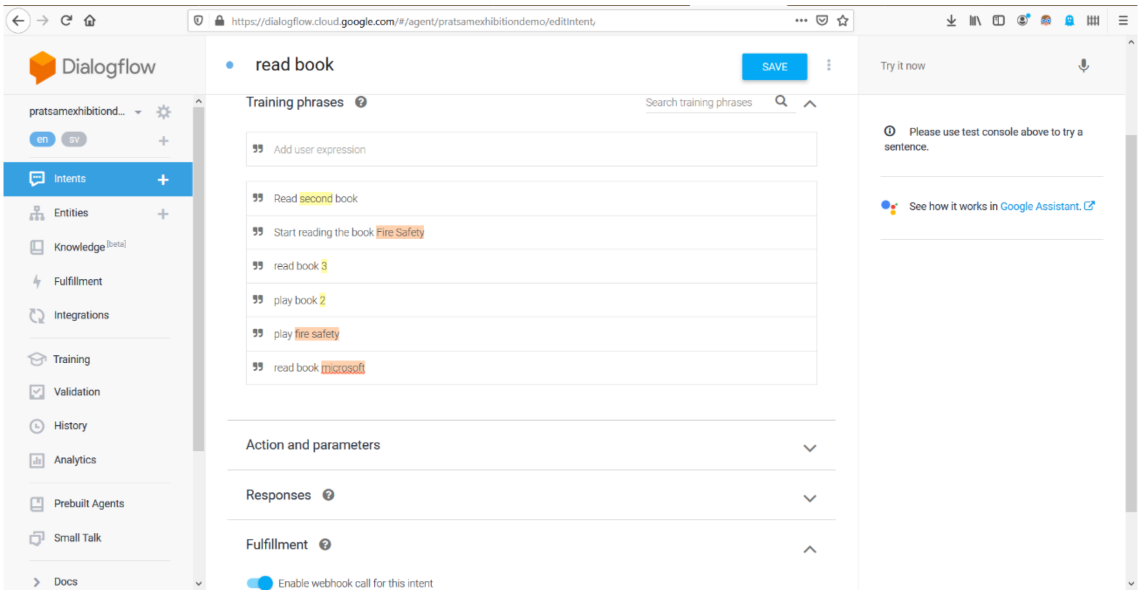
Picture 2. Action invocation name

Streaming audio from backend is supported in two different ways. SSML and Media object. SSML is meant for text-to-speech response such as answer to user. Media object is meant for playing larger audio files such as music, video or as in this case, a clip of an audiobook. Media object also supports chaining multiple audio files together, which is necessary for playing an audiobook (Google: Media response, 2021).

Daisy 2.02 audio files are in mp3 format, which is also the requirement for media object audio (Daisy: 2.02, 2001)- Audio files can be stored in anywhere as long as it is accessible through public internet. One simple way is to store the audio files in a cloud platform. Google has its own storage that can be used. The Google cloud storage is not preferable for terabytes of audio data due to the costs, but can be used for the proof-of-concept purposes.

An intent is required to be configured with Dialogflow. Intent is an action that the user tries to do. Dialoflow needs multiple example user phrases for the Natural Language Prosessing (NLP) AI to figure out what the user intent is. These phrases can be given in text as Actions on Google converts audio to text for Dialogflow to process. The phases for an intent such as starting the reading can be “Read the book”, “Read my book <book name>”. It is important to remember that users have multiple ways of saying the same intent. That is why it is preferable to include as many example phrases as possible for each intent. The Dialogflow NLP AI can figure out the user intent even if the phase is little bit different from the example phrases.

When an intent is triggered, an answer can be made. An intent can be configured to trigger webhook tied to developer made API. Webhook API can answer to the user with the with the required commands and parameters to start playing the media audio.



https://dialogflow.cloud.google.com/#/agent/pratsamexhibitiondemo/editIntent/

read book

Training phrases

Search training phrases

Add user expression

Read second book

Start reading the book Fire Safety

read book 3

play book 2

play fire safety

read book microsoft

Action and parameters

Responses

Fulfillment

Enable webhook call for this intent

Please use test console above to try a sentence.

See how it works in Google Assistant.

Picture 3. Intent Creation in Dialogflow

Pause and continue features are built in within all Google Assistants devices. Pause and continue cannot however be controlled by external API. As no notification, event or any response comes when user uses those commands (Google: Media response 2021).

Changing playback volume is not possible with voice commands. Feature is not available for Conversational Assistant Actions (Google: Media response 2021).

Change playback speed of an audio is not supported directly in the Google Assistant; however, it is possible to make the Assistant play another audio file that is speeded up duplicate of the normal speed audio. Changing the audio can be done by changing URL.

Example:

http://weburl.com/bookname/speed100/chapter1.mp3

→

http://weburl.com/bookname/speed200/chapter1.mp3

List and select contents from a virtual bookshelf feature can be implemented with use of a database. User can be identified after sign in process. It is possible to use user email or Google user id to define the owner of a virtual bookshelf. Bookshelf stores reference to user and holds only the ID of each owned book. Further implementation could have expiration dates and other logic that libraries usually have. User is required to have signed in before the bookshelf can be generated or accessed. So that the references can be set.

Jumping back and forward in the audio is not directly supported by Google Home device. Also, the Google Assistant does not support starting audio clip from anywhere else except from the beginning (Google: Media response 2021). This means going backwards or forward can only be as accurate as how short the audiobook clip lengths are in the audiobook. If user wants to go forwards for 10 seconds, and every audio clip within audiobook is length of 3 minutes. It simply is not possible to be accurate. However, if user wants to jump forward for 60 minutes, the user will probably be happy even if the jump is 60minutes $\pm$ 3 minutes.
Usually, Daisy 2.02 books are divided to the length of book chapters. The Daisy 2.02 audiobook standard, however, does not force any exact way of splitting book audio (Daisy: 2.02 2001). Meaning the length of the audio clips could be anything, even as big as only one audio file for the length of the whole book.

Going to the next and previous chapter with commands in example *"Hey Google, go to the next/previous chapter"* is possible if the audio clips are divided into the length of its chapters. Media response also supports pause, continue, start over and next commands (Google: Media response, 2021). Problem with these commands is that they are fully controlled by Google and do not send any response back to a webhook. these Google controlled commands cannot be overridden and will likely result in confusion for a user. In practice these commands make it impossible to control what happens when a user says next or previous.

For Authentication alternatives, Google provides a possibility to authenticate using voice. This does not mean that user could a create a new Google Account using voice, but instead user can give his current Google user details for the application to use (Google: Sign In, 2021). A user can be identified with user id, even when the user uses multiple devices with same Google account. The authentication makes it possible to always give the user its current bookmarks, lastmarks and bookshelf even when the device is changed. Authentication provides developers with full name, email address and user id. These details can be stored in database and can be used as identification every time user invokes the application.

Adding and selecting bookmarks have similar problems to the navigation in time -features. Bookmarks suffer from same limitations regarding accuracy in time. As an audio clip can only be started from the beginning and not from middle of the audio, it is impossible to have accurate bookmarks. Instead, bookmarks can be as accurate as length of the audio clips in each book. Another problem is that there is not a way for webhook API to receive anything about the current state of running audio. Meaning there is no way of knowing if the user is closer to start of end of the audio clip. Measuring the time server side is also impossible as Google Home device has built in Pause command for all media, that does not trigger any event for a webhook API. With these limitations in mind, it is possible to develop inaccurate but functional bookmarks. The book, user, and the time in the book can be inserted into a database.

Lastmark means the last position where user has left. This can be implemented as a bookmark. The only difference being that lastmark needs to be updated constantly and needs to be differentiated for normal bookmark in the database.

DODP v1 and v2 features mirroring the POC backend

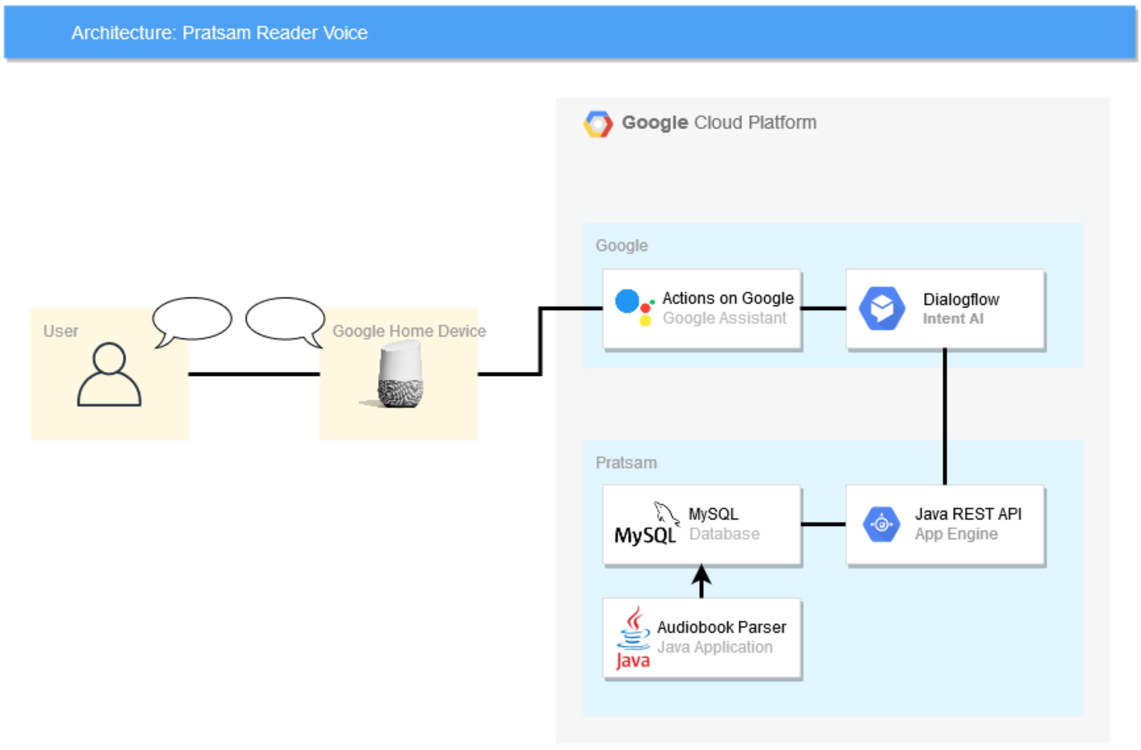
Daisy online protocol has bookmark, lastmark and navigation features build into it. The features for proof-of-concept should use similar structure so that they can be easily converted to use Daisy online protocol in the future.

