Mobile Device Programming

NIA

The language assistant

Subject	Mobile Device Programming (2023-24)
Students	Guillem Godoy (guillem.godoy) Biel Carpi (biel.carpi) Marc Geremias (marc.geremias) Alex Cano (alex.cano)
Professor	Alex Tarragó

laSalle
UNIVERSITAT RAMON LLULL

Table of contents

1) Introduction 3
2) SDKs 3
3) Why choosing OpenAI API? 4
4) Design of our mobile app 5
4.1. Splash View 6
4.2. Onboarding 7
4.3. Authentication 8
4.4. Register and Login 9
4.5. Home 10
4.6. Profile 11
4.7. Information about Nia 12
4.8. Subscription 13
4.9. Frequently Asked Questions 14
4.10. Change Language 15
4.11. Timeline 16
5) Backend 17
6) Observed problems 19
7) Conclusions and reflections 20
8) References 20

1) Introduction

We present NIA, an artificial intelligence assistant or companion for people who want to learn a new language or train their speaking skills. This AI is trained with the API of OpenAI, so all the possible responses you can expect from ChatGPT are incorporated in NIA.

The app is so simple that everyone can use it. Once you passed a simple register process (only if you aren't registered yet), you get access to the main menu, whereby clicking on the main button you can start chatting with your intelligent assistant. You can ask whatever you want, and NIA will try to answer as good as possible. Moreover, NIA is also able to answer in any language you can imagine asking and learn from the context you've been given to improve its answers.

During the whole conversation with NIA, a user-friendly interface will be showing all the historic of the messages send and received. That way, the user can read past messages and improve its user experience.

2) SDKs

To enable interaction with Nia, it is essential to use the microphone SDK and speakers. These SDKs are universally available on smartphones, offering a straightforward integration process for our project.

Considering our case of use, initiating a recording triggers the establishment of a WebSocket connection with our server, facilitating real-time audio streaming for enhanced API latency. Once the recording concludes, the server processes the request, giving its corresponding response. Consequently, the phone displays the chat and plays Nia's audio.

In the future, we plan to keep improving our app by adding more SDKs., such as the camera SDK for profile photo capture. In addition, we plan to implement biometric authentication to streamline user login processes, eliminating the need for passwords. These advancements will further enhance the user experience and functionality of our application.

3) Why choosing OpenAI API?

Given the recent progress or developments in AI and the requirements of our application, we decided to use the OpenAI API to empower our application and take profit of the possible advantages offered by AI-driven conversational chat such as ChatGPT. In addition, our application's user case represents one of the basic uses of the ChatGPT API Integration Services: The Virtual Assistant.

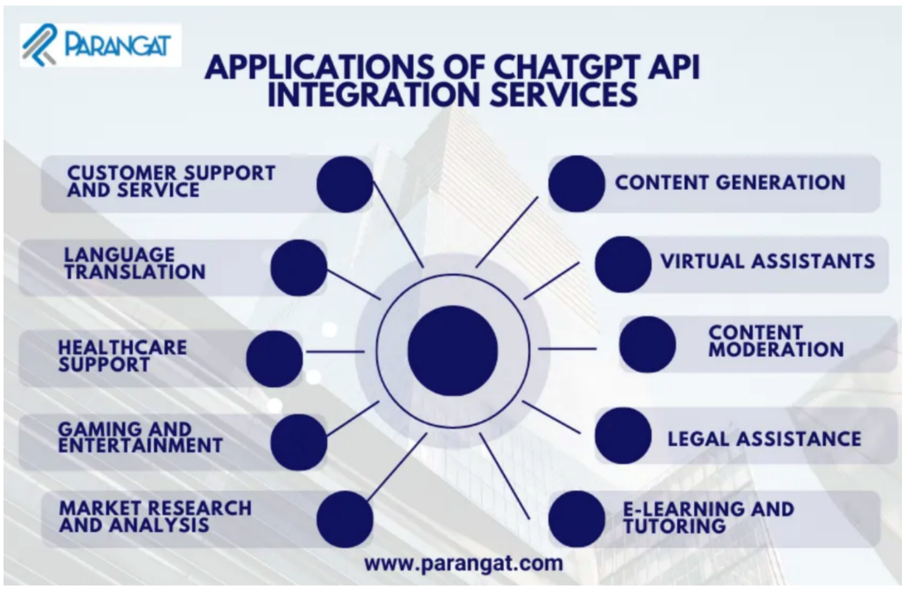


Figure 1: Schema of applications of ChatGPT API Integration Services.

