UNIVERSIDAD DE LAS AMÉRICAS PUEBLA
ENGINEERING SCHOOL
DEPARTMENT OF INFORMATICS, ELECTRONICS AND MECHATRONICS

Desarrollo de Aplicaciones Móviles (LIS4012)

Project Progress

Professor: Julio Noé Hernández Torres

Cecilia Soriano Tochimani, 173633
Melisa Sampieri Espinoza, 173948
Luis Felipe Jarquin Romero, 186812

Spring 2025
March 20th, 2025

Tasks Accomplished by Each Team Member by March 6th

Team member 1: Cecilia Soriano Tochimani

Cecilia has been focused on developing the app's user interface using SwiftUI in Xcode. She has designed and implemented various views, ensuring a seamless and intuitive user experience. Her work includes creating layouts, animations, and navigation flows to make the app visually appealing and user-friendly. Additionally, she has been refining the app's responsiveness and accessibility to ensure a high-quality user experience across different iOS devices.

Tasks done:

- The splash frame has been completed.
- The home view is in progress and has advanced to 30% completion.

Team member 2: Melisa Sampieri Espinoza

Melisa has been working on integrating the BERT pretrained model into the app for emotion recognition and tone analysis. She is responsible for adapting the model to fit the specific needs of the app, leveraging her experience with BERT. Additionally, she has been testing different hyperparameter configurations to optimize performance. She is also exploring ways to connect the model's output to the app's views for real-time suggestions while ensuring efficiency and accuracy.

Tasks done:

- Trained multiple versions of the BERT model to evaluate which performs best for our application.

Team member 3: Luis Felipe Jarquin Romero

Luis has been focused on integrating the OpenAI API to enhance the app's functionality, particularly for text rewriting and tone refinement. He is working on establishing a secure connection between the app and the API to ensure smooth data processing. Additionally, he is optimizing the API's performance to provide quick and accurate responses to user inputs. To achieve this, he has collaborated with another team member to research the best implementation strategies and ensure compliance with security standards. He has also been involved in approving and reviewing the progress made by other team members to ensure alignment with project goals.

Tasks done:

- Conducted research with a teammate on how to effectively implement the OpenAI API.
- Approved and reviewed the progress of different implementation aspects to ensure consistency and efficiency and responsible to make the documentation

Tasks Accomplished by Each Team Member by March 13th

Team member 1: Cecilia Soriano Tochimani

Cecilia has been focused on developing the app's user interface using SwiftUI in Xcode. She has designed and implemented various views, ensuring a seamless and intuitive user experience. Her

work includes creating layouts, animations, and navigation flows to make the app visually appealing and user-friendly. Additionally, she developed another view for the app, keeping it clean and easy on the eyes.

Tasks done:

- Fixed errors in the home view.
- Implemented a feature where the identified sentiment is highlighted with color, fulfilling one of the app's requirements.

Team member 2: Melisa Sampieri Espinoza

Melisa worked on converting the BERT model into a CoreML so that it can be implemented in Swift UI.

Tasks done:

- Find versions of BERT to include it in the application
- https://huggingface.co/docs/transformers/model_doc/distilbert#tfdistilbertformaskedlm
- https://apple.github.io/coremltools/docs-guides/source/convert-tensorflow-2-bert-transformer-models.html
- https://huggingface.co/dlaiymani/bert-base-ner-coreml
- https://github.com/huggingface/swift-coreml-transformers/tree/master

Team member 3: Luis Felipe Jarquin Romero

Luis researched methods for cybersecurity. A log in is obviously necessary and there may be messages that the user would like to further protect. Also, it is important to encrypt users' passwords. After considering Argon2, PBKDF2, and bcrypt, we decided to go with bcrypt because of how easy it is to implement while maintaining strong security [1]. It is much faster than Argon2, as the hashing is done in less than one second [2].

Tasks done:

- Deciding on bcrypt for Swift UI called SwiftBcrypt.
- Updating documentation on GitHub.

Justification of the project

In developing VibeCheck, we focused on building a mobile-first experience that delivers real-time emotion recognition and tone improvement suggestions. Our priority was ensuring the app could perform fast, accurate, and privacy-conscious natural language processing (NLP) across multiple languages and platforms. To support this, we explored, evaluated, and integrated a range of

transformer-based models, culminating in the selection and fine-tuning of compact yet capable pre-trained models.

We experimented with several BERT-based architectures and trained different model versions tailored specifically for emotion recognition. This iterative approach allowed us to evaluate trade-offs in model performance, size, and inference speed, ensuring the best fit for both mobile deployment and user experience. We successfully converted our selected model to CoreML, enabling seamless integration within our SwiftUI-based interface and allowing on-device processing—critical for maintaining user privacy and reducing server load.

Beyond just adopting existing technologies, we invested in optimizing real-world usability: we integrated sentiment visualization into the UI, added secure authentication with bcrypt for password management, and implemented OpenAI's GPT-4o mini model to support text rewriting and suggestion features under budget constraints.

Our work reflects a hands-on approach: evaluating model architectures, implementing training pipelines, handling model conversion and optimization, and embedding NLP functionality directly into the app. These efforts ensured that VibeCheck remains fast, responsive, and secure, while still providing high-quality emotion and tone analysis across a diverse user base.

Pretrainded models

DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. DistilBERT is a small, fast, cheap and light Transformer model trained by distilling BERT base. It has 40% less parameters than google-bert/bert-base-uncased, runs 60% faster while preserving over 95% of BERT's performances as measured on the GLUE language understanding benchmark. VibeCheck, requires emotion recognition in text and automated suggestions for tone improvement while being multilingual and efficient. DistilBERT is a great choice for use case due to the following reasons:

- Lightweight & Efficient for Mobile Applications
- Emotion Recognition & Sentiment Analysis
- Multilingual Capabilities
- Pre-trained Model with Fine-Tuning Flexibility.
- Compatibility with iOS (CoreML & TensorFlow Lite)
- Low Latency & Fast Inference
- Integration with API-based Workflows

DistilBERT is a smaller, faster, and cheaper variant of BERT, making it ideal for mobile applications where computational resources are limited. It retains 97% of BERT's accuracy while being 60% faster and 40% smaller, which reduces latency in real-time applications. Since the app requires real-time text analysis and inline suggestions, using a pre-trained DistilBERT model can save time and computational resources while allowing further fine-tuning for your specific needs. DistilBERT models can be converted to ONNX, TensorFlow Lite, or CoreML, making them compatible with iOS devices. DistilBERT can be deployed on-device (using CoreML/TensorFlow Lite) rather than a cloud-based API, ensuring privacy and compliance with data regulations.

- General Language Understanding Evaluation (GLUE) Benchmark: DistilBERT retains 97% of BERT's performance across multiple tasks, indicating its robust language understanding capabilities.
- IMDb Sentiment Analysis: DistilBERT achieves an accuracy of 92.82%, closely matching BERT's 93.46%. Wikipedia+5ar5iv+5Medium+5
- Stanford Question Answering Dataset (SQuAD) 1.1: DistilBERT attains an F1 score of 85.8 and an Exact Match (EM) score of 77.7, compared to BERT's 88.5 F1 and 81.2 EM scores. Medium+1ar5iv+1
- Stanford Sentiment Treebank (SST-2): A fine-tuned version of DistilBERT on SST-2 achieves an accuracy of 91.3%, whereas BERT reaches 92.7%.

BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

VibeCheck mobile application focuses on emotion recognition, sentiment analysis, and tone improvement for social media posts. Since the app needs high-accuracy NLP processing, BERT (Bidirectional Encoder