Daisy 2.02 navigation model (levels, pages, percent, by time)
These Daisy navigation features cannot be implemented accurately due to same problem as with any navigation by time and bookmarks. Google assistant only supports starting each audio file from beginning.

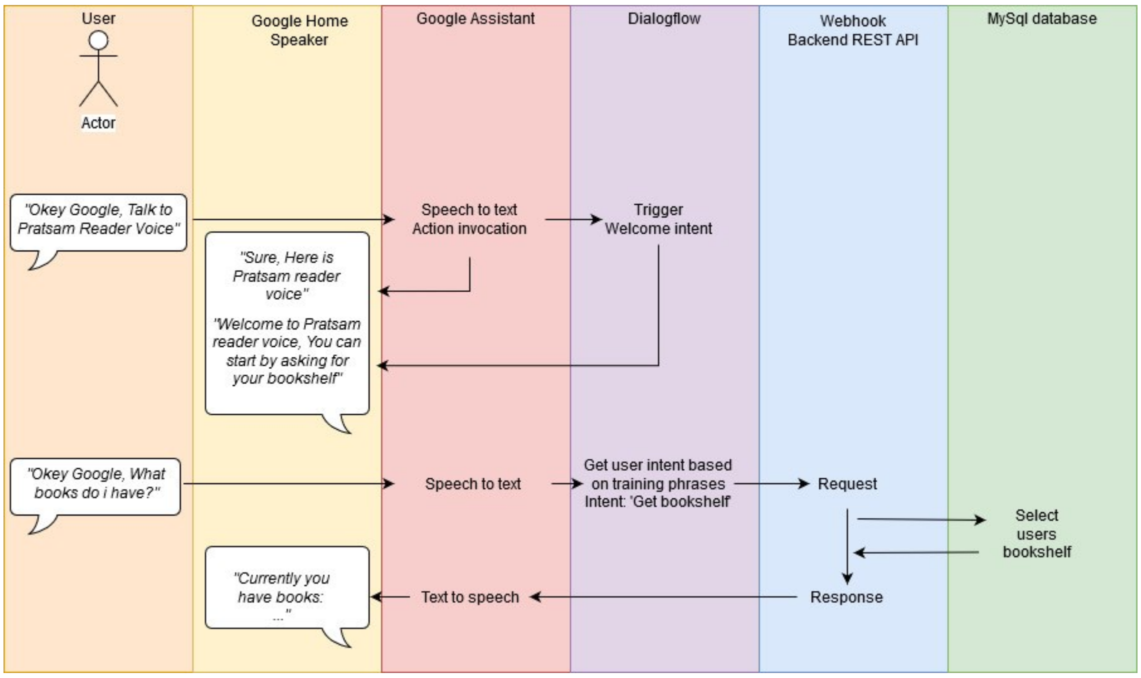
5.3. System architecture

The system architecture will be built using Google Cloud platform. Reasoning for this is its simplicity as all the guides Google provides have examples on how to implement it in the Google Cloud environment. Actions on Google platform controls the Google Home device and sends user speech as text to Dialogflow, which will use it to determine user intent. After user intent is known Dialogflow will trigger the REST API webhook. The REST API will decide the response and inserts the important parts into MySql database. Actions on Google converts the answer back to voice and sends the response to users Google Home device.

The audiobooks are inserted into the application with help of a parser. A Java application that goes through the audiobook files and sets references of audio files and navigational data to the database. The audio files of all audiobooks are stored into Google Cloud Storage.



Picture 4. System Architecture diagram


User
Actor
Google Home Speaker
Google Assistant
Dialogflow
Webhook Backend REST API
MySql database
"Okey Google, Talk to Pratsam Reader Voice"
Speech to text Action invocation
Trigger Welcome intent
"Sure, Here is Pratsam reader voice"
"Welcome to Pratsam reader voice, You can start by asking for your bookshelf"
"Okey Google, What books do i have?"
Speech to text
Get user intent based on training phrases Intent: 'Get bookshelf'
Request
Select users bookshelf
Response
Text to speech
"Currently you have books: ..."

Picture 5. System flow diagram

5.4. Webhook REST API

The REST API will be using Googles recommended boilerplate as the initial code base. This will provide simple empty project that can be used as base of the development. Google provides has Actions on Google SDK available in Java and NodeJs. Java was decided due to prior experience with the language.

Google App Engine will be used as the cloud platform to host this web service. App Engine supports automatic scaling by default by multiplying instances when one instance reaches full capacity. Scaling can be configured with limitations to instance amounts, but it is not necessary for this proof-of-concept. The instances can be configured with different memory and CPU limitations. By default, the App Engine uses slowest instance class which has 256 MB memory and 600MHz CPU limits. This is good enough for testing the functionality (Google: App Engine runtimes, 2021).

Gradle will be used for build automation. This includes handling package dependencies, building the project and deploying it into App Engine. Google has developed a Gradle plugin that helps setting App Engine configurations and to set up one line command to handle the whole deployment process.

Actions on Google provides two different types of storage to be used within the webhook: conversation storage and user storage. Conversation storage lasts only for the time of a single conversation. This storage should be used to keep track of anything that is needed for current conversation for example, what book has the user selected or what is the state of the book that is currently playing. Conversation storage can also be used to cache book data to keep database connections to minimum. The user storage persists across conversations and is only cleared if the user decides to delete the content. User storage can be used to save the identification of a user (Google: Save Data, 2021).

5.5. Daisy 2.02 audiobook parser

The parser is a Java console application that reads through the Daisy audiobook and collects all meaningful data to be inserted into database. Parser is not part of the proof-of-concept and is not required to poof anything. Instead, it is used as a simple way to insert new books into the database.

The relevant files that need to be parsed are ncc.html and multiple SMIL documents. NCC stands for navigation control center and is used to define how the book is structured. Ncc.html contains all the metadata of a book. Including book name, creator, language, length, and the narrator. The NCC file contains the order in which all the SMIL files are meant to be played. SMIL stands for Synchronized Multimedia Integration Language. The SMIL documents have text-audio synchronization functionality, it defines the segments of a book and most importantly it defines the order how the audio files are meant to be played (Daisy: 2.02 2001).

Parser is meant to be ran once every time a new book is available. After the proof-of-concept period, the parser is meant to be developed further and to be inserted into a server that will run it every time a new book is available. It does not need to be production ready for the proof-of-concept. But it should be built in a way that it could be developed into production ready parser and does not need to be remade.

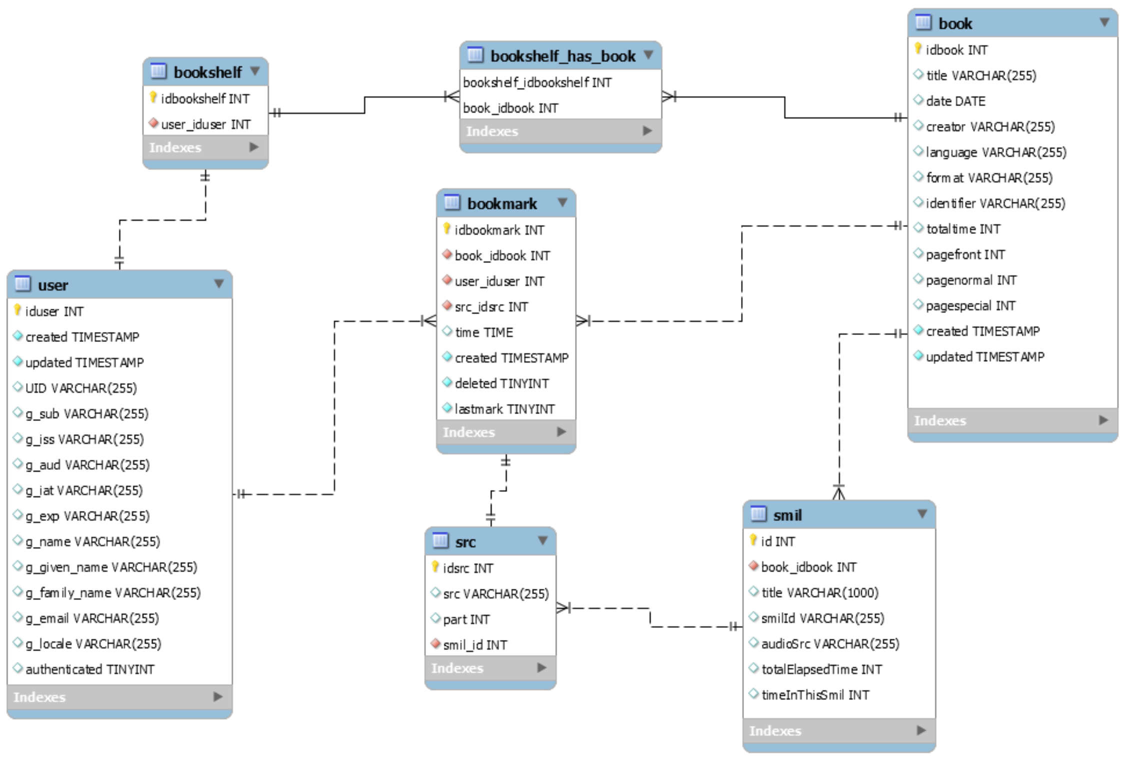
5.6. Google Cloud Storage

Audiobook files are saved into cloud storage. Each book has a single folder named after the title of the book. The storage is opened to public internet so that it can be easily accessed. While the data can be secured, it is not needed for proof-of-concept purposes. As all the audiobooks used in this proof-of-concept are test books that are already available in public internet (Google: Cloud Storage 2021).

