Charles Cortez

Creating and Integrating a Google Dialogflow Chatbot with a React Native Frontend

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Bachelor's Thesis

31 March 2025

Abstract

Author(s):	Charles Cortez
Title:	Creating and Integrating a Google Dialogflow Chat-bot with a React Native Frontend
Number of Pages:	35 pages
Date:	31 March 2025
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Major:	Mobile Solutions
Instructor(s):	Matti Peltoniemi, Senior Lecturer

The objective of this project was to develop and integrate a chatbot using Google Dialogflow as the foundation and React Native as the frontend framework, enhancing accessibility to peer-to-peer and professional support for new parents.

The chatbot was implemented into the Hello There Oy mobile application using React Native, allowing cross-platform compatibility for both Android and iOS devices. Google Dialogflow was used to manage conversational logic, intent recognition, and entity extraction. For backend operations, Google Cloud Functions facilitated secure and scalable connections between the chatbot and Google Cloud SQL database application. Cloud Functions processed user input and dynamically retrieved relevant data from the SQL database as needed.

The chatbot acted as an intelligent intermediary between users and the available support options. By analyzing users' input in real time, it identified their needs and matched them with the most relevant peer or professional support. This automated matching process significantly reduced the time required for users to find appropriate help. Additionally, by incorporating real-time responses and a ranking mechanism to prioritize support options, the system improved both user engagement and overall efficiency.

The implementation demonstrated the potential of AI-driven solutions in streamlining support services, with future improvements aimed at enhancing chatbot functionality and refining NLP accuracy.

Keywords: Google Dialogflow, Chatbot, React Native, Natural Language Processing (NLP), Cloud Functions, Cloud Run, Conversational AI

Tiivistelmä

Tekijä:	Charles Cortez
Otsikko:	Googlen Dialogflow-keskustelurobotin luominen ja integrointi React Native Frontendiin
Sivumäärä:	35 sivua
Aika:	31.3.2025
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Mobile Solutions
Ohjaaja:	Lehtori Matti Peltoniemi

Insinöörityön tavoitteena oli kehittää ja integroida Google Dialogflow -järjestelmään perustuva chatbot sekä React Native -ohjelmistokehystä hyödyntävä käyttöliittymä, joiden avulla tuettiin uusien vanhempien pääsyä vertaistukeen ja ammatilliseen apuun.

Chatbot toteutettiin osaksi Hello There Oy:n mobiilisovellusta käyttäen React Nativea, mikä mahdollisti sovelluksen yhteensopivuuden sekä Android- että iOS-laitteilla. Keskustelulogiikan hallinta, aikomusten tunnistus ja entiteettien poiminta toteutettiin Google Dialogflow'n avulla. Taustajärjestelmän toiminnot rakennettiin Google Cloud Functions -palvelun avulla, joka mahdollisti turvallisen ja skaalautuvan yhteydenpidon chatbotin ja Google Cloud SQL -tietokantasovelluksen välillä. Cloud Functions käsitteli käyttäjien syötteitä ja haki tarvittaessa dynaamisesti tietoa tietokannasta.

Chatbot toimi älykkäänä välikappaleena käyttäjien ja tukipalveluiden välillä. Käyttäjien syötteitä analysoimalla chatbot pystyi tunnistamaan heidän tarpeensa reaaliaikaisesti ja yhdistämään heidät sopivimpaan vertaistukeen tai ammatilliseen apuun. Automaattinen yhdistämisprosessi lyhensi merkittävästi käyttäjien avun löytymiseen kuluvaa aikaa. Lisäksi reaaliaikaisten vastausten ja tukivaihtoehtojen priorisointiin rakennetun pisteytysmekanismin avulla parannettiin sekä käyttäjäkokemusta että järjestelmän tehokkuutta.

Toteutuksessa havaittiin, että tekoälypohjaisilla ratkaisuilla voidaan merkittävästi tehostaa tukipalveluiden saavutettavuutta. Tulevaisuudessa kehitystyötä suunnataan erityisesti chatbotin toiminnallisuuksien laajentamiseen ja luonnollisen kielen prosessoinnin tarkkuuden parantamiseen.

Avainsanat:	Google Dialogflow, chatbot, React Native, luonnollisen kielen käsittely (NLP), Cloud Functions, Cloud Run, keskusteleva tekoäly

---

Contents

List of Abbreviations

1 Introduction 1

2 Core Concepts 3

2.1 Natural Language Processing (NLP) in Chatbot Development 3
2.2 Conversational AI and Chatbots 6
2.3 Google Dialogflow Overview 9
2.4 Cloud Functions and Cloud Run Overview 9
2.5 React Native Overview 11

3 Google Dialogflow in Chatbot Development 12

3.1 Intent Recognition and Context Handling 12
3.2 Overview of Google Dialogflow Integration Options 14
3.3 Webhooks and Fulfillment in Google Dialogflow 18

4 Implementation 20

4.1 Project Overview and Objectives 20
4.2 Creating a Google Dialogflow Agent 21
4.3 Integrating Dialogflow with React Native via API 24
4.4 Integrating Cloud Functions with Google Dialogflow and the SQL Database 26
4.5 Integrating Webhooks for Dialogflow Fulfillments 28

5 Conclusion 31

References 33

List of Abbreviations

AI:	Artificial Intelligence
API:	Application Programming Interface
JSON:	JavaScript Object Notation
NLP:	Natural Language Processing
UI:	User Interface

1 Introduction

Chatbots have become an increasingly prominent tool in the delivery of digital services, offering real-time assistance through natural language conversations. Powered by advancements in Artificial Intelligence (AI) and Natural Language Processing (NLP), modern chatbots are capable of interpreting user intent, managing context, and providing dynamic, personalized responses. These systems have been widely adopted in fields such as customer service, education, and healthcare due to their scalability, responsiveness, and ability to automate routine interactions.

The aim for this project was to develop a chatbot using Google Dialogflow as the base, integrate it into the application of Hello There Oy, which is built using React Native. When starting a new family, parents are often overwhelmed by the volume of information they need to gather for their child and may have insufficient time to study it all. This can often lead to stress and hasty decision-making. The chatbot is designed to reduce the search time needed to connect with parents who have similar experiences or professionals for assistance. The chatbot streamlines the process of finding support, making it more efficient.

Hello There Oy is an early-stage start-up company, whose main product is a mobile application for new parents. The goal of the project was to help parents easily connect with the right people and professionals, without the need of a third-party individual. The chatbot is already in use and provides the most suitable options quickly, eliminating the need for users to browse through long lists of peer supporters or professionals. Based on the information provided by the client, it offers the most appropriate match.

The reasoning behind choosing Google Dialogflow and React Native stems from the decision of Hello There Oy to transfer their database to a new platform. Google was selected because it offers both a robust chatbot system (Dialogflow) and a comprehensive database solution. Additionally, Google’s

infrastructure ensures seamless integration with their existing database, streamlining the process and enhancing overall efficiency. The decision of Hello There Oy to adopt React Native was motivated by the desire to integrate the chatbot into the existing application of the company, ensuring a seamless and unified user experience.

The current issue is that as the list of experienced parents and professionals expands over time, it becomes increasingly time-consuming for new parents to search for the right support for their specific needs. This process is often overwhelming and inefficient. The chatbot's purpose was to streamline the process by helping parents quickly find the most appropriate experienced parent or professional based on their concerns, saving time and reducing frustration.

This project involved developing a chatbot that provided results based on the user's selected category. The chatbot retrieves data from the database and displays the three most active users, ranked by the number of messages sent and recent activity within the chosen category. In cases of equal message counts, the user with the most recent activity is prioritized.

If the user prefers not to connect with another parent, they will have the option to speak with a professional. To find a suitable professional, the user can select one based on their chosen category. Currently, the chatbot displays one professional, but in the future, more will be listed. Professionals are selected randomly within the selected category, ensuring that everyone has an equal opportunity to be chosen.