With the integration of this API, we're able to use the most of ChatGPT's power to improve how users interact with our app, create personalized experiences, and easily scale our application in the future. It's a smart choice that gives our app the ability to understand and process language naturally, making conversations smoother and more effective. What's more, this integration lets our apps have interactive and context-aware chats with users, offering natural conversations and improving the user experience of the app.

4) Design of our mobile app

After completing various smaller projects in the Mobile Development subject, we concluded that a more robust project structure was essential for future development, one that is reliable, scalable, and easier to work with.

In pursuit of this objective, we initiated research to explore various options. Considering our goal of creating a cross-platform application that can seamlessly deploy on any smartphone operating system, we discovered that the most efficient way to achieve this was by adopting a framework. After careful consideration, we opted for the GetX framework due to its lightweight nature, simplicity, and stability.

GetX is a state management and dependency injection framework for Flutter that simplifies and accelerates the development process. It provides an easy-to-use and intuitive syntax for managing global state, navigation, and dependency injection. The framework's lightweight footprint ensures efficient performance, and finally ensures scalability to the project by decoupling the View, presentation logic, business logic, dependency injection, and navigation.

Moving forward, the different screens of the app will be broken down and explained, providing a comprehensive understanding of the project's architecture and functionality.

4.1. Splash View

The application starts with a splash view where users can see the NIA logo while they wait for the application to start. Once the app is launched, the flow is redirected to show the onboarding screens of the system.

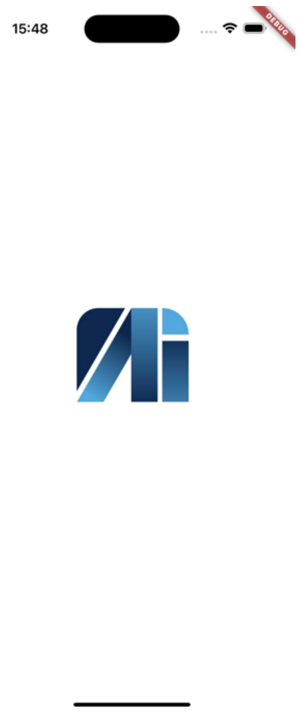
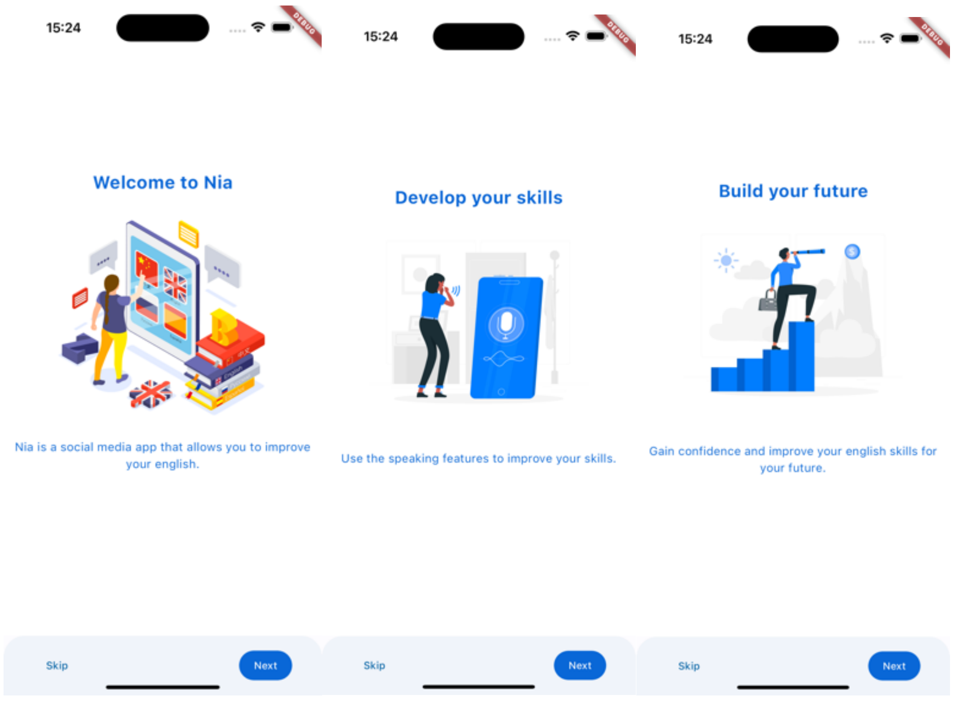


Figure 2: Splash screen of NIA.