5.7. Database

Google Cloud SQL has multiple machine type to be used. Smallest ones ‘db-f1-micro’ and ‘db-g1-small’ have shared CPU cores and standard ones go up from 1 CPU up to 96 CPUs and 0.6 ram up to 360GB or ram. The machine type can be changed any time. For proof-of-concept the smallest machine type with shared CPU and 0.6GB or ram should be enough as there are not any users. This may cause some delays in database queries (Google: SQL Pricing, 2021).

Database connection within Google Cloud is by default configured to automatically allow only connections from the same Google Cloud project. The database server location is set to Finland to keep latency small. The Cloud SQL has automatic backups that are ran daily. MySQL is open-source relational database by Oracle Corporation. “A relational database stores data in tables rather than putting all the data in one big storeroom.” (MySQL, 2021) The database is designed to mirror the Daisy format in books and contain all necessary tables needed for features.



Picture 6. Database schema

User table contains all information about users of the system. Authenticating to the system with google gives us name, email, locale and most importantly google user id (g_sub). These can be used to identify every user and give them their own unique bookshelf even if they are using multiple devices.

Bookshelf table is a simple collection of users' books. Users books can only be added and removed.

Book table contains everything descriptive about the book. Title, publish date, creator, language, page number and total length of the book. All information is collected from NCC.html file of a daisy book.

Smil table is collection of .smil file contents in a Daisy 2.02 book. Smil file is usually a chapter or smaller part of a book, but it can be as big as a single book. It contains reference to a mp3 audio file which contains voice reading of that part of the book. Smil

also contains timestamp of total time of book to the current point and the length of the current part in mp3.
Src table contains URL references to all mp3 audios in each smil file. Each smil can contain as multiple audio files.

Bookmark table contains references to user, src and the book. Bookmark can be user placed or lastmark. Lastmark is the last position that user was reading.

5.8. Voice user interface design

The voice user interface is designed to follow conversational principles. Conversational voice user interface tries to guide user through conversation as if user is in conversation with a human being (Cathy Pearl, 2016). This is different from traditional menu-based systems in a way that it has memory of the conversation so far, it does not have conversational dead ends and is building the conversation to be more precise as the conversation progresses (Michael H. Cohen, J. P. Giangola, J. Balogh, 2004). For example, after user has selected a book and decides to ask for current bookmarks, the user might change its mind in middle of bookmark process and instead wants to change the book completely. Voice user interface need to understand that the user intent has changed and not to lock the state into browsing bookmarks which can be considered as conversational error. The voice user interface is not trying to replicate text or button-based menus and instead is trying to behave as human conversation.

To make text to speech conversation be more human like Actions on Google has support for SSML, which stands for speech synthesis markup language. SSML has important configuration for text-to-speech conversation such as allowing speech to be segmented with pauses between sentences and paragraphs. Pause length can be varied based on the situation. This will be especially important when presenting a list of something for the user. Having longer pause between each item in a list makes understanding the list much easier. SSML also features for interpreting words, fractions, characters, dates, possibility to emphasize a word, but mostly the most important one will be the pausing between sentences.

```
Currently you have books: <pause>
Book number 1 is <small pause> Fire Safety by Blaxland, Wendy <pause>
Book number 2 is <small pause> xxx by xxx <pause>
...
```

Slot filling feature is needed for all selecting intents. Such intents are selecting a book and selecting bookmark. Slot filling means when used tries to select something, and the intent understood but the parameter is not. The system should ask for clarification and not lose the user. Simple question such as 'What was the bookmark?' is enough.

When the user invokes the application for the first time there will be more guidance. User is notified of the help section and given rough idea on how to proceed.
During the conversation it is important that user knows if it is expected to give a command. This can be solved by ending each reply with a question. for example, "What would like to do next?".

Voice user interface is developed to be state based. When user enters a new state, a new set of commands becomes available. The states are book selected, bookmarks asked and book playing. Commands that are always available are change speed, sign in, get bookshelf, select book and help.

5.8.1. Book selected state

In the book selected state the system needs to have a reference to the book the user has selected. The reference is given by user using the select book command. Conversational example of this would be:

User: *"Hey Google, Talk to Pratsam Reader Voice"*

Assistant: *"Welcome to Pratsam reader voice. You can start by asking for your book-shelf."*
User: *"Open my bookshelf"*
Assistant: *"Currently you have books: 1. Fire Safety by Wendy Blaxland, 2. Climbing the Highest mountain by Per Sennels. Which book would you like to read?"*
User: *"Select book 2"*
Assistant: *"You have selected the book Climbing the highest mountain by Per Sennels"*

After the book has been selected, the user can start reading the book, get more information of the book and browse bookmarks.

5.8.2. Bookmarks asked state

After selecting a book user can ask for the bookmarks of a book using get bookmarks command. Continuing the conversation from selecting a book example:

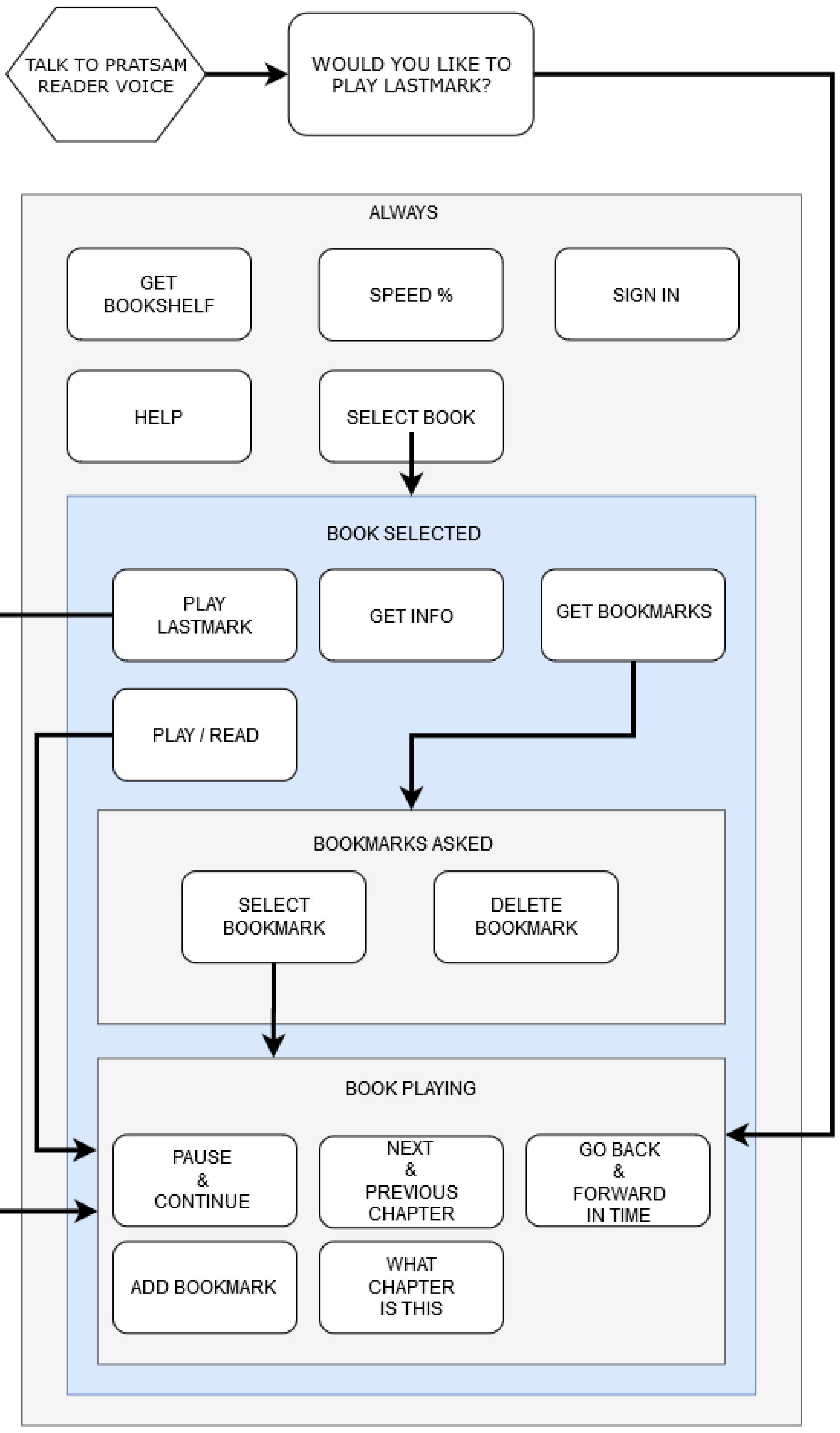
Assistant: *"You have selected the book Climbing the highest mountain by Per Sennels"*
User: *"What bookmarks do I have?"*
Assistant: *"You have one bookmark. Bookmark one title is Chapter one: Climbing the highest mountains".*

While in 'bookmarks asked' state user can delete and select the bookmarks. This state lasts for only until next voice command.

User: *"Select bookmark 1"*
Assistant: *"Reading you book Climbing the Highest Mountains from Chapter one, Climbing the highest mountains"*

5.8.3. Book playing state

Reading a book is the main feature of every audiobook reader. Minimizing the amount of commands user needs to give is the most important part of this voice user interface. User can continue reading from the lastmark of the last read book as the first command of a conversation. By Using the “Play lastmark” command. There are in total four ways to get into the reading book state as seen in the user interface diagram below. While the user is reading a book all the playback control commands are available, such as pausing, continuing, and moving in time. User can also add a bookmark and ask the current chapter that is playing at the time.



Picture 7. Voice user interface diagram

6. DEVELOPMENT

Development phase consists of implementing the designed Voice user interface to Dialogflow with all its required example phrases, programming the webhook Java REST API, implementing the database design, programming the audiobook parser, and connecting all the pieces together.

Development was done in long Agile development sprints, but since there is only one developer there was no need for strict Agile approach. The length of the sprints was on average two weeks. At the start of each sprint new tasks and new set of goals were given, where some tasks are on higher priority than others. At the end of each sprint all the results were discussed, and adjustment for the next sprint were made if any problems occurred.