By automating this process, the chatbot will significantly reduce the time and effort required for parents to find the right support. Currently, a manual search may take 1-2 minutes, but with the chatbot, it will take less than a minute. As the database expands, the search time for the user will increase, but the chatbot will still be faster than a user manually searching through the list.

This improvement, implemented as part of the current project, is expected to enhance user satisfaction, increase engagement within the Hello There Oy application, and create a more seamless support system.

2 Core Concepts

This section explores the core concepts of Natural Language Processing (NLP) and Conversational AI, which are necessary to effectively build a chatbot using Google Dialogflow. The section also provides an overview of React Native for application development and explores how Google Cloud Functions connects both Google Dialogflow and the database, enabling efficient chatbot functionality.

2.1 Natural Language Processing (NLP) in Chatbot Development

Natural Language Processing (NLP) is a key technology in chatbot development. NLP is a subfield of AI and computer science that enables computers to understand, interpret, and effectively process human language for communication. NLP enables users to interact with machines in their own language, facilitating a more accessible and user-friendly experience. NLP is important in reducing repetitive tasks and automating them, improving data analysis and insights, and enhancing search capabilities [1], as shown in Figure 1.

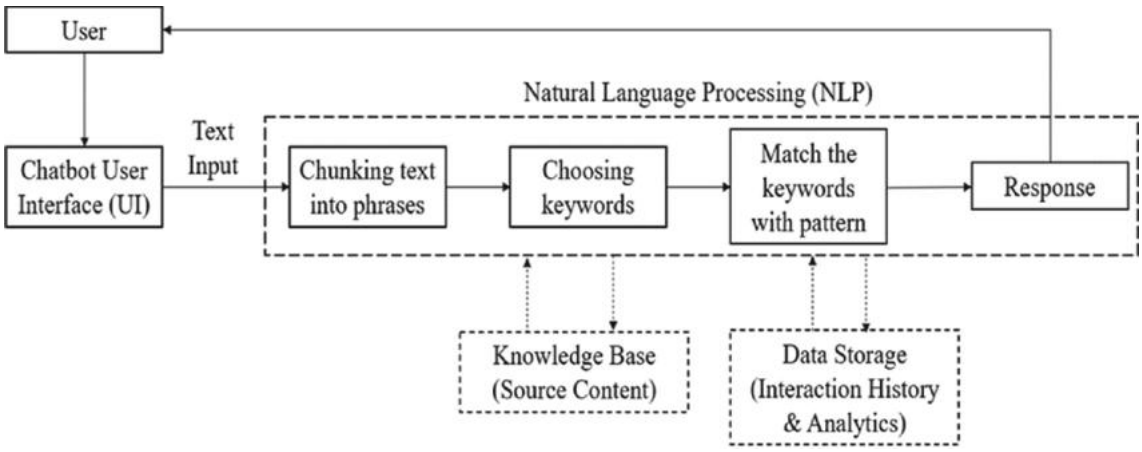

Figure 1. Natural Language Processing (NLP) process in chatbots [2].

The diagram illustrates how the NLP process in a chatbot works. First, the text is chunked into phrases that the computer understands. Second, the keywords are chosen, and then the keywords are matched with a pattern. This enables finding a helpful response needed. By following this structure, the chatbot can mimic human conversation, making interactions feel more natural for the user. [1; 3.]

The main objective of NLP is to recognize, understand, generate, and respond to text in a way that closely mimics human communication. NLP has become an essential part of modern life, appearing in various technologies encountered daily. It powers search engines, customer service chatbots, voice-activated GPS systems, and AI assistants on smartphones, such as Siri from Apple, Alexa from Amazon, and Cortana from Microsoft. [1; 3.]

Despite advancing significantly, NLP still has its challenges in recognizing human language as a regular person learns over the years. NLP must be taught by programmers to recognize and understand irregularities in language. Issues can arise, such as the introduction of new vocabulary, variations in tone, or the potential for misinterpretation. [1; 3.]

The term NLP has its roots in 1950, when a person named Alan Turing stated, "*If a machine had a conversation through the teleprinter and was able to imitate a person's behaviour so completely that there would be no noticeable difference, then it could be considered thinking.*" In 1964, one of the most significant leaps in NLP was the creation of ELIZA, a type-and-response computer program that imitated a psychiatrist. This groundwork helped pave the way for future developments in conversational AI. These advancements in NLP contributed to the development of frameworks such as Google Dialogflow, which was employed in the Hello There Oy application to facilitate the chatbot's functionality. [1.]

The way Google Dialogflow utilizes NLP techniques includes intent recognition, entity extraction, and context management. With these techniques, Google Dialogflow can understand and interact with users more smoothly. [4.]

First, with intent recognition, it analyzes the keywords and phrases from the user to determine their intent. Secondly, with entity extraction, it identifies important details in the user's input that are relevant to their request. Finally, context management helps maintain the flow of the conversation by keeping track of previous interactions, ensuring a more coherent dialogue. [4.]

Some of the well-known uses of Google Dialogflow include customer support, marketing, education, healthcare, and finance. A few examples are given below.

An example of this is Ticketmaster, which uses Google Dialogflow to help customers buy tickets or find events. It can also answer frequently asked questions. [5.]

Another example is Domino’s Pizza, which uses Google Dialogflow to simplify and enhance the user experience when ordering. Customers can place orders through the chatbot or voice-enabled devices, making the process more efficient and convenient. [5.]

A third example is KLM, which uses Google Dialogflow to assist customers with their booking, baggage tracking, and check-in. They named their chatbot "BlueBot" (also known as BB), allowing customers to easily manage their flight-related tasks in a simpler and more efficient manner. [5.]

Lastly, Volkswagen has integrated Google Dialogflow into its mobile application, allowing users to issue voice commands to access information about their car, book service appointments, and receive notifications. [6.]

These examples highlight the versatility of NLP-driven chatbots across various industries. Whether assisting customers with travel arrangements, processing

orders, or providing real-time updates, NLP enables systems to deliver context-aware, user-friendly experiences.

NLP capabilities were applied in this project to enhance the user-friendliness of the Hello There Oy chatbot for new parents. By leveraging Google Dialogflow's intent recognition and entity extraction, the chatbot can accurately identify users' specific needs and provide timely, relevant support, whether by connecting them with experienced parents or professionals.

For example, if a parent wants to discuss their mental health, the chatbot can recognize keywords and phrases related to mental health concerns. This enables it to provide the top three helper parents or connect the user to a specialized mental health professional. Additionally, NLP allows the chatbot to engage in more natural conversations with users, facilitating smoother interactions.

All in all, this will help users find helper parents and professionals faster than searching for them on their own, making the process less frustrating.

2.2 Conversational AI and Chatbots

Many people have made the mistake of thinking that conversational AI and chatbots are the same; however, they are quite different. Understanding the difference between these two chatbot types is crucial, as it directly affects how users interact with them. [7.]

The first step in this discussion is to examine chatbots. Chatbots typically follow predefined sets of rules and scripts to respond to user input. As seen in early chatbots, they were unable to fully understand a user's input. [7.]

In contrast, Conversational AI represents a broader aspect of AI-driven communication technology, with capabilities that extend far beyond those of a normal chatbot. It can understand the user's input and respond appropriately to the current situation. [7.]

As seen in Figure 2, the image highlights the key differences between Conversational AI and chatbots. [7.]

	Basic chatbot	Chatbot with conversational AI
Online 24/7	✓	✓
Natural language understanding	Keyword-based tech	✓
Dynamic, context-based navigation	Button-focused navigation	✓
Multi-level intent hierachy	If/Then statements	✓
Unlimited scalability	Limited improvement capacity	✓
Broad scope	Narrow scope	✓
3rd-party integration support	Limited understanding model	✓
Self-improving over time	⊘	✓
Consistetly high-resolution rates	⊘	✓
Omni-channel	⊘	✓
Entity extraction	⊘	✓
User authentication	⊘	✓
Voice and conversation IVR	⊘	✓
Multilingual	⊘	✓
Privacy and security compliant	⊘	✓

Figure 2. Comparison of basic chatbots and chatbots with Conversational AI [8].