4.2. Onboarding

The aplication has three basic onboarding screens where some information of the app is displayed. Here, the user can briefly read which features has the app and for what purpose it was created.



Welcome to Nia

Nia is a social media app that allows you to improve your english.

Develop your skills

Use the speaking features to improve your skills.

Build your future

Gain confidence and improve your english skills for your future.

Figure 3: Set of onboarding screens of NIA.

4.3. Authentication

When a user accesses the app, after passing all the onboardings, it is redirected to a menu where the user can choose whether Login (if it already has an account or using Apple, Google or Facebook) or Register in the app.

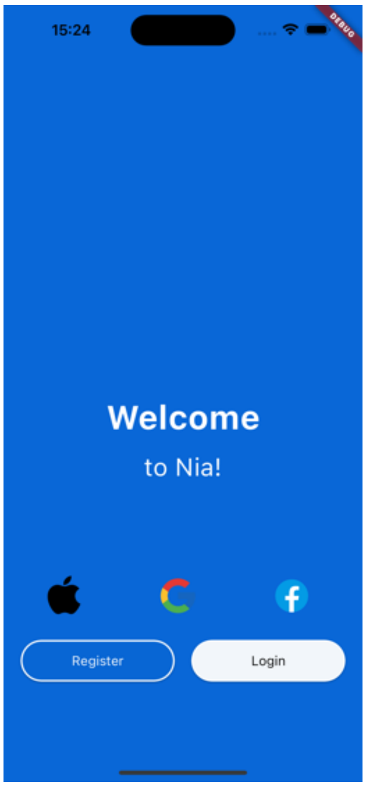
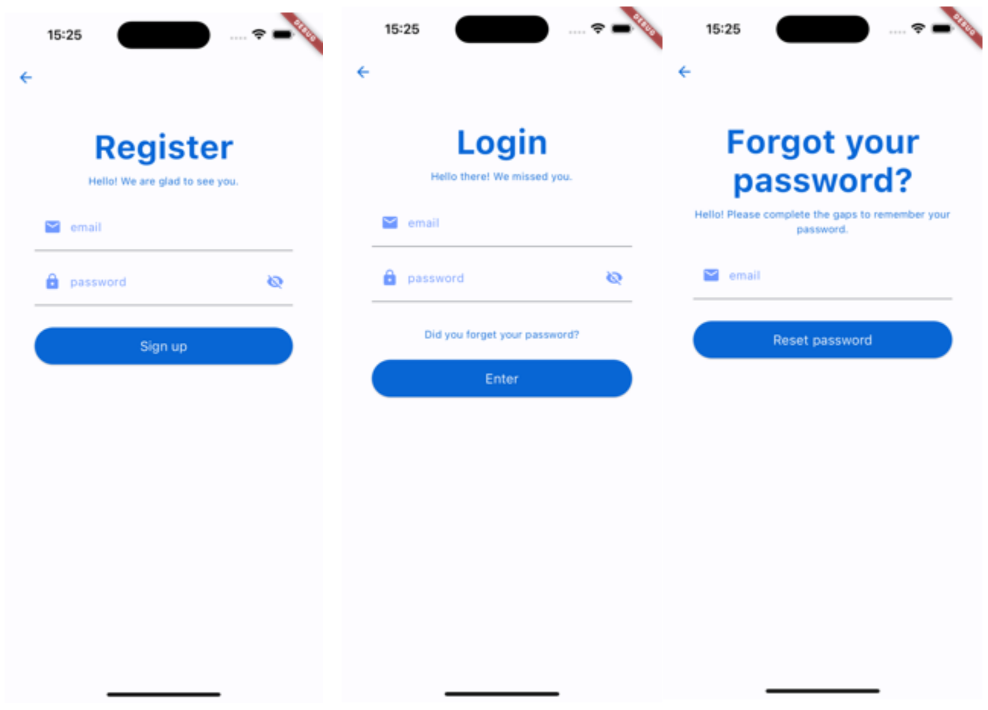


Figure 4: Screen to manage authentication processes of NIA.