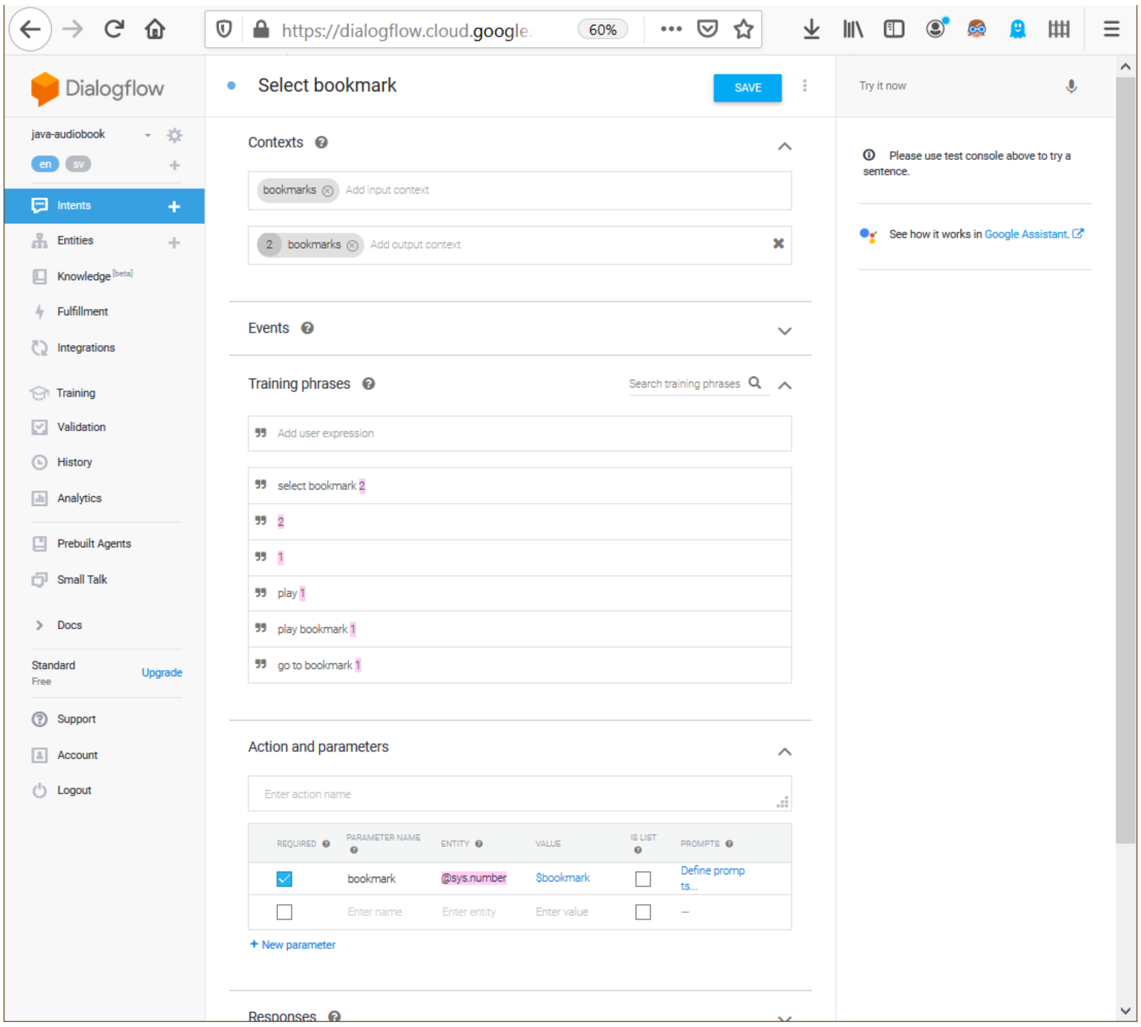
6.1. Voice User Interface (VUI) development

Voice user interface is mainly developed in Dialogflow. First step is configuring all intents which have eight configurable parameters. Intent name, Intent priority, Intent context, Intent event, Training phrases, phrase parameters, response, and fulfillment.

6.1.1. Dialogflow Intents

Intent name is used to keep track of what it does and used to trigger correct function call in the Webhook.
Intent Priority is value that controls how likely the intent is selected. This will be set as normal for most intents. If testing shows a conflict between two intents, this is one parameter that could help in resolving issues. For the most part all the training phrases should be enough to determine the intent.
Intent context is a way to remember parameter values between intents. Context can be used to keep track of state of the conversation. Such as reading state, bookmarks

asked state. Intent can have input and output context. Input context can be set if the intent requires a state such as reading state is required for a bookmark to be added.
Intent event is additional way to trigger an intent. Intents that use this are Welcome intent, media update intent and fallback intent.
Training phrases are the examples for the Dialogflow Natural Language Processing AI to determine the user intent. Each intent should contain at least 6 different example phrases. The AI can determine the user intent even if the user says something slightly differently from the training phrases as long as it is close enough and the training phrases are broad enough.
Phrase parameters are the important parameters user might give when phrasing a voice command. Needed parameters are such as numbers, words ‘yes’ and ‘no’, book names etc.
Response is the response back for the user. These fields can be left empty if the response is handled in the webhook. Some intents where there is only simple text response could use response field, but for sake of keeping everything consistent we will always use the webhook.
Fulfillment should always be set to enabled for webhook. This will enable the intent to be triggered in webhook API and response can be controlled.



Select bookmark

Contexts

bookmarks — Add input context

2 bookmarks — Add output context

Training phrases

- select bookmark 2
- 2
- 1
- play 1
- play bookmark 1
- go to bookmark 1

Action and parameters

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST	PROMPTS
✓	bookmark	@sys.number	$bookmark	☐	Define prompts...
☐	Enter name	Enter entity	Enter value	☐	–

Picture 8. Bookmark intent configured.

6.2. Webhook REST API

Webhook REST API will be written in Java. Google has built Action on Google Libraries for two programming languages. Java and Node.js are both fully supported and have as good documentation. Node.js is used more as it is faster to start the developing with. Java provides better debugging possibilities which might provide to save time in the future.

All the programming is done using Eclipse as the IDE. Eclipse is one of the most used IDEs when programming for Java. Version controlling is done using Git as that is what Pratsam has been using in the past.

6.2.1. Java Project structure

Google recommends using their project boilerplate to start the project from. The boilerplate is an empty project with all Actions on Google Java libraries in place. The boilerplate also has artifacts in place for quicker start with Google App Engine deployment. Choosing to use this boilerplate will save some time and is great starting point for a proof-of-concept project.

6.2.2. Handling Intent request and response

Dialogflow will generate the JSON request for the REST API after the user intent has been identified.

JSON Request format

```
{
  "responseId": "5ed78cb1-87ea-4762-92a1-2d37c94cce55-a14fa99c",
  "queryResult": {
    "queryText": "hello",
    "parameters": {},
    "allRequiredParamsPresent": true,
    "intent": {
      "name": "projects/pratsamexhibitiondemo/agent/intents/91a2d797-
7e16-4db1-9a28-19cdd8b2c729",
      "displayName": "Default Welcome Intent"
    },
    "intentDetectionConfidence": 1,
    "languageCode": "en"
  },
  "originalDetectIntentRequest": {
    "payload": {}
  },
  "session": "projects/pratsamexhibitiondemo/agent/sessions/xxxx-xxxx-
xxxx-xxxx-sensored"
```

Intents can be handled with help of the Actions on Google library. Methods need to be tagged as intent handlers with special annotation @ForIntent("<Intent name>"). The method needs to take ActionRequest as a parameter and return ActionResponse. ActionRequest parses the JSON request body and converts it into an object. ActionResponse will be generated into the JSON response. ResponseBuilder is a helper containing helpful tools for response generation.

```
@ForIntent("Default Welcome Intent")
 public ActionResponse Welcome(ActionRequest request) {
   ResponseBuilder responseBuilder = getResponseBuilder(request);
   responseBuilder.add("Welcome to Pratsam Reader Voice, you can start
                         by asking for your bookshelf");
   return responseBuilder.build();
 }
```

This response is converted into following JSON response.

```
{
  "fulfillmentText": "Welcome to Pratsam Reader Voice. You can start
by asking for your bookshelf.",
  "outputContexts": [
    {
      "lifespanCount": 99,
      "name": "projects/pratsamexhibitiondemo/agent/sessions/f998f9a9-
daef-00d6-9fe2-3803b3b74bee/contexts/_actions_on_google",
      "parameters": {
        "data": "{}"
      }
    },
    {
      "lifespanCount": 0,
      "name": "projects/pratsamexhibitiondemo/agent/sessions/f998f9a9-
daef-00d6-9fe2-3803b3b74bee/contexts/DefaultWelcomeIntent-followup"
    }
  ],
  "payload": {
    "google": {
      "expectUserResponse": true,
      "richResponse": {
        "items": [
          {
            "simpleResponse": {
              "textToSpeech": "Welcome to Pratsam Reader Voice demo.
You can start by asking for your bookshelf."
            }
          }
        ]
      },
      "isSsml": false,
      "systemIntent": {
        "intent": "actions.intent.TEXT",
        "data": {}
      },
      "userStorage": "{\"data\":{}}"
    }
  }
}
```

Data of users can be saved between conversations in user storage or for the duration of single conversation with conversation storage. User storage persists between conversations and especially useful when saving Google User Id (Google: Save Data 2021). The user id can be used to get user bookmarks and bookshelf from a database. Conversational storage can be used for data that is required for duration of single conversation, such as current state, selected book, current chapter the user is reading and URLs to audiobook mp3 files. Conversation storage is faster than getting data from a MySQL database.

```
// Example of how to add data to conversation and user storage
Map<String, Object> convData = request.getConversationData();
Map<String, Object> userStorage = request.getUserStorage();
convData.put("book", book.getTitle()); //lasts for converation
userStorage.put("book", book.getTitle()); //lasts forever
```

6.2.3. User interface states

The backend needs to keep track of the state the user is in. The states are book selected, reading and bookmark asked state. Book selected state is active if the title of a book is in conversational storage. Reading state is active if chapter number of a book is in conversational storage. Bookmarks asked state is small enough so that it can be handled with Dialogflow context and does not need a state in the backend.

```
Map<String, Object> convData = request.getConversationData();
// book selected
if (convData.containsKey("book")
...

//reading has been started
if (convData.containsKey("book") && convData.containsKey("chapter"))
...
```