As previously discussed in the NLP section, the roots of both Conversational AI and chatbots can be traced back to foundational developments in the 1950s. [1; 8.]

A good example of an early chatbot was ELIZA, which had quite limited responses. As seen in Figure 3, ELIZA could only respond by following predefined patterns, rather than truly understanding the context of the conversation. This highlights how limited early chatbots were for users, as they could not comprehend what the user was typing or talking about. [1; 8.]

In contrast to ELIZA, modern AI systems can recognize and understand user intentions, helping based on the subjects that users have typed or spoken about. Unlike early chatbots that followed straightforward patterns, modern chatbots enhance the user experience by making conversations feel more natural and remembering past interactions. [8.]

As shown in Figure 3, the AI chatbot helps users locate their packages by comprehending their requests and guiding them to find the desired items.

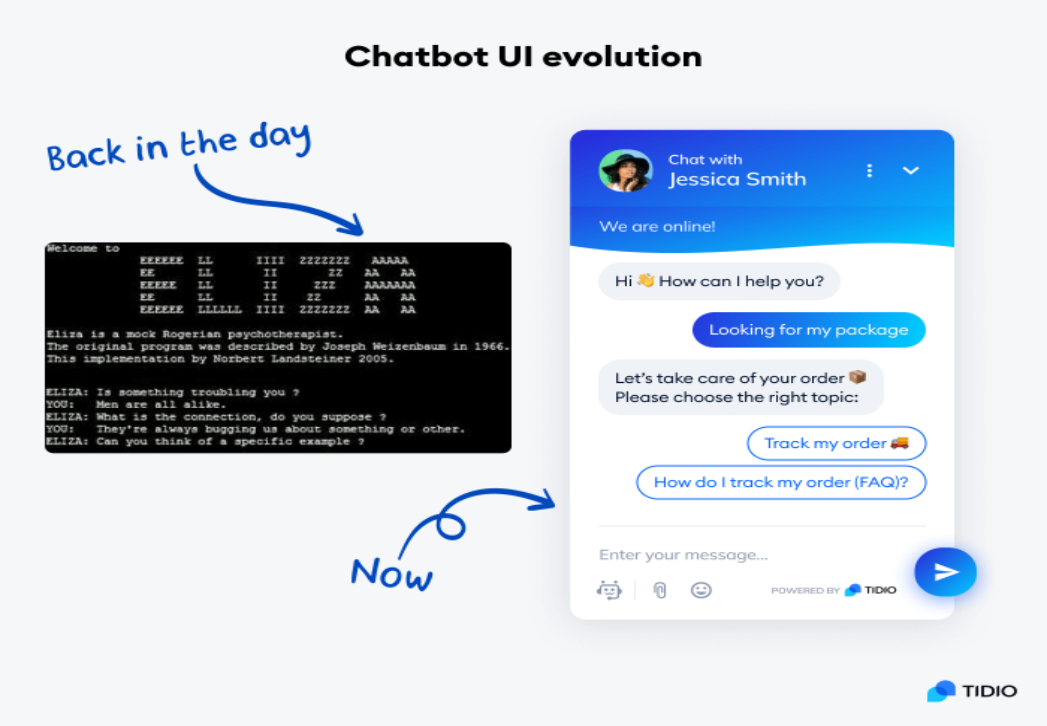

Figure 3. Evolution of chatbots from early to modern AI assistants [9].

Understanding the difference between conversational AI and chatbots is crucial for enhancing the user experiences. While early chatbots like ELIZA were limited in their responses, modern Conversational AI systems, such as those powered by Google Dialogflow, are more dynamic and capable of understanding context. This understanding is essential for developing with Google Dialogflow, enabling more natural and dynamic interactions with users.

2.3 Google Dialogflow Overview

Google Dialogflow is Google's cloud-based conversational AI platform that enables developers, companies, and individual users to create chatbots and virtual assistants capable of understanding and responding to users' natural language inputs. Google Dialogflow simplifies the development of chatbots by offering pre-built machine learning models and integrations with multiple platforms, making it an essential tool for developers seeking to implement AI-driven conversations. [10; 11.]

Following this overview of Google Dialogflow's capabilities, it is important to understand its history. Before it was known as Google Dialogflow, the platform was developed by a company called Speaktoit as API.ai in 2010, the same year the company was founded. Ilya Gelfenbeyn served as the founding CEO of API.ai, leading a team that was pioneering in the voice AI industry. Before selling their API.ai platform to Google, this company was known as the Siri of Android, as they amassed over 40 million users for their virtual assistant. Later, in 2016, Speaktoit was acquired by Google and was rebranded in 2017 as what is now known as Google Dialogflow. Since then, it has become a core component of Google Cloud's AI services. [10; 11.]

2.4 Cloud Functions and Cloud Run Overview

Google Cloud offers two serverless options: Cloud Run and Cloud Functions. While their names may sound similar, they serve different use cases and are designed to meet various development needs. Having previously discussed Google Dialogflow, it is also important to understand the history of Cloud Functions and Cloud Run to appreciate their differences and how they contribute to the overall Google Cloud ecosystem. [12; 13.]

This section begins with Cloud Functions, introduced in February 2016 as Google Cloud’s initial serverless service. With Cloud Functions, developers can set up a serverless execution environment that allows them to run simple

functions in response to HTTP requests. It also supports event-driven execution, allowing functions to be triggered by various Google Cloud services. Additionally, Cloud Functions simplifies development and automates scaling based on the number of incoming requests and events, ensuring efficient resource management. Cloud Functions is commonly used for mobile backends, webhook handling, and lightweight API implementations. [12; 13.]

Soon after the launch of Cloud Functions, Cloud Run was introduced as a more flexible serverless platform. With Cloud Run, developers can run stateless containers that automatically scale based on incoming requests. Compared to Cloud Functions, Cloud Run has several key advantages. As a container-based platform, it supports any programming language or library, offering greater flexibility in development. Like Cloud Functions, it automatically scales based on incoming requests, but it also has the capability to scale down to zero when there is no traffic, reducing costs. Additionally, Cloud Run supports HTTP/2 and HTTPS, improving communication efficiency and security. Another notable feature is its Graphics Processing Units support, making it suitable for workloads that require hardware acceleration, such as machine learning and video processing. With Cloud Run, it is possible to run long-running services, such as web applications and complex workloads, without worrying about timeouts, unlike Cloud Functions. [13; 14.]

In August 2022, a significant evolution of Cloud Functions occurred with the release of the second generation, which introduced improvements over the first generation. On August 21, 2024, Google rebranded Cloud Functions as Google Cloud Run Functions, merging Cloud Functions and Cloud Run into a unified service. [13.]

Cloud Functions were utilized to connect Google Dialogflow with the Hello There database. This integration enables seamless data access and significantly enhances usability by providing up-to-date information.

2.5 React Native Overview

React Native is a popular JavaScript-based mobile application framework created by Facebook that allows building applications for both Android and iOS simultaneously. With this framework, an application can be created for various platforms using the same code base, which significantly reduces development time and effort. [15.] Hello There Oy chose React Native because they wanted the application to work on both Android and iOS, ensuring compatibility across different devices. This approach also reduced development time by allowing a single codebase to be used for both platforms. TypeScript was used with React Native to improve type safety and maintainability, making the development process more efficient.