4.4. Register and Login

To utilize the application, it is essential to register within the platform. During the signup process, you are required to provide your email and password, which will serve as your credentials for accessing the application. Additionally, you have the option to streamline the authentication process by using your Google, Facebook, or Apple accounts. The implementation of these registration and authentication features is facilitated through the integration of Firebase dependencies.



Hello! We are glad to see you.

Hello there! We missed you.

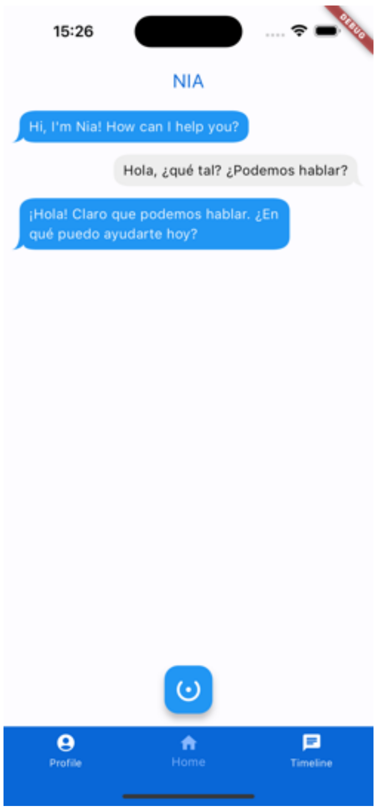
Did you forget your password?

Hello! Please complete the gaps to remember your password.

Figure 5: Set of login, register and forget password screens of NIA.

4.5. Home

The Home screen of NIA is the main element of the application and basically implements the key features of the app. In this screen you can interact with NIA using the microphone button, move to your profile or even though see a timeline of all the chats you've had with the virtual assistant.



Hi, I'm Nia! How can I help you?

Hola, ¿qué tal? ¿Podemos hablar?

¡Hola! Claro que podemos hablar. ¿En qué puedo ayudarte hoy?

Figure 6: NIA's Home screen.

4.6. Profile

The profile screen is an essential component that provides a personalized user experience and functionalities focused on the user's profile. This screen is where users can access various features related to the application.

The ProfileController is a GetxController, a key piece in the application architecture pattern that uses GetX for state management and navigation. This handler is responsible for loading and updating the user's profile details, email and profile picture. Upon start-up, the controller retrieves the current user's data from Firebase Authentication. This includes the profile image URL, which is stored in a reactive variable, allowing any changes to this data to be automatically reflected in the user interface. The ProfileScreen is where the display of this screen is implemented. The user will be able to see and change the profile photo, see their email, and interact with the different screens implemented.

Finally, the profile screen offers a logout option, allowing users to log out of the application and be redirected to the authentication decision screen.

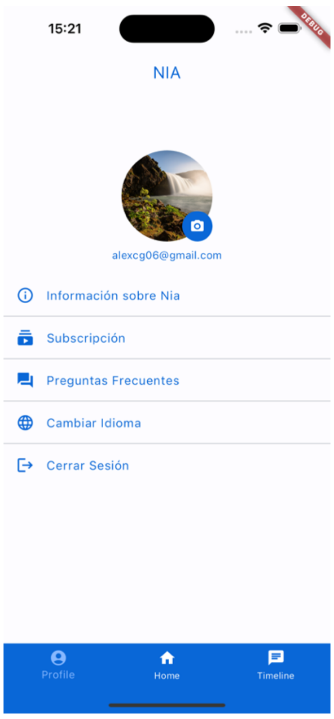
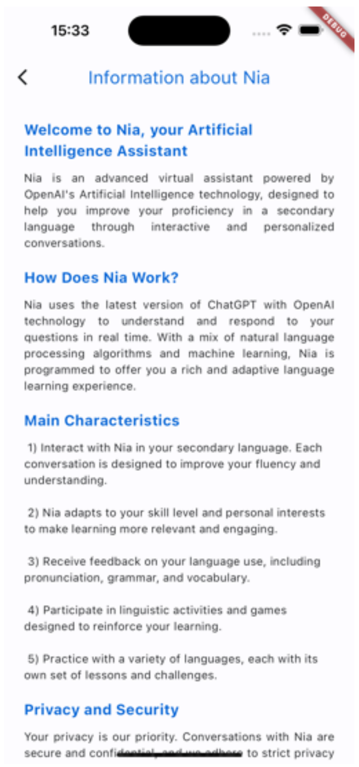


Figure 7: Profile screen of NIA.