6.2.4. Account linking

Sign in process requires user permission to access Google account information. The permission can be asked using SignIn method in the Actions on Google Java library.

```
@ForIntent("Start Signin")
public ActionResponse startSignIn(ActionRequest request){
      ResponseBuilder rb = getResponseBuilder(request);
      return rb.add(new SignIn()
                  .setContext("To get your account details"))
                  .build();
}
```

After user has given permission to access account information it can be accessed from the request. The access the user information, the user profile contained in request needs to be decoded with Google Cloud client ID. The client ID is a Google generated string that is used to identify the service. Once the profile has been decoded, the backend can read user data, give user the correct bookshelf and bookmarks. The user can use multiple devices to access the same bookshelf and bookmarks if the same Google account is used (Google: Sign In 2021).

```
private GoogleIdToken.Payload decodeIdToken(String idTokenString)
      throws GeneralSecurityException, IOException
{
      HttpTransport transport = GoogleNetHttpTransport
                                          .newTrustedTransport();
      JacksonFactory jsonFactory = JacksonFactory
                                          .getDefaultInstance();
      List<String> audience = new ArrayList<String>();
      audience.add(clientId);
      GoogleIdTokenVerifier verifier =
            new GoogleIdTokenVerifier
                  .Builder(transport, jsonFactory)
                  .setAudience(audience).build();
      GoogleIdToken idToken = verifier.verify(idTokenString);
      return idToken.getPayload();
}
```

6.2.5. Playing a book

Audio can be started with a media response. The media response takes content URL as parameter as well as optional visual elements picture and description, for users that are using a device that has screen on it. The Google Home device sends an event back to the backend once the audio clip has finished. This allows the backend to respond with another media response, making it possible to chain multiple audio files together and to play the full book from start to finish. Since the audio is sent as URL the audio file needs to be accessible from public internet without restrictions. The URL can however contain a token or any other methodology that limits the file access to a timeframe and to single user.

Google Home device has two big problems in functionality. First it does not send any notification if the user interrupts the audio with another command, such as pause, add bookmark or anything. And second it does not allow playing the audio files from any other point other than from beginning. This will result in user confusion as books cannot be navigated or bookmarked precisely (Google: Media response 2021).

```
//starting to play an audio file
public ActionResponse mediaResponse(ActionRequest request) {
      ResponseBuilder responseBuilder = getResponseBuilder(request);
      Map<String, Object> convData = request.getConversationData();

      MediaResponse mediaresp = new MediaResponse();
      MediaObject obj = new MediaObject();
      obj.setContentUrl("https://storageUrl/bookName/audiofile.mp3");
      List<MediaObject> mediaObjects = new ArrayList<>();
      mediaObjects.add(obj);
      mediaresp.setMediaObjects(mediaObjects);
      mediaresp.setMediaType("AUDIO");
      responseBuilder.add(mediaresp);

      //updating the conversational storage with current chapter id
      convData.put("current_chapter", {current chapter id});

      return responseBuilder.build();
}

//status update triggers when an audio playing has finished
@ForIntent("media.status.update")
public ActionResponse mediaStatusUpdate(ActionRequest request) {
      ResponseBuilder responseBuilder = getResponseBuilder(request);
      String mediaStatus = request.getMediaStatus();
      if (mediaStatus != null && mediaStatus.equals("FINISHED")) {
            //Getting the last played audiofile
            String lastchapter = convData.get("current_chapter");

            //creating new media response with next chapter
            ...
            //updating the conversational storage again
            convData.put("current_chapter", {new chapter id});
            responseBuilder.add(mediaresp); //the next chapter
      }
      return responseBuilder.build();
}
```

6.2.6. Changing speed

Google Home device does not have built in functionality to change audio speed. The speed change however can be done manually, by having multiple files with different speeds. In this proof-of-concept we are manually speeding up only one book as it takes a lot of time for each book. In a full product the generating of speeded up files should be done automatically as part of parsing process.

Changing the audio playing speed is done by changing the URL to the same file that has been modified to be faster. The speeded-up audio files are stored in folders that are named after the speeds. Folder 'speed120' contains audio that has been speed to 120% and so on. The book was speeded up to speeds 100%, 120%, 140%, 170% and 200%.

User can change the speed by saying command 'Change speed to 200%'. The percentage is saved into conversional storage and accessed after the user starts playing an audio book. This results in following example URLs for 100%, 120% or 200% speed.

https://storageurl.com/bookname/audiofile.mp3
https://storageurl.com/bookname/speed120/audiofile.mp3
https://storageurl.com/bookname/speed200/audiofile.mp3

```
//setting the speed
@ForIntent("Set speed")
public ActionResponse SetSpeed(ActionRequest request) {
      ResponseBuilder responseBuilder = getResponseBuilder(request);
      String speed = request.getParameter("percentage").toString()
      Map<String, Object> convData = request.getConversationData();
      convData.put("speed", speed);
      responseBuilder.add("The speed was set to " + speed +" %");
      return responseBuilder.build();
}
```

```
//when playing the book
...
String speed = convData.get("speed");
bookURL += "/speed" + speed;
...
```

6.2.7. Jumping back and forward in the audio

Navigating the book is precisely has the same problems as bookmarks have. Audio clips can only be started from the beginning. Usually, the audiobooks are divided so that each chapter has its own file. This makes navigating between chapters possible. Users highly likely want a feature where they can go back short amount of time in case, they missed something and want to listen it again.

In addition to the problems with navigation accuracy, Actions on Google has commands next and previous locked as system intents and those cannot be changed. 'Next' command works the same way as when audio clip ends. 'Previous' command for goes to the beginning of audio clip making it really confusing to the users. New intent can be created that uses every way of saying 'previous' that Google has missed. For example, if user says 'go back in time' or 'last chapter' etc. It is not good idea to create intents that in users' point of view works sometimes and sometimes not. But as proof-of-concept it can be implemented, and taken to use later, if Google eventually allows system intents to be overridden. Similarly, the book can be navigated by jumping multiple chapters with same command.

6.2.8. Bookshelf

Bookshelves are collection of books that user currently has. The information of the available books is stored in a database. A user is identified with its Google Id after the sign in process and given access to the bookshelf. The Google id is saved to user storage which persists between multiple conversations.

```
@ForIntent("Bookshelf")
public ActionResponse Bookshelf(ActionRequest request) {
      ResponseBuilder responseBuilder = getResponseBuilder(request);
      Map<String, Object> userStorage = request.getUserStorage();
      String subject = userStorage.get("subject").toString();//user id
      Bookshelf bookshelf = null;
      try {
            bookshelf = Database.getBookshelf(subject);
      }
      ...
      List<Book> books = bookshelf.getBooks();
      responseBuilder.add( "currently you have books "
                        + {list of books});
      return responseBuilder.build();
}
```

6.2.9. Bookmark and lastmark

Bookmarks can be added while reading a book by ‘add bookmark’ intent, bookmarks can be browsed with ‘get bookmarks’ intent, deleted with ‘delete bookmark’ intent and bookmark can be selected to start playing the book with ‘select bookmark’ intent. Both bookmark and lastmark are saved in bookmark database table. Bookmark requires book id, location in a book, user id, and if it is a lastmark or not. Lastmarks are updated in every possible event to have the most precise lastmark possible. User can play the lastmark when the application opens or after selecting a book. The system will prompt user with “would you like to continue from where you left” and simple “yes” will start playing the book from the last known position. Playing the book from bookmark location is the same as playing book from beginning with exception of starting location, which is taken from database.

The words ‘bookmark’, ‘bookshelf’, and ‘book’ are all similar and sometimes the speech recognition mistakes one to another. To minimize critical conversational errors, the delete bookmark and the select bookmark intents are only available after user has asked for a list of bookmarks. Get bookmarks intent will save the bookmarks in conversational storage in same order as they were presented to the user. User can select or delete a bookmark by referring to the number in a list.

Example: Selecting bookmark
Google Home: *“Currently you have 3 bookmarks. Bookmark number 1 chapter title is ‘x’, bookmark number 2 chapter title is ‘y’, bookmark number 3 chapter title is ‘z’”*
User: *“Select bookmark 1”*
Google Home: *Playing the book from chapter x <book starts playing>*

```
//Getting list of bookmarks from a book
List<Bookmark> bookmarks = Database.getBookmarks(userId, bookId);
convData.put(“bookmarks”, bookmarks)
```

```
@ForIntent("Select bookmark")
public ActionResponse SelectBookmark(ActionRequest request) {
      ResponseBuilder responseBuilder = getResponseBuilder(request);
      Map<String, Object> convData = request.getConversationData();
      int bmParameter = request.getParameter(“bookmarkSelection”);
      List<Bookmark> bookmarks = (ArrayList<Bookmark>) con-
vData.get(“bookmarks”);
      Bookmark selectedBookmark = bookmarks.get(bmParameter - 1);
      ... //start playing the book from bookmark location.
}
```

6.2.10. Database connection

Connecting to the database can take multiple seconds and this would make the conversation slow. Database connection can be pooled to keep the connection open for multiple queries (Jesper Wisborg Krogh, 2018). “Connection pool is a technique of creating and managing a pool of connections that are ready for use by any thread that needs them” (MySql, 2021b). The database pool can be saved to servlet context.

Connection between Google Cloud App Engine and Google Cloud MySQL database is open by default if they are within the same Google Cloud project. Google provides required configurations and libraries that go with them.