Understanding React Native is essential, but it is also important to grasp how it has evolved to reach its current state. Before the creation of React Native, Facebook decided to transition its service to a mobile web application based on HTML5 instead of developing a native application. However, HTML5 did not meet the desired standards, particularly in terms of the UI and performance, which highlighted the need for a more robust solution. As Mark Zuckerberg stated in 2012, "*The biggest mistake we made as a company was betting too much on HTML instead of native.*" In 2013, React Native began to take shape when Facebook engineer Jordan Walke discovered a way to generate UI elements for iOS using JavaScript. This idea was further developed during a hackathon, marking the official beginning of React Native's development. Originally, React Native was created for iOS, but Facebook quickly expanded its support to Android, allowing developers to build cross-platform applications with a shared codebase. In 2015, Facebook officially released React Native to the public, allowing developers worldwide to use it. Three years after its public release, React Native became the second most popular project on GitHub. [15.]

With the history of React Native established, the focus now shifts to TypeScript and its value as an addition to the development process. This is an extension of JavaScript that adds type definitions, reducing the chance of mistakes and

making the language easier to work with. TypeScript is an enhancement of JavaScript that includes a static type checker, allowing errors to be detected before code execution. Its error checking ensures that values match their expected types, reducing runtime bugs. Since it enforces static typing, TypeScript is considered a typed superset of JavaScript. Using TypeScript with React Native helps reduce the number of errors, maintain a cleaner codebase, and streamline development across multiple devices. For these reasons, TypeScript was chosen as the codebase for Hello There Oy. [16.]

3 Google Dialogflow in Chatbot Development

This section will explore Google Dialogflow in chatbot development in more detail and explain how it works. It will cover intent recognition and context handling, provide an overview of Google Dialogflow integration options, and discuss webhooks and fulfillment.

3.1 Intent Recognition and Context Handling

In the context of Google Dialogflow and Natural Language Processing (NLP), intent recognition is a fundamental feature that allows the platform to understand user interactions effectively. As mentioned earlier, intent recognition analyzes the keywords and phrases in the user's input to determine their underlying intention. [17.]

Intent Recognition is a crucial feature of Google Dialogflow, enabling the platform to understand user queries without relying on rigid, rule-based approaches. This capability allows the chatbot to interpret user intent more flexibly, reducing the need for users to follow a predetermined script or make guesses. [17.]

As show in Figure 4, when a user asks about the weather, date, and location, Dialogflow's agent can identify that the primary intent is to inquire about the weather. It also extracts relevant information, such as the location and date,

from the user's input. This capability allows for a more responsive and accurate interaction between the user and the chatbot, ultimately enhancing the user experience. [17.]

Figure 4. Example of weather intent recognition [17].

After exploring intent recognition, the next crucial aspect to address is context handling. With context handling, the chatbot can remember relevant details from the conversation, allowing it to maintain continuity and respond appropriately based on previous interactions. Regular chatbots without context handling ignore previous statements and treat each user input as an isolated message. [18.]

Google Dialogflow uses input and output contexts to track the conversation's flow. Input contexts help control intent matching by defining the expected context for the current user input. On the other hand, output contexts store important information from the current interaction, which is then passed to the next user input. [18.]

As shown in Figure 5, when a user types a message into Google Dialogflow, the system matches the intent with the appropriate response. With its capability to remember the conversation, Google Dialogflow allows smoothly transitioning from discussing the account details directly to asking about the balance without having to repeat any information. The system will remember what was typed and understand the context, allowing for more natural and seamless conversations. [19.]

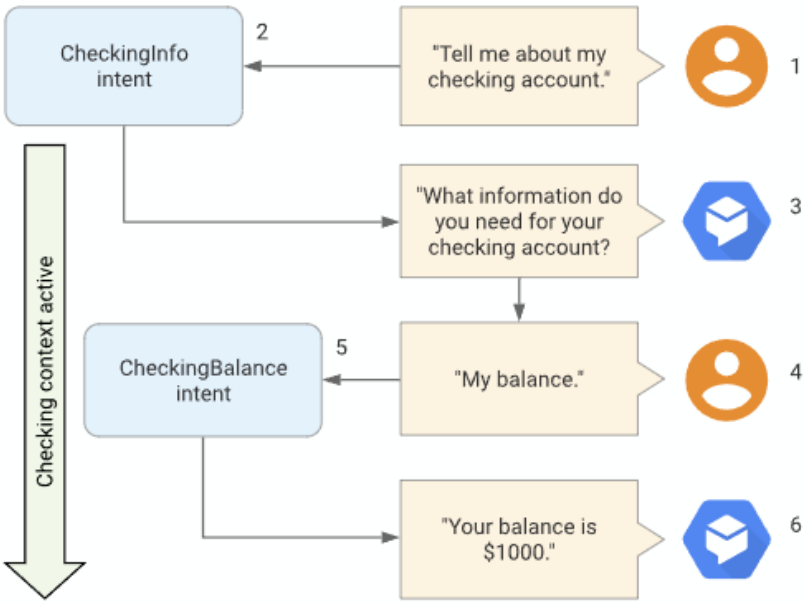

Figure 5. Diagram showing an example that uses context for a banking agent [19].

In summary, intent recognition and context handling are essential features in Google Dialogflow. These two capabilities significantly enhance a chatbot's ability to interact effectively with users. For the "Hello There" application, leveraging these features will improve the overall user experience, allowing the chatbot to better understand and respond to user queries in a more natural way.

3.2 Overview of Google Dialogflow Integration Options

This section explains how Google Dialogflow can be connected to a mobile application, as Google Dialogflow offers multiple integration methods, including Dialogflow built-in integrations, partner telephony integrations, Google-contributed open-source integrations, and independent integrations. Integrating Google Dialogflow will enhance the user experience by providing easy access to top helper parents and making it simpler to find the right professional. Additionally, with Google Dialogflow, users will have an easier time accessing and retrieving data from the database, streamlining the process of matching users with the appropriate support. [20.]

Before diving into the specifics of the chosen integration method, it is important to first explore the available integration options that Google Dialogflow offers. Understanding these methods will provide context for the approach the implementation of this approach is intended for the application.

The process begins with one of the integration options: Partner Telephony Integrations. These are not directly created by Google but are developed through collaborations between Google and third-party partners. The current partners in the list are AudioCodes, Avaya, SignalWire, and Voximplant. Since these integrations are made by collaborators, Google does not provide support for them. [20.]

This example focuses on SignalWire, one of Google's telephony partners, which can be integrated through Google's one-click telephony integration. This integration enables the addition of telephony functionality to a Google Dialogflow interface. [21.]

The benefits of using SignalWire include the ability to dial a telephone number and experience conversational AI agents in action. SignalWire's built-in Automatic Speech Recognition (ASR) and Text-to-Speech (TTS) enable advanced voice capabilities. Additionally, with conversational AI agents, it is possible to route the caller to other agents or users as needed. [21.]

The second known integration option is Google Dialogflow built-in integrations. Unlike Partner Telephony Integrations, which rely on third-party collaborators, built-in integrations are fully supported by Google Dialogflow. These integrations can be configured directly within Google Dialogflow console, making setup more straightforward. The available built-in integrations in Google Dialogflow include Dialogflow Messenger, Dialogflow Phone Gateway, Dialogflow Web Demo, Messenger from Facebook, Workplace from Facebook, Google Assistant (legacy), Google Chat, LINE, Slack, and Telegram. The main benefit of Google's own built-in integrations is the simplicity of integrating them into an application

or website, thanks to their native support and easy configuration within Google Dialogflow console. [20.]

This example focuses on Dialogflow Messenger. With the help of Dialogflow Messenger, a developer can easily add Google Dialogflow to their website without the hassle of building a chatbot from scratch. It offers the benefit of customization, allowing developers to tailor the chatbot to their needs. Once integrated, the chat interface will appear on the lower right side of the website, and users can open or close it as they wish. [22.]

In the Figure 6 example, the focus is on four ways a developer can customize Dialogflow Messenger although there are many other customization options available. To begin, consider the upper right section of the figure. This area allows for the customization of Dialogflow Messenger appearance using Cascading Style Sheets (CSS). As shown in the figure, developers can modify elements such as the title bar color, the user message background color, the bot message background color, and the color of the chat close icon. There are many more options, but for this example, only minimal customization is presented. [22.]