4.7. Information about Nia

The information screen is a crucial component designed to inform users about Nia, an artificial intelligence-based virtual assistant. This screen not only serves as a hub of information about Nia's features and functionality, but also reflects the development team's commitment to transparency and user education.

The design of nia's information view has a modular and easy-to-follow approach, using the easy_localization package for internationalization, allowing content to be adapted to different languages and cultures.



Welcome to Nia, your Artificial Intelligence Assistant

Nia is an advanced virtual assistant powered by OpenAI's Artificial Intelligence technology, designed to help you improve your proficiency in a secondary language through interactive and personalized conversations.

How Does Nia Work?

Nia uses the latest version of ChatGPT with OpenAI technology to understand and respond to your questions in real time. With a mix of natural language processing algorithms and machine learning, Nia is programmed to offer you a rich and adaptive language learning experience.

Main Characteristics

1) Interact with Nia in your secondary language. Each conversation is designed to improve your fluency and understanding.

2) Nia adapts to your skill level and personal interests to make learning more relevant and engaging.

3) Receive feedback on your language use, including pronunciation, grammar, and vocabulary.

4) Participate in linguistic activities and games designed to reinforce your learning.

5) Practice with a variety of languages, each with its own set of lessons and challenges.

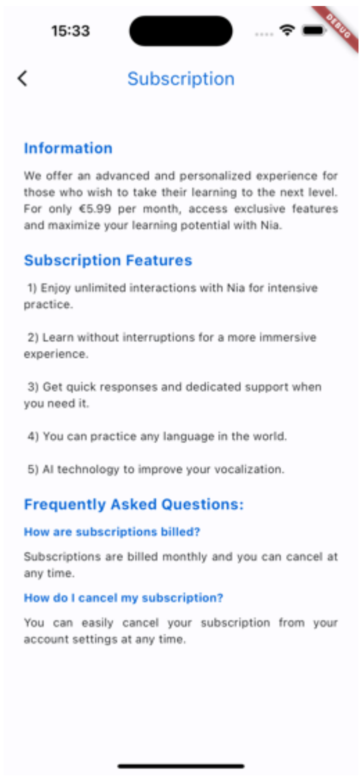
Privacy and Security

Your privacy is our priority. Conversations with Nia are secure and confidential, and we adhere to strict privacy

Figure 8: Screen to display some information about the app.

4.8. Subscription

This screen is designed to inform users about available subscription options, their features, and answers to frequently asked questions. This screen is essential for users considering purchasing the app through a paid subscription.


15:33

Subscription

Information

We offer an advanced and personalized experience for those who wish to take their learning to the next level. For only €5.99 per month, access exclusive features and maximize your learning potential with Nia.

Subscription Features

1) Enjoy unlimited interactions with Nia for intensive practice.

2) Learn without interruptions for a more immersive experience.

3) Get quick responses and dedicated support when you need it.

4) You can practice any language in the world.

5) AI technology to improve your vocalization.

Frequently Asked Questions:

How are subscriptions billed?

Subscriptions are billed monthly and you can cancel at any time.

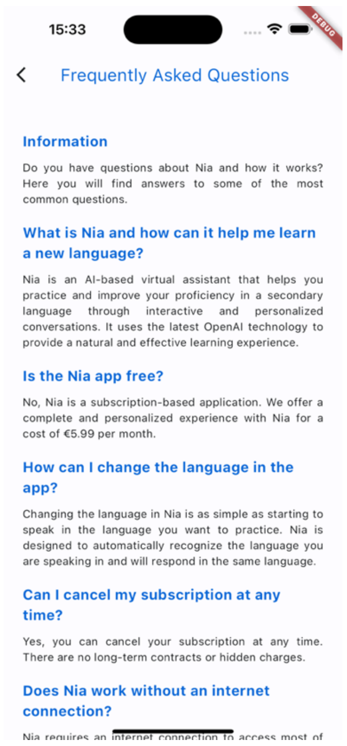
How do I cancel my subscription?

You can easily cancel your subscription from your account settings at any time.

Figure 9: Informative screen about NIA's subscription system.

4.9. Frequently Asked Questions

This screen offers users an accessible and organized space where they can find answers to frequently asked questions related to the application. This screen is essential in any user-centric application as it provides crucial information and resolves common queries, thus improving the overall user experience.