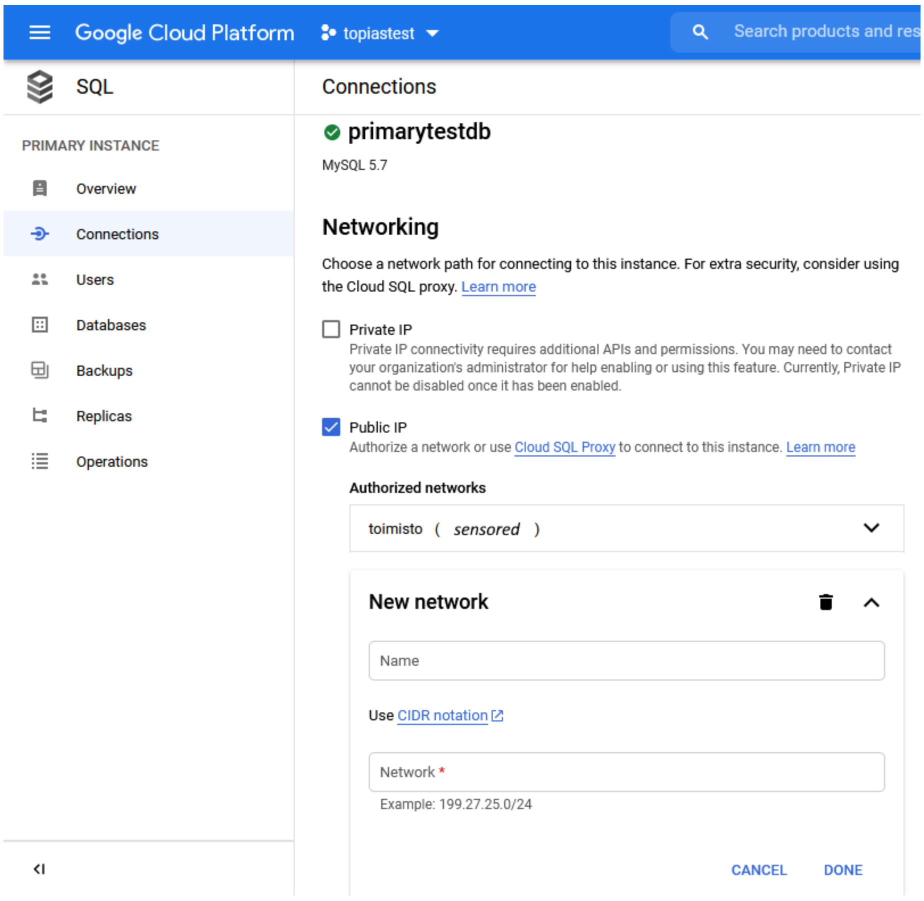
```
private HikariDataSource createConnectionPool() {
      HikariConfig config = new HikariConfig();
      config.setJdbcUrl(String.format("jdbc:mysql:///%s", DB_NAME));
      config.setUsername(DB_USER);
      config.setPassword(DB_PASS);

      config.addDataSourceProperty("socketFactory",
      "com.google.cloud.sql.mysql.SocketFactory");
      config.addDataSourceProperty("cloudSqlInstance",CLOUD_SQL_NAME);

      HikariDataSource pool = new HikariDataSource(config);
      return pool;
}

@Override
public void contextInitialized(ServletContextEvent event) {
      HikariDataSource pool = (HikariDataSource) event.getServletCon-
text().getAttribute("my-pool");
      if (pool == null) {
            pool = createConnectionPool();
            event.getServletContext().setAttribute("my-pool", pool);
      }
}
```

6.2.11. Deploying to Google Cloud

Deploying the Java project to the Google Cloud App Engine is done with use of gcloud command line tool. This is installed with Google Cloud SDK. The gcloud tool can be used in Windows command prompt with text commands. The most important commands are 'gcloud init' for initializing cloud deployment to the correct Google Cloud project and 'gcloud app deploy' for deploying the project. The 'gcloud init' command includes logging into Google Cloud with command prompt and selecting the Google Cloud project. The command 'gcloud app deploy' is used for deploying the project. Google recommends using Maven or Gradle to handle the deployment. The boilerplate includes Gradle ready configurations for the deployment of the App Engine. This can be done with single command 'gradle appengineDeploy' after the the gcloud tool has been initiated. The default configurations have 'stopPreviousVersion' and 'promote' properties set to true. These shut down the previous version of the App Engine instance and sets the newly deployed instance to be the new main version that will take all the traffic.

6.3. Database

MySQL database was using the Google Cloud SQL. The biggest benefits of using the Google Cloud SQL are that it has immediate access to App Engine within the same cloud project. Setting it up requires instance name, region, zone, and database versions to be selected. Region needs to be set to the region closest the users to minimize latency. In this case it was europe-north1 which is the only one located in Finland. The machine type was set to the smallest one which is db-f1-micro, which has one shared CPU and 614.4MB of memory. Google Cloud SQL has great tools to monitor Database usage to determine if better machine type is needed, and the type can be changed at any point. Connecting to the database to deploy the designed schema requires authorizing the public IP of the device that is used to connect.



Connections

primarytestdb

MySQL 5.7

Networking

Choose a network path for connecting to this instance. For extra security, consider using the Cloud SQL proxy. Learn more

☐ Private IP
Private IP connectivity requires additional APIs and permissions. You may need to contact your organization's administrator for help enabling or using this feature. Currently, Private IP cannot be disabled once it has been enabled.

☑ Public IP
Authorize a network or use Cloud SQL Proxy to connect to this instance. Learn more

Authorized networks

toimisto (*sensored*)

New network

Name

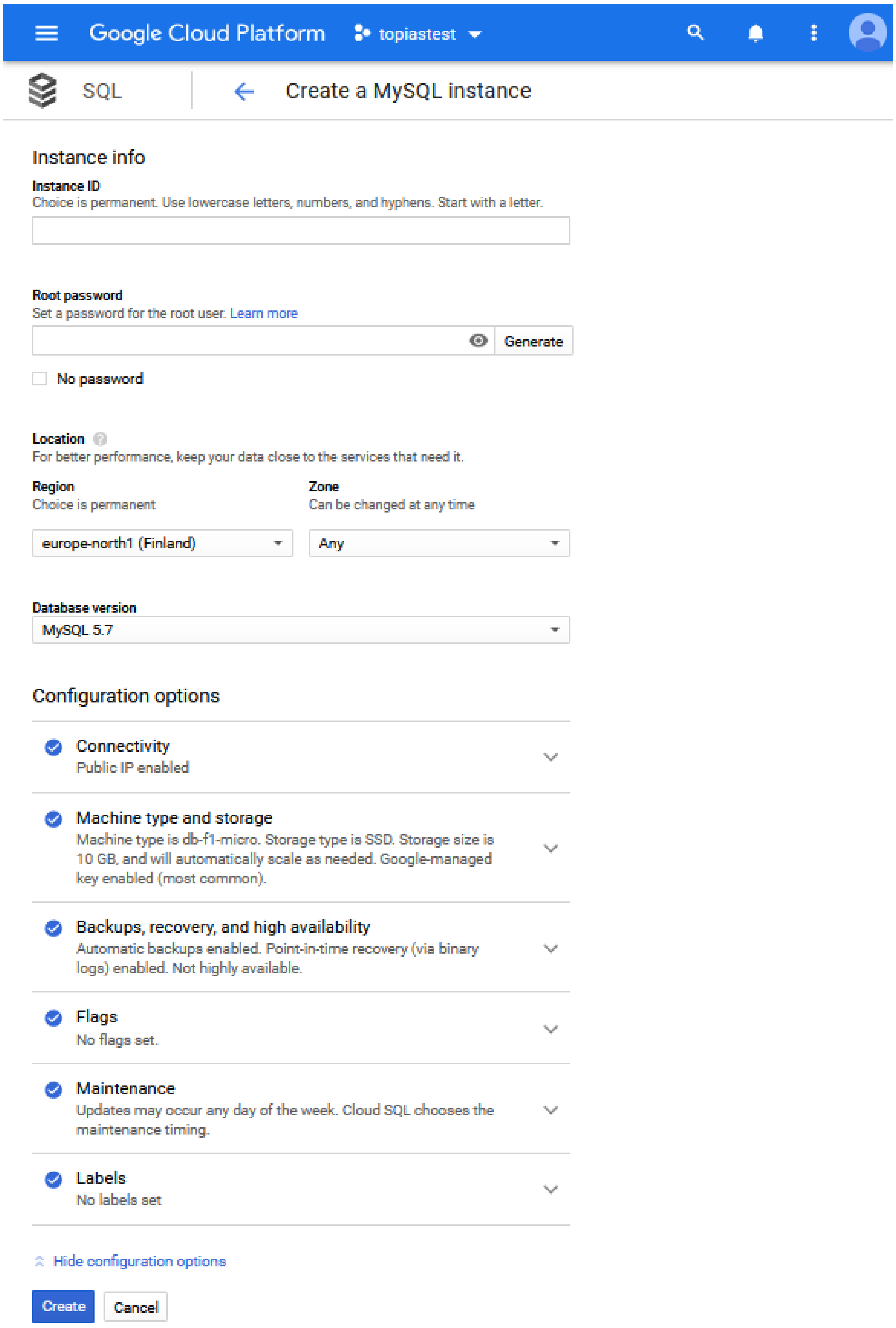
Use CIDR notation

Network *

Example: 199.27.25.0/24

CANCEL DONE

Picture 9. opening database to a single IP address


Google Cloud Platform
topiastest
SQL
Create a MySQL instance

Instance info

Instance ID
Choice is permanent. Use lowercase letters, numbers, and hyphens. Start with a letter.

Root password
Set a password for the root user. Learn more

Generate

No password

Location
For better performance, keep your data close to the services that need it.

Region
Choice is permanent
europe-north1 (Finland)

Zone
Can be changed at any time
Any

Database version
MySQL 5.7

Configuration options

Connectivity
Public IP enabled

Machine type and storage
Machine type is db-f1-micro. Storage type is SSD. Storage size is 10 GB, and will automatically scale as needed. Google-managed key enabled (most common).

Backups, recovery, and high availability
Automatic backups enabled. Point-in-time recovery (via binary logs) enabled. Not highly available.

Flags
No flags set.

Maintenance
Updates may occur any day of the week. Cloud SQL chooses the maintenance timing.