The second style for customizing Dialogflow Messenger, seen in the lower-left side of the figure, allows developers to create suggestion chip responses. These chips appear as clickable buttons that users can press to navigate to a link reference. Developers can customize the number of chips, as shown in the figure where there are two, and add both text and images to each chip. Additionally, as mentioned earlier, developers can link each chip to a reference, so when the user clicks on a chip, it takes them to the specified link reference. [22.]

The third style for customizing Dialogflow Messenger, seen in the lower-right side of the figure, allows developers to create an image response type. It is a simple one, where the image is displayed as a card that the user can click or tap. As the developer, it is also possible to add alternative text for the image in case it fails to load. [22.]

The last response type discussed for Dialogflow Messenger allows developers to combine multiple response types, as seen in the upper-left side of the figure. In this example, the developer has combined an image, text information, and suggestion chips. [22.]

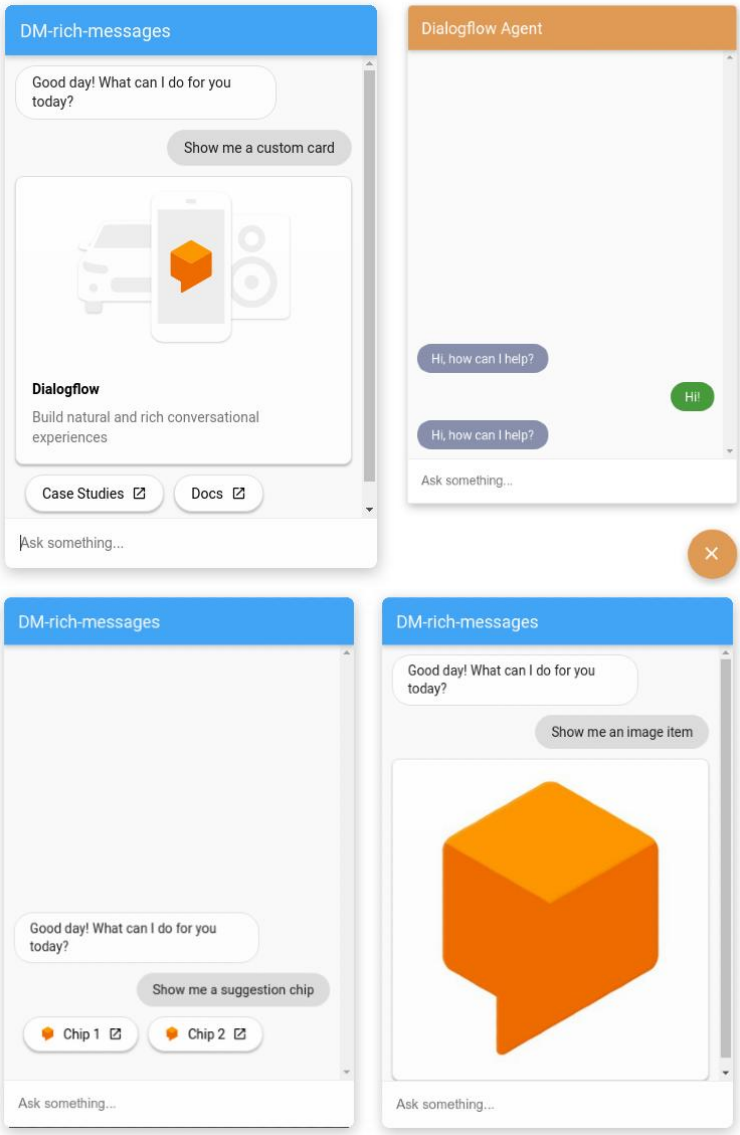

Figure 6. Demonstration of Dialogflow Messenger Rich Responses UI [22].

Having explored one of Google Dialogflow's built-in integrations, Dialogflow Messenger, the focus now shifts to Google-contributed open-source integrations.

Google-contributed open-source integrations are projects that Google has made publicly available. Developers can fork, modify, or even publish their own versions of these integrations. The following are some of the open-source integrations contributed by Google: Kik, Skype, Spark (Cisco WebEx), Twitter, and Viber. [20.]

Here, X (formerly known as Twitter) can be used as an example. In its open-source integration, a user can automatically send their own tweet to Google Dialogflow. For example, a direct message can be sent, and the integrated Google Dialogflow can respond to it. It can even reply to tweets. However, an important part of this integration is not just Google Dialogflow—it also requires confirmation and authentication through the X API. [23.]

Another similar example is Viber, where users can interact with chatbots that are powered by Google Dialogflow. Through this integration, Viber chatbots can respond to messages from users, providing automated replies for general inquiries. [24.]

Finally, the integration method that Hello There Oy will use is the independent integration, where the project utilizes Google Dialogflow API. The reasoning behind this choice is that a simpler, more cost-effective approach to building the chatbot was preferred. This method allows integrating Dialogflow with the Hello There application while maintaining flexibility and control over the implementation.

Each integration method has its benefits and drawbacks. However, for this project, the biggest advantage of the independent integration is the flexibility and control it offers, which is why this approach was selected.

3.3 Webhooks and Fulfillment in Google Dialogflow

This section explores the general concept of webhooks and fulfillments in Google Dialogflow and how they enhance the user experience with the chatbot.

The first step was to define what a webhook was in Google Dialogflow. A webhook enabled Google Dialogflow to generate dynamic responses using NLP, validate data, or trigger actions in backend services, providing real-time information. [25.]

Next, this section explores fulfillment. With Google Dialogflow, fulfillment can handle multiple tasks. It can send webhook requests, save parameters that users send, and even provide static responses. The specific actions depend largely on how the developer configures the fulfillment setup. [26.]

Having covered the basics of webhooks and fulfillment, this section will explore an example to illustrate how they work in practice. As seen in Figure 7, the example begins with Step 1, where the user types or speaks to the application. The application then sends the user's input to Google Dialogflow.

In Step 2, during intent matching, Google Dialogflow processes the input and attempts to match it with the appropriate intent based on what the user has typed or said.

Step 3: If the developer has set up a webhook, Google Dialogflow will send a request to the webhook service. This service will then execute an action based on the developer’s configuration.

Step 4 (Action): The webhook will fetch live data from the backend system, depending on how the developer has programmed it. For example, the webhook might retrieve the total number of registered users.

Step 5: After processing the request, the webhook sends the response back to Google Dialogflow.

Step 6: Google Dialogflow then forwards this response to the chosen integration method set by the developer (e.g., a mobile application, website, or messaging platform).

Step 7: Finally, the end user receives the response from the chatbot, completing the interaction.

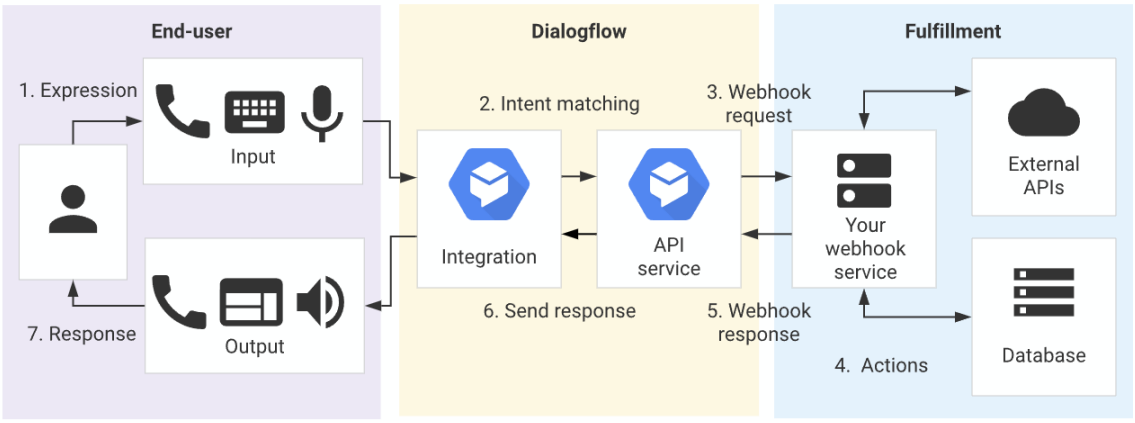

Figure 7. Webhook and Fulfillment Process in Google Dialogflow [27].