15:33

Frequently Asked Questions

Information

Do you have questions about Nia and how it works? Here you will find answers to some of the most common questions.

What is Nia and how can it help me learn a new language?

Nia is an AI-based virtual assistant that helps you practice and improve your proficiency in a secondary language through interactive and personalized conversations. It uses the latest OpenAI technology to provide a natural and effective learning experience.

Is the Nia app free?

No, Nia is a subscription-based application. We offer a complete and personalized experience with Nia for a cost of €5.99 per month.

How can I change the language in the app?

Changing the language in Nia is as simple as starting to speak in the language you want to practice. Nia is designed to automatically recognize the language you are speaking in and will respond in the same language.

Can I cancel my subscription at any time?

Yes, you can cancel your subscription at any time. There are no long-term contracts or hidden charges.

Does Nia work without an internet connection?

Nia requires an internet connection to access most of

Figure 10: Display of frequently asked questions related with NIA's features.

4.10. Change Language

This language change screen demonstrates a user-centric approach to app design, providing users with a simple and effective way to customize their experience based on their language preference. The combination of a clear interface, effortless language switching functionality and the integration of modern development tools such as GetX and easy_localization reflects a well-thought-out application adaptable to an English, Spanish and Catalan language audience.

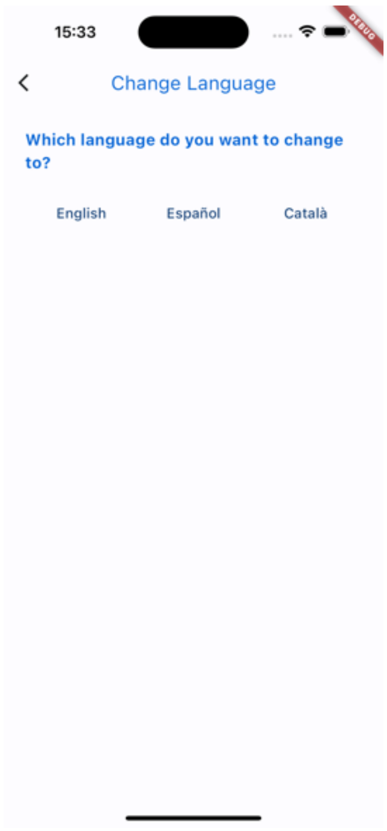
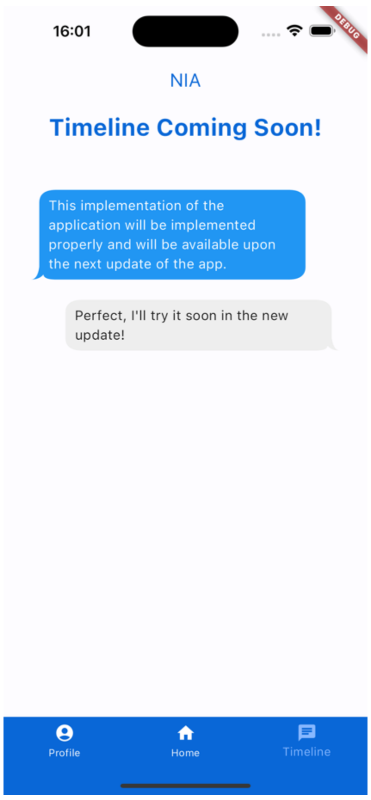


Figure 11: Accessibility screen to change language of the app.

4.11. Timeline

The Timeline screen was going to be a view of your timeline with Nia. On this screen you could go to any previous chat that you would like to have a conversation with Nia again.

Due to lack of time, it is a future innovation that we will have in the application in the next Nia update. That is why the explanation is represented with a small chat with Nia explaining to the user that in the future they will be able to use the timeline.



Timeline Coming Soon!

This implementation of the application will be implemented properly and will be available upon the next update of the app.

Perfect, I'll try it soon in the new update!

Figure 12: Timeline screen to view the historic of a chat with NIA.

5) Backend

The main concern regarding our app architecture is how could we implement the functionality of the Nia language assistant itself, and in a way that users weren't able to abuse it.

As we explained, we took advantage of the OpenAI API and its language models, and a project like this one wouldn't be able with such amazing technologies we currently have.