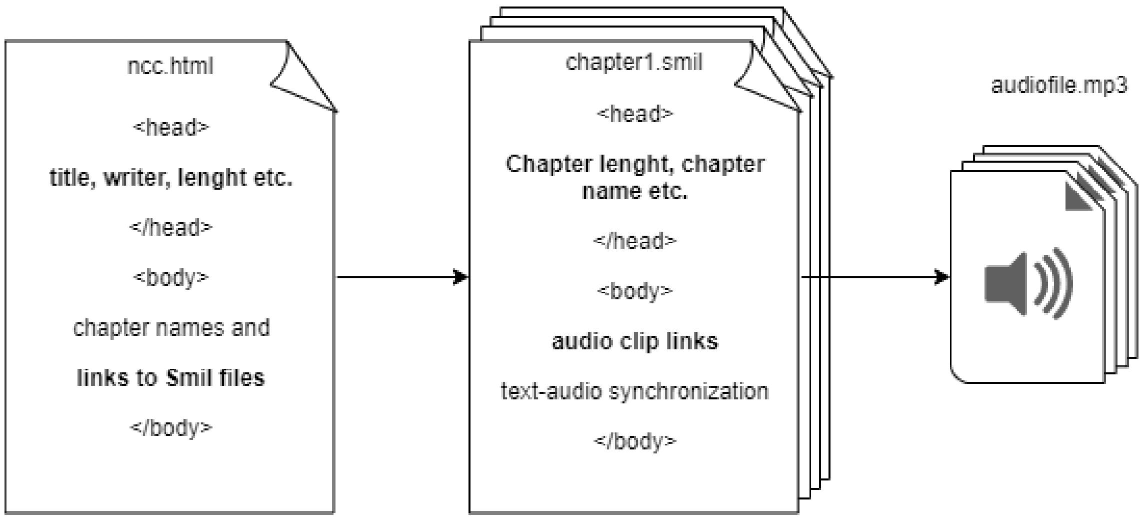
Labels
No labels set

Picture 10. Setting up a database in Google Cloud platform

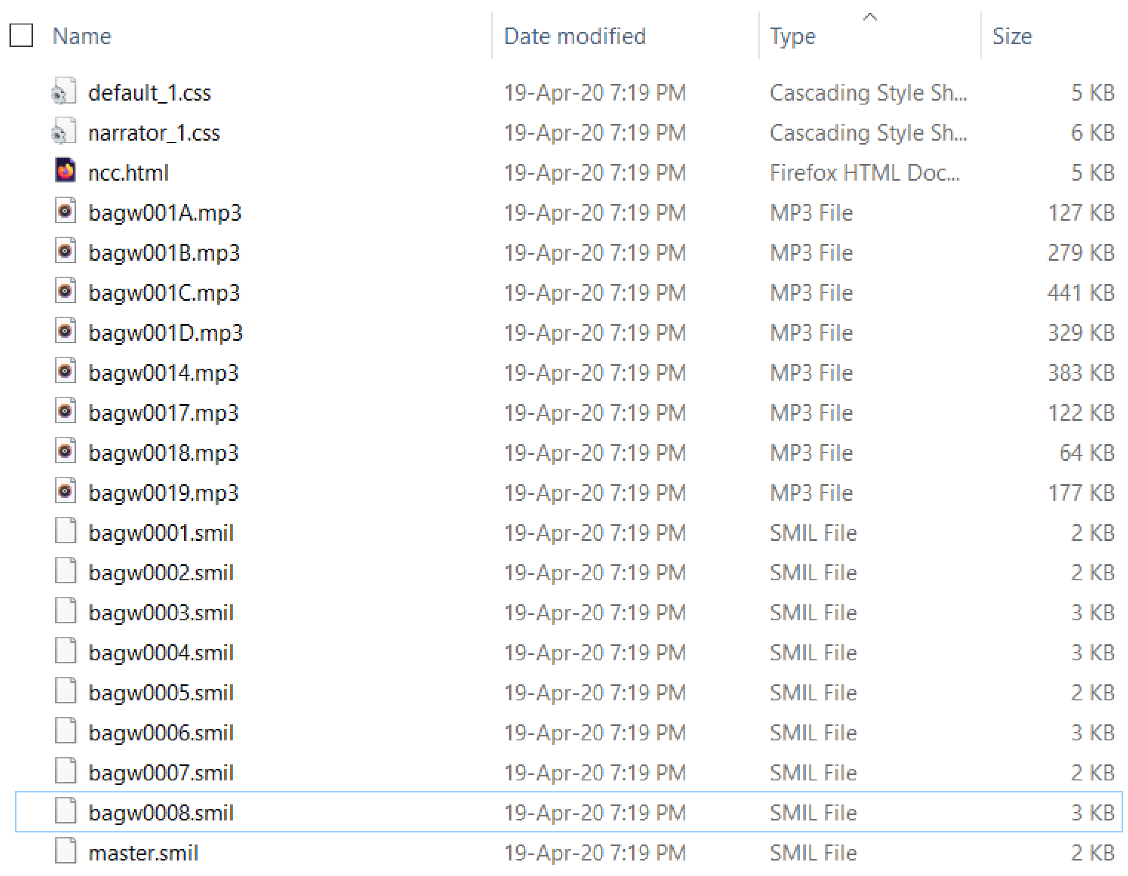
6.4. Daisy 2.02 audiobook Parser

Audiobook parser is a simple program made to transfer data from Daisy 2.02 audiobooks into a database. While in proof-of-concept period of the project, the parser can require some manual labor to upload each book. Parser itself does not need to prove anything as everything required related to reading data from audiobooks in Daisy 2.02 format have been done in the past by Pratsam. Daisy 2.02 is using standard XHTML 1.0 for the audiobook formatting (Daisy: 2.02 2001).

The data we are interested in are the book title, writer, publish date, names of the mp3 files and the length of the book in pages and in time. Most of the data can be parsed from a single file named NCC.html. NCC stands for Navigation Control Center and contains all the book meta information and has links to SMIL documents. SMIL stands for Synchronized Multimedia Integration Language. The main point of the SMIL files is to provide the text-audio synchronization. Since we are playing the audiobook from a speaker only, we do not need the synchronization data. Instead, only the names of all mp3 files and chapter data are parsed and inserted into the database.



Picture 11. Daisy 2.02 book structure



Name	Date modified	Type	Size
default_1.css	19-Apr-20 7:19 PM	Cascading Style Sh...	5 KB
narrator_1.css	19-Apr-20 7:19 PM	Cascading Style Sh...	6 KB
ncc.html	19-Apr-20 7:19 PM	Firefox HTML Doc...	5 KB
bagw001A.mp3	19-Apr-20 7:19 PM	MP3 File	127 KB
bagw001B.mp3	19-Apr-20 7:19 PM	MP3 File	279 KB
bagw001C.mp3	19-Apr-20 7:19 PM	MP3 File	441 KB
bagw001D.mp3	19-Apr-20 7:19 PM	MP3 File	329 KB
bagw0014.mp3	19-Apr-20 7:19 PM	MP3 File	383 KB
bagw0017.mp3	19-Apr-20 7:19 PM	MP3 File	122 KB
bagw0018.mp3	19-Apr-20 7:19 PM	MP3 File	64 KB
bagw0019.mp3	19-Apr-20 7:19 PM	MP3 File	177 KB
bagw0001.smil	19-Apr-20 7:19 PM	SMIL File	2 KB
bagw0002.smil	19-Apr-20 7:19 PM	SMIL File	2 KB
bagw0003.smil	19-Apr-20 7:19 PM	SMIL File	3 KB
bagw0004.smil	19-Apr-20 7:19 PM	SMIL File	3 KB
bagw0005.smil	19-Apr-20 7:19 PM	SMIL File	2 KB
bagw0006.smil	19-Apr-20 7:19 PM	SMIL File	3 KB
bagw0007.smil	19-Apr-20 7:19 PM	SMIL File	2 KB
bagw0008.smil	19-Apr-20 7:19 PM	SMIL File	3 KB
master.smil	19-Apr-20 7:19 PM	SMIL File	2 KB

Picture 12. Daisy 2.02 folder of the example book Climbing the Highest Mountain

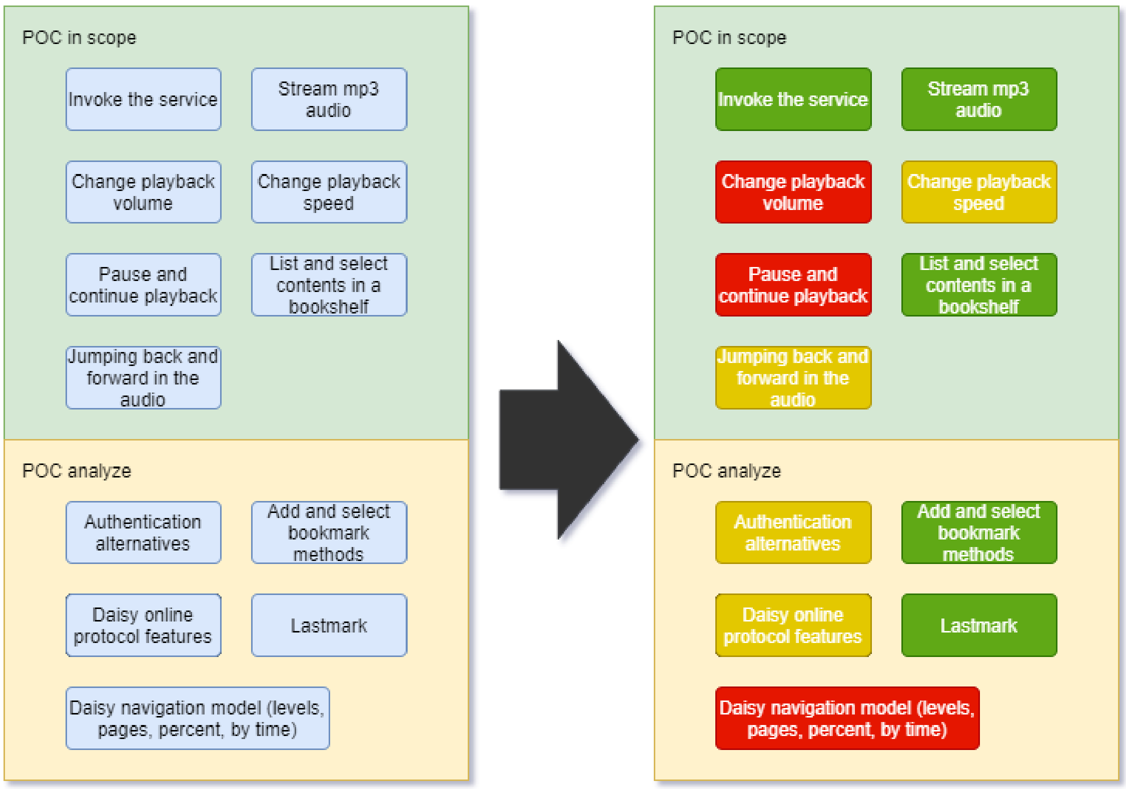
Parsing the SMIL and NCC files was done using Jsoup which is a Java library that has easy to use html parsing and works well with xHTML style documents such as SMIL. Parsing NCC and SMIL files work the same with exception of different data and the number of files. Parsed files are constructed into a Book object that will be inserted into the database which makes them available on users’ bookshelves.

```
// parsing NCC file for metadata
File nccFile = new File(folder_src + "ncc.html");
Document doc = Jsoup.parse(nccFile, "UTF-8");
Elements metatags = doc.getElementsByTag("meta");
newBook = new Book();
for (Element meta : metatags) {
      String name = meta.attr("name");
      String content = meta.attr("content");
      switch (name)
```

```
        {
              case "dc:title":
                    newBook.setTitle(content);
                    break;
              case "dc:creator":
                    newBook.setCreator(content);
                    break;
              ... // and same for rest of the data
        }
db.putBook(dbconn, newBook); // uploading the book into database
```