In summary, webhooks and fulfillment in Google Dialogflow are powerful tools that enhance its capabilities. By allowing the chatbot to send requests via the webhook, it enables the chatbot to retrieve live data from the backend without the need for a separate person to fetch the data. With fulfillment, the chatbot can provide more dynamic responses compared to static ones. Together, these two features enhance the chatbot's capabilities and significantly improve the user experience.

4 Implementation

This section discusses the implementation of Google Dialogflow API via React Native, including the steps required to make the integration functional. Key features of Google Dialogflow and its interaction with React Native are also explored.

4.1 Project Overview and Objectives

For this project, the goal is for the Hello There application to connect with Google’s Dialogflow chatbot, which will interface with an SQL database to fetch

new data. This will automate the process of selecting helper parents and professionals, eliminating the need for users to manually search for the user and professionals from the list. The project aims to enhance the user experience for new and confused parents by providing them with a simple and easy way to find the help they need, reducing the time and stress of searching for assistance. The technologies used include React Native for application development, Google Dialogflow for natural language processing and conversational AI, and Google Cloud Functions to facilitate seamless communication between the SQL database and Google Dialogflow. The ultimate objective was to create an efficient, user-friendly system that streamlines the process of finding support for new parents and professionals.

4.2 Creating a Google Dialogflow Agent

The process began with the creation of a project in Google Cloud Console. Once the project was established, the next step was to access Google Dialogflow to configure an agent. To initiate this, the Dialogflow API was enabled, which permitted the creation of an agent within the selected project. During the agent creation process, a display name was assigned, the desired location was selected, the appropriate time zone was set, and the preferred language was chosen. It was important to verify the selected location, as this setting could not be modified after the agent was created.

Once the agent is created, the next step was to configure its intents. By default, Dialogflow provides a Default Welcome Intent, which can be modified to suit the chatbot's initial greeting. Intents are crucial as they determine how the chatbot responds to user inputs.

Each intent contains training phrases, which are sample user inputs that help Dialogflow understand various ways people might phrase a request. The more diverse the training phrases, the better the chatbot can recognize and match user input to the correct intent.

For example, if a user is searching for a helper parent, an intent might include training phrases like:

- “I would like to talk to someone that has the same experience as me in breastfeeding”
- “I would like to talk to someone about baby food”
- “I would like to talk with someone who knows about”

By providing a variety of training phrases, Google Dialogflow improves its ability to correctly identify user intent and respond appropriately.

However, there may be situations where the user’s input does not match any intent. In such cases, Google Dialogflow uses the Default Fallback Intent, which is triggered when no suitable match is found. With this intent, the chatbot does not appear unresponsive. Instead, it can reply with a message such as, "I didn't get that. Can you say it again?" This helps maintain a smooth user experience and prevents frustration when the chatbot encounters unexpected input. The fallback intent can also be customized to offer more helpful suggestions based on the context of the conversation.

Once all directions for each intent have been configured (such as the welcome intent, helper parent intent, and subject intent), the developer can define the flow of the conversation within the chatbot.

The user begins on the start page, which triggers the default welcome intent. From here, the conversation proceeds to selecting the type of support needed.

The interaction with Google Dialogflow begins when a greeting, such as “Hi”, is detected so the default welcome intent will recognize it. This intent typically includes a greeting. For example, the chatbot will respond with "Hi! Welcome to ParentBot."

Additionally, a parameter can be introduced to track if the user has already triggered the welcome intent. By using session or context parameters, a condition can be set to ensure that the user is not redirected to the welcome intent once it has been activated. For example, once the user says "Hi" and the chatbot responds with the welcome message, a session parameter (e.g., $session.params.userGreeted != true) can be set. This will allow us to prevent the chatbot from returning to the welcome intent in subsequent interactions, keeping the conversation flow smooth.

After the greeting, the user will be directed to the next page, where the chatbot will ask, "Which one would you like to talk with today: a helper parent or a professional?" This question will be handled by two separate intents:

- The helper parent intent recognizes if the user is looking for a helper parent. A good example would be that the user will type “I would like to talk to a helper parent today.”

- The professional intent recognizes if the user is looking for a professional. A good example would be that the user will type “I would like to talk to professional today.”

Both the 'helper parent' and 'professional' intents will direct the user to their respective pages. For the professional intent, after the user selects this option, the chatbot will ask for their preferred location (Expert Location) and the category of expertise they need (Expert Category). Based on these inputs, the chatbot will list relevant professionals.

For the 'helper parent' intent, the user will be taken to a new page where they can select from categories fetched from the database. A more detailed explanation is provided in a subsequent section.

The selection made by the user will be handled by entity types. With entities, the user can type terms like "parenting," and Google Dialogflow will save it as a parameter. Then, Cloud Functions can use this parameter to fetch the relevant results from the backend. The importance of entities lies in their ability to handle synonyms. For example, "parenting" could have synonyms like "caregiving."

However, “caregiving” is avoided because it may not match data in the backend. The entity is configured to recognize and capture the correct term, ensuring that the backend can find the appropriate data.

Once the results for the top 3 helper parents have been listed, the next page will ask the user if they want to ask another question or request help from a professional. There will also be an option to end the session. However, it is important to note that Google Dialogflow sessions will automatically reset after 30 minutes of inactivity.

4.3 Integrating Dialogflow with React Native via API

This section explains how Google Dialogflow was integrated into the Hello There application, detailing the steps taken to enable seamless communication between the chatbot and the mobile application.

To begin, the chatbot UI was designed to align with the visual style of the Hello There application and user experience preferences. A dedicated button was implemented on the right side of the interface, allowing users to open the chatbot modal with a single tap, as shown in Figure 8.