The API lets you fine-tune any model to adapt it to the behaviour you want, and that was the ideal use case with Nia: to pretrain the model as best as we can and let it know how it has to help us.

The first problem we faced was whether to call or not the OpenAI API client-side. We saw this can be a pretty bad idea, as we needed to make our API key public, and that could be a problem to our wallet (the API is very expensive).

So, there was no doubt: we needed a backend that controlled that only authenticated users can run our model and talk to it. We decided Go would be great, and we used the *go-openai* package to make things easier on server-side.

The interaction between client and backend is the following:

1. Once we want to talk with Nia (e.g. we press the record button), the client opens a WebSocket connection with our backend.

2. The audio is streamed to the backend in binary (currently using *aac* codec), which sends it to the Whisper service at OpenAI API.

3. Once the audio is processed, we send the client a text with its voice, through the open WebSocket.

4. This text is sent to a GPT3.5-Turbo service, which will generate a response for that specific input

5. This response is sent back to the client too, and then sent to the TTS service at OpenAI API.

6. Once we have the text, we stream it in binary (again, using *aac*) to the client, and finally end the communication.

The main issue we faced during this process was to find a way to speed up the whole communication. We first started with an approach that the user saved the file locally when he ended speaking, and the same back again. This could take more than 10 seconds for a simple "Hi! How are you?" -> "Good, thank you".

Using WebSockets helped a lot, as we could send everything without opening new connections for every communication. But it wasn't enough.

We delved deep into audio codecs and use cases of them, and found out PCM16 is the one always used for streaming in binary:

- In server, Whisper can start analysing the audio while it's arriving, and it is way quicker.

- Back to the client, when the response is sent, the audio can start playing as soon as we get some bytes.

Although it can seem trivial, implementing PCM16 communication is hard and must be well-managed and configured between server and client. Due to lack of time, we adopted a half-way approach, as neither Whisper nor TTS that the OpenAI API gave us don't support PCM16. If we keep improving the project, we could run both in our own server (they are Open Source) and achieve this kind of codec support.

At the end, we achieved very low latency and fast response time, which can be further improved a lot.

We also hosted the backend at Google Cloud App Engine's flexible environment, which gave us the scalability, performance, and tools we required for our project.

6) Observed problems

Considering what we've commented above, after successfully completing various smaller projects related with Mobile Development, we decided that a more robust project structure was needed this time for our app development: One that was reliable, scalable, and easier to work with. To do so, we decided to work with GetX, so the first problem we faced was to learn how to use GetX properly.

Even though the GetX framework structure is based on the Model View Controller (MVC) pattern and is similar to PHP Web Development, we encountered some initial difficulties when determining where to initialize the controllers and how to manage the flow of the app. Figuring out how to use GetX for routing was key for our app development, so through exploration and learning from online resources and the GetX documentation, we eventually got everything to work. As we learned more about GetX, we improved gradually and successfully overcome the initial challenges.

We also talked about the latency problems we had with the interaction between client-server, and how we solved them in the Backend part. This was the hardest part of the project by far, and we got very frustrated at start when seeing such response times. After delving deep into codecs and implementing a WebSocket approach, we are satisfied of the system we achieved.

7) Conclusions and reflections

In conclusion, we find NIA represents a very useful tool for language learning and speaking skill development. Capable of responding in any language, NIA adapts and learns from each interaction. Not only this, but also it is continuously refining its responses based on context, so that it contributes to a dynamic and personalized learning experience.

Furthermore, the user-friendly interface ensures accessibility for learners of all levels and all types. It is practical and simple to use, and all the elements of the app are clearly distinguishable.

Apart from improving a lot our Flutter knowledge, we focused much more on the server side, critical for any app like this. We learnt about the Go programming language and more Firebase features, which will be valuable knowledge to apply on any other app we may made in the future.

We enjoyed a lot programming Nia, but we feel we would have needed more time to achieve what we really wanted to. Although it's very functional, useful, and funny to use, it is in an immature state and not ready to be launched at all. We plan to keep improving it and launch it later this year, and we are very proud of what we have achieved until now.

8) References

Vaddoriya, P. (2023, October 3). *Flutter : GetX State Management*. Retrieved from Medium: https://prashantv03.medium.com/flutter-getx-state-management-f60d59b5778b

GitHub. (2023). *Flutter Getx Documentation* . Retrieved from GetX Document: https://chornthorn.github.io/getx-docs/