7. RESULTS

The result of the development is a completely working application that is able do most of the requested features. The picture 13 has the requested features on left and the end result on right. The features marked with green were fully developed without problems. Yellow features had some issues but are somewhat working. Red features were not developed.


POC in scope
Invoke the service
Stream mp3 audio
Change playback volume
Change playback speed
Pause and continue playback
List and select contents in a bookshelf
Jumping back and forward in the audio
POC analyze
Authentication alternatives
Add and select bookmark methods
Daisy online protocol features
Lastmark
Daisy navigation model (levels, pages, percent, by time)

Picture 13. proof-of-concept result

7.1. Proof of concept feature requirement analysis

Invoking the service works without issues. The solution can be accessed with Google Home Devices with key words “Ok Google, Talk to Pratsam Reader Voice”.

Streaming mp3 audio can be used for any Daisy 2.02 audiobooks.

Changing the Playback volume is not supported by Google yet, but since it is the most requested feature by developers, it may come later if Google decides to develop it.
Change playback speed feature is possible but required huge amount of storage space since each audio file is duplicated for every speed. This will likely be developed by Google into Action on Google media player as it already works on Google Play.
Pause and continue playback is working by default but the feature cannot be controlled by developers. It is completely closed and will result in huge confusion.
List and select contents in a bookshelf feature is fully supported.
Jumping back and forward in the audio was partly done. Navigating by exact amount of time is impossible but moving between chapters is possible. There are problems with conflicting system intents that makes using navigation extremely confusing for users.
Authentication alternatives were developed using built in Google Sign in. This will require more research in case it needs to be connected to a library service.
Adding and selecting bookmarks was found to be possible during the research phase and it was implemented to the system.
Lastmark feature was implement together with bookmarks.
Daisy online protocol features task meant that the backend system should mirror the functionality of Daisy online protocol so that it could eventually be connected without need for big changes. This task was mostly forgotten during the development, but in the end the resulting functionalities are not too far off.
Daisy navigation model could not be implemented due to restrictions in Actions on Google.

7.2. Googles Assistant downtimes, system errors and bugs

During the development Google pushed multiple updates to their Google Home devices. Some of these updates resulted in critical bugs that delayed the development process and broke main Actions on Google features even on live product. The downtimes lasted from weeks to months. The longest lasting bug was never fixed even with full effort from Pratsam to rise its importance. Pratsam sent regular email back and forth with Google support who only responded with copy paste replies. Google never gave time estimations or even progress report of the problems. Google never sent developers any information about new bugs which resulted in weeks of wasted time in debugging.

8. CONCLUSION

The proof-of-concept period did not prove that the concept is fully viable yet. There are required features that cannot be implemented, and Google has not provided any lists of upcoming features, improvements, or estimations on bug fixes. However, most of the necessary features such as volume control, speed changing and navigation are already working on YouTube music and Google Play Books, which are Googles own music and audiobook platforms. These features could be on the way for other developers. I can only give my recommendation on continuing into full product with the premise that Google would eventually extend the features on their own platforms for everyone to use.

Four months after completing the proof-of-concept period Google surprisingly released completely new version of Actions on Google which made navigation, volume, changing speed, and accurate bookmark features possible. Some problems are still present such as system intents back and forward are still confusing the users. The new version however had no Java SDK support and the whole backend had to be remade. Google did not inform about the decision to drop Java support beforehand, and they are not answering to the question if there will be a Java version.

Couple months after releasing new Action on Google version Google made an update to the old version which broke the bookmark functionality. This and some other critical bugs were never fixed but fortunately Pratsam was fast to change to the new version.

LIST OF REFERENCES

Cathy Pearl (2016) *Designing voice user interfaces.*

Pages 16-17, ISBN: 978-1-491-95541-3

Celia (2019) Welcome to Celia

https://www.celia.fi/eng/

Chelsea Myers, Anushay Furqan, Jessica Nebolsky, Karina Caro, and Jichen Zhu. (2018). Patterns for How Users Overcome Obstacles in Voice User Interfaces. Paper presented at 2018 CHI Conference on Human Factors in Computing Systems (CHI '18), Association for Computing Machinery, April 21–26, 2018 New York

https://doi.org/10.1145/3173574.3173580

Daisy: 2.02 (2001) Daisy 2.02 Specification

http://www.daisy.org/z3986/specifications/daisy_202.html

Daisy: 3 (2005) Specification for the digital talking book

https://daisy.org/activities/standards/daisy/daisy-3/z39-86-2005-r2012-specifications-for-the-digital-talking-book/

Google: App Engine (2021) App Engine

https://cloud.google.com/appengine

Google: App Engine runtimes (2021) The App Engine Standard Environment

https://cloud.google.com/appengine/docs/standard#first-gen-runtimes

Google: Assistant (2021) Integrate with Google Assistant

https://developers.google.com/assistant

Google: Cloud (2021) Accelerate your transformation with Google Cloud

https://cloud.google.com/

Google: Cloud Storage (2021) Cloud Storage

https://cloud.google.com/storage

Google: Cloud SQL (2021) Cloud SQL

https://cloud.google.com/sql

Google: Dialogflow (2021) Dialogflow

https://cloud.google.com/dialogflow

Google: Media response (2021) Rich response

https://developers.google.com/assistant/conversational/rich-responses#df-java-media-response

Google: Save Data (2021) Save data in conversation

https://developers.google.com/assistant/conversational/df-asdk/save-data

Google: Sign In (2021) Google Sign-In concept guide

https://developers.google.com/assistant/identity/gsi-concept-guide

Google: SQL Pricing (2021) MySQL Pricing

https://cloud.google.com/sql/pricing#2nd-gen-pricing

Google: SSML (2021) Speech Synthesis Markup Language (SSML)

https://cloud.google.com/text-to-speech/docs/ssml

Jesper Wisborg Krogh (2018) *MySQL Connector/Python Revealed*

Pages 223-224 ISBN:978-1-4842-3693-2

Lau J., Zimmerman B., & Schaub F. (2018). Alexa, Are You Listening? Privacy Perceptions, Concerns and Privacy-seeking Behaviors with Smart Speakers. ACM Human-Computer Interaction, Article 102 (November 2018).

https://doi.org/10.1145/3274371

McTear M. F. (2002). Spoken dialogue technology: enabling the conversational user interface. ACM Computing Surveys 34, 90–169.

https://doi.org/10.1145/505282.505285

MySQL (2021) What is MySQL?

https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

MySql (2021b) Chapter 8 Connection Pooling with Connector/J

https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-usagenotes-j2ee-concepts-connection-pooling.html

Michael H. Cohen, J. P. Giangola, J. Balogh (2004) *Voice user interface design*

Pages 136-137, ISBN: 978-0-321-18576-1

Näkövammaisten liitto (2019) Arviot näkövammaisten lukumäärästä

https://www.nkl.fi/fi/etusivu/nakeminen/julkaisu/nvrek_vuosikirja/1_3_ar-viot_nv_lukumaarasta

Pratsam (2021) The story of Pratsam
https://www.pratsam.com/the-story-of-pratsam.html

S. Gamm, R. Haeb-Umbach (1995). User interface design of voice controlled consumer electronics. Philips Journal of Research, Volume 49, Issue 4, Pages 439-454,
https://doi.org/10.1016/0165-5817(96)81590-7

S. Schlögl, G. Chollet, M. Garschall, M. Tscheligi, and G. Legouverneur (2013). Exploring voice user interfaces for seniors. In Proceedings of the 6th International Conference on PErvasive Technologies Related to Assistive Environments (PETRA '13). Association for Computing Machinery, New York, Article 52, 1–2.
https://doi.org/10.1145/2504335.2504391

Techradar (2019) Google Nest Mini release date, price and features
https://www.techradar.com/news/google-nest-mini-release-date-price-and-features

Ye Yuan, Stryker Thompson, Kathleen Watson, Alice Chase, Ashwin Senthilkumar, A.J. Bernheim Brush, Svetlana Yarosh (2019). Speech interface reformulations and voice assistant personification preferences of children and parents. International Journal of Child-Computer Interaction, Volume 21, Pages 77-88,
https://doi.org/10.1016/j.ijcci.2019.04.005

Yeasmin, F., Das, S, & Backstrom, T. (2020). Privacy Analysis of Voice User Interfaces. In Proceedings of the 27th FRUCT conference (Vol. 27, pp. 390–395). Trento.
http://doi.org/10.5281/zenodo.4026514

Verge (2016) Google Assistant will open up to developers in December with 'Actions on Google' https://www.theverge.com/2016/10/4/13164882/google-assistant-actions-on-google-developer-sd

YourDictionary (2021) Proof-of-concept meaning
https://www.yourdictionary.com/proof-of-concept