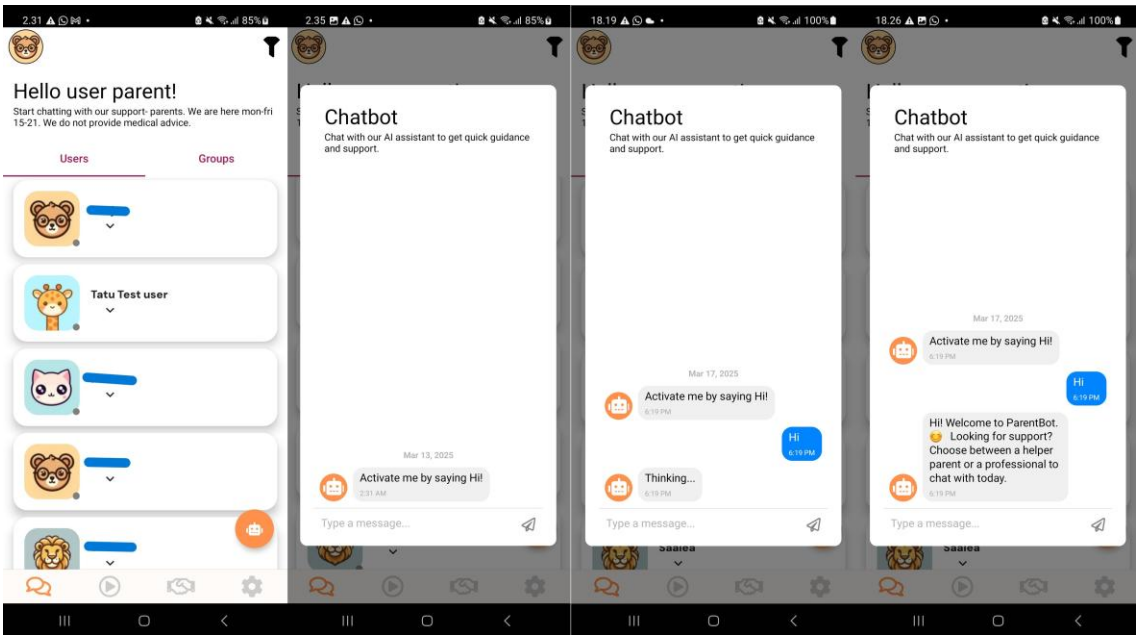

Figure 8. Hello there chatbot UI

To authenticate the communication between the Hello There application and Google Dialogflow, a service account key file is obtained from Google Cloud Console. This JSON-formatted key file contains the necessary credentials to securely authenticate the application with Google services. It includes sensitive details, such as the private key, which is essential for establishing a secure connection to Google Dialogflow. Assigning the Dialogflow API Administrator role to the service account when generating the key ensures the necessary permissions for interacting with Dialogflow. The Bearer token helps restrict access to authorized Hello There users. All authentication and authorization processes are managed in the backend, ensuring that only valid requests are processed while keeping sensitive information protected. The backend securely forwards the user's message to Google Dialogflow via an API request. The request body follows a structured JSON format, ensuring clear and consistent communication via the backend and from there to Google Dialogflow. The JSON object contains the user's input text.

Before processing the request, the system verifies access by checking the Bearer token included in the request header. While the system processes the

user's input, a "thinking" indicator is displayed in the UI, as shown in Figure 8, informing the user that the chatbot is generating a response.

Google Dialogflow processes the input, determines the appropriate response, and sends it back to the backend, which then delivers it to the user in the frontend.

The backend, after receiving the response from Google Dialogflow, processes it and forwards it to the frontend. For example, Dialogflow might return a JSON response.

Once the response is received, it appears in the chatbot UI as a message bubble, formatted to match the application’s chat design. The system ensures a smooth user experience, maintaining the natural conversational flow while clearly differentiating chatbot responses from user messages. The date is displayed above the conversation, while the time is shown within each message bubble, providing clear context for the chat history.

4.4 Integrating Cloud Functions with Google Dialogflow and the SQL Database

This section explains how the database will be accessed for Google Dialogflow using Cloud Functions

As mentioned in Section 4.2, further details are provided in this section. The Cloud Function serves as an intermediary between Google Dialogflow and the SQL database, facilitating secure data retrieval and exchange. When the user selects 'helper parents,' Google Dialogflow sends a request to the Cloud Function to retrieve the current categories containing users. Upon successful data retrieval, the information is returned to Google Dialogflow and stored in session_info.parameters.topics_list, making it available for subsequent responses or logic.

To prevent redundant data fetching, a parameter checks whether a category has already been selected. Since a new session always triggers a fresh data fetch, this ensures users receive the most up-to-date information.

When the user selects a category, it is saved as a parameter within Google Dialogflow and included in the request body sent to the Cloud Function. The Cloud Function then uses this parameter to query the database for the top 3 entries from the selected category. If no results are found, the Cloud Function ensures Google Dialogflow provides an appropriate response, such as notifying the user that no matches were found.

Before fetching the data, however, it is essential to ensure that the necessary permissions are in place for database access. Once the permissions are configured, the Cloud Function responds by either sending the session information with the list of categories or the fulfillment response with the top results, depending on whether a category is selected.

Next, it is necessary to establish a secure connection between the Cloud Function and the SQL database. This is achieved by creating a connector that handles both authentication and network configuration. This connector enables secure communication with the database, whether using a public or private IP. The authentication process involves specifying the instanceConnectionName, which identifies the location of the database, as well as the database credentials (username, password, and database name). These details are necessary for establishing a secure connection within Google Cloud environment.

The following code demonstrates how to use the Cloud SQL connector to securely connect to a Cloud SQL instance in Google Cloud. It manages both authentication and network configuration, as shown in Listing 1.

```
const mysql = require('mysql2/promise');
const {
     Connector
} = require('@google-cloud/cloud-sql-connector');

const connectWithConnector = async () => {
     const connector = new Connector();
     const clientOpts = await connector.getOptions({
          instanceConnectionName: 'your-project-id:your-region:your-in-
stance-id',
          ipType: 'PUBLIC' || 'PRIVATE',
     });

     return mysql.createPool({
          ...clientOpts,
          user: 'your-db-username',
          password: 'your-db-password',
          database: 'your-database-name',
     });
};
```

Listing 1. Setting Up Google Cloud SQL Connector with MySQL Pool in TypeScript [28].

This code demonstrates how to use the Cloud SQL connector for database access. It requires the instanceConnectionName to identify the Cloud SQL instance and credentials (username, password, and database name) to establish a secure connection. Once this setup is complete, the Cloud Function can securely access the database through the Cloud SQL connector, ensuring proper configuration and authentication.

4.5 Integrating Webhooks for Dialogflow Fulfillments

This section explores how webhooks serve as a bridge between Google Dialogflow and Cloud Functions, enabling dynamic responses and real-time data retrieval. Unlike static responses, webhooks allow the chatbot to process user input, interact with external services, and return customized replies. The setup of the webhook for fulfillment is first covered, ensuring seamless communication between Google Dialogflow and Cloud Functions. This integration allows for more flexible interactions, making the chatbot capable of handling complex queries and retrieving live data when needed.

In this chatbot flow, each page can be configured to trigger a webhook. In this example, the focus is on the category selection process. When the user enters the category selection page, a webhook call is made to fetch the available categories. Similarly, when the user selects a category, the chatbot sends the entity via the webhook to fetch the top three results.

To use the webhook, it is necessary to set up a Google Dialogflow agent. In Google Dialogflow console, the user needs to navigate to the Manage page, then select Webhooks under Resources. While this example uses a Cloud Function, other webhook URLs can also be configured.

When setting up the webhook, the following steps need to be taken:

1. Enter a Display Name (e.g., "Category").

2. Set the Type to Generic Web Service.

3. Provide the Webhook URL, which should be the URL of your Cloud Function or other service.

4. Set the Subtype to Standard.

This configuration ensures that Dialogflow can communicate with the external service (Cloud Function) to fetch dynamic data based on user input.

Before proceeding with more complex logic, it is possible to verify that the webhook is functioning correctly by implementing a basic response. This ensures that the webhook is triggered properly when the chatbot reaches the designated page.

Listing 2 shows an example of a simple webhook test in Cloud Functions:

```
exports.webhookTest = (req, res) => {
    res.status(200).json({
        fulfillment_response: {
                messages: [{
                    text: {
                        text: ["Hello, World!"]
                    }
                }]
        }
    });
};
```

Listing 2. Simple Webhook Test Implementation

After deploying the function, it is important to trigger the chatbot flow in the Dialogflow console that invokes the webhook. If the configuration is correct, the chatbot should return 'Hello, World!' as a response. This basic test serves as a validation step to ensure the webhook is functioning properly before integrating more advanced logic into the webhook.

Once the test is completed, the integration of the webhook into the chatbot page will proceed. It will then be configured in both the Entry Fulfillment and Routes. Adding the webhook for both Entry Fulfillment and Routes is similar; however, in Routes, a parameter must be added to ensure it is forwarded to the Cloud Function.

To begin, the focus is on setting up the webhook in the Entry Fulfillment. This is where the chatbot processes incoming requests from the user and responds accordingly. The Entry Fulfillment ensures that the webhook is called as soon as the conversation reaches the entry point, enabling the chatbot to retrieve and process relevant data before providing a response.

In the Entry Fulfillment section, the webhook settings must be enabled, followed by the selection of the previously created webhook. After choosing the appropriate webhook, a tag should be added. As demonstrated in the webhook creation example, the tag "category" can be used to facilitate the process. Once the tag is added, the webhook becomes ready to handle the fulfillment process,

allowing the chatbot to dynamically retrieve and return the appropriate data to the user.

The configuration in Routes is like Entry Fulfillment, with one key difference: the addition of a parameter. This parameter is essential for the Cloud Function to access and process the relevant data effectively. Including this parameter ensures that data flows correctly from Google Dialogflow to the Cloud Function for dynamic responses.

To create the required parameter, navigate to the Parameters section. There, a Display Name must be provided, and the appropriate Entity Type should be selected. It is important to mark the parameter as required to prevent the user from proceeding without providing the necessary information. This step ensures that all essential data is collected before advancing.

5 Conclusion

The objective of this project was to design and implement a chatbot solution using Google Dialogflow and integrate it into the existing React Native application of Hello There Oy. The main goal was to streamline the process for new parents to find suitable peer support or professional help, reducing the time and complexity involved in manual searches. By combining the capabilities of Google Dialogflow, Google Cloud Functions, and React Native, the project successfully automated the matching process between users and support providers within the application.

Throughout the project, key milestones included setting up a fully functional Google Dialogflow agent, establishing secure data exchange via Google Cloud Functions, and embedding the chatbot into the Hello There Oy mobile application using React Native. Additionally, webhook and fulfillment logic were implemented to enable the chatbot to retrieve live database records, ensuring dynamic and personalized responses to user queries.

During the creation of this project, there was no opportunity to conduct user testing due to time constraints. While the technical implementation met the planned objectives, gathering feedback from real users would have provided valuable insights into the chatbot's usability and overall user experience. This represents an area for future development, where iterative testing and user feedback can further refine the chatbot’s conversational flow and efficiency.

On a personal level, this project allowed for a deeper understanding of chatbot development and the broader field of conversational AI. Valuable insights were gained into how natural language processing, AI-driven conversation flow, and backend integrations work together to create an intelligent, user-friendly system.

For future development, one of the key improvements would be the addition of Finnish language compatibility, as the current chatbot was designed primarily in English. Introducing multilingual support, starting with Finnish, would significantly improve accessibility and inclusivity, ensuring the chatbot better serves the target audience in Finland. This would involve training the chatbot with Finnish-language intents, entities, and context handling, as well as accounting for the unique grammatical structures and variations found in Finnish.

Additionally, future improvements could explore optimizing system performance to handle a growing user base, expanding professional listings, and implementing continuous testing cycles to ensure the chatbot evolves alongside user needs.

In conclusion, this project has laid a strong foundation for an intelligent support system within the Hello There Oy application, with promising potential for further development and real-world impact.

References

1 Foote, K. D. 2023. A Brief History of Natural Language Processing. Online. DATAVERSITY. <https://www.dataversity.net/a-brief-history-of-natural-language-processing-nlp/>. Accessed 21 February 2025.

2 Binekas, Hisyam, and Belgiawan, Prawira Fajarindra. 2023. Factors Influencing Satisfaction and Continuance Intention of Chatbot Users. Online. ResearchGate. <https://www.researchgate.net/figure/Process-of-NLP-Based-Chatbot-System_fig1_372759164>. Accessed 23 February 2025.

3 Stryker, Chris and Holdsworth, James. 2024. What Is NLP? Online. IBM. <https://www.ibm.com/think/topics/natural-language-processing>. Accessed 21 February 2025.

4 Dialogflow ES Documentation. Online. Google Cloud. <https://cloud.google.com/dialogflow/es/docs>. Accessed 23 February 2025.

5 Case Studies. 2020. Online. Google Cloud. <https://cloud.google.com/static/dialogflow/docs/case-studies>. Accessed 26 February 2025.

6 Top 5 Real World Uses for Google Dialogflow. Online. DevNumberTwo. <https://www.devnumbertwo.com/blog/top-5-real-world-uses-for-google-dialogflow>. Accessed 23 February 2025.

7 Bishop, Court. 2024. Chatbot vs Conversational AI. Online. Zendesk. <https://www.zendesk.com/blog/chatbot-vs-conversational-ai>. Accessed 24 February 2025.

8 What’s the Difference between Chatbots and Conversational AI? 2025. Online. boost.ai. <https://boost.ai/blog/conversational-ai-vs-chatbot/>. Accessed 24 February 2025.

9 Rajnerowicz, Kazimierz. 2024. 7 Best Chatbot User Interface Design Examples for Website [+ Templates]." Tidio Blog. <https://www.tidio.com/blog/chatbot-ui>. Accessed 24 February 2025.

10 Kinsella, Bret. 2021. Founding CEO of API.ai Acquired by Google and Now Dialogflow. Online. Voicebot. <https://voicebot.ai/2021/02/28/ilya-gelfenbeyn-founding-ceo-of-api-ai-acquired-by-google-and-now-dialogflow-voicebot-podcast-ep-197/>. Accessed 25 February 2025.

11 Speaktoit Inc. 2010. Online. ITHistory. <https://www.ithistory.org/db/companies/speaktoit-inc>. Accessed 25 February 2025.

12 Google Cloud Functions Overview. 2025. Online. Google Cloud. <https://cloud.google.com/functions/docs/concepts/overview>. Accessed 25 February 2025.

13 Lu, Yiren. 2024. Google Cloud Run vs Google Cloud Function: understanding Google's serverless offerings. Online. Modal. <https://modal.com/blog/google-cloud-run-vs-google-cloud-function-article>. Accessed 25 February 2025.

14 What Is Google Cloud Run. 2025. Online. Google Cloud. <https://cloud.google.com/run/docs/overview/what-is-cloud-run>. Accessed 25 February 2025.

15 Budziński, Maciej. 2024. What Is React Native? Complex Guide for 2024. Online. Netguru. <https://www.netguru.com/glossary/react-native#how-does-react-native-work>. Accessed 25 February 2026.

16 TypeScript for the New Programmer. 2025. Online. TypeScript. <https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html>. Accessed 26 February 2025.

17 Intents. 2025. Online. Google Cloud. <https://cloud.google.com/dialogflow/es/docs/intents-overview>. Accessed 26 February 2025.

18 Input and Output. 2025. Online. Google Cloud. <https://cloud.google.com/dialogflow/es/docs/contexts-input-output>. Accessed 26 February 2025.

19 Contexts. 2025. Online. Google Cloud. <https://cloud.google.com/dialogflow/es/docs/contexts-overview>. Accessed 26 February 2025.

20 Integrations. 2025. Online. Google Cloud. <https://cloud.google.com/dialogflow/es/docs/integrations>. Accessed 7 March 2025.

21 One-Click Dialogflow Integration. Online. SignalWire. <https://signalwire.com/blogs/product/signalwire-dialogflow-one-click>. Accessed 7 March 2025.

22 Dialogflow Messenger. 2025. Online. Google Cloud. <https://cloud.google.com/dialogflow/es/docs/integrations/dialogflow-messenger>. Accessed 7 March 2025.

23 Twitter Integration for Dialogflow CX. 2023. Online. Github. <https://github.com/GoogleCloudPlatform/dialogflow-integrations/tree/master/cx/twitter>. Accessed 10 March 2025.

24 Viber Integration for Dialogflow CX. 2023. Online. Github. <https://github.com/GoogleCloudPlatform/dialogflow-integrations/tree/master/cx/viber>. Accessed 10 March 2025.

25 Fulfillments. 2025. Online. Google Cloud. <https://cloud.google.com/dialogflow/cx/docs/concept/fulfillment>. Accessed 11 March 2025.

26 Webhooks. 2025. Online. Google Cloud. <https://cloud.google.com/dialogflow/cx/docs/concept/webhook>. Accessed 11 March 2025.

27 Fulfillment. 2025. Online. Google Cloud. <https://cloud.google.com/dialogflow/es/docs/fulfillment-overview>. Accessed 11 March 2025.

28 Connect using Cloud SQL Language Connectors. 2025. Online. Google Cloud. <https://cloud.google.com/sql/docs/mysql/connect-connectors>. Accessed 19 March 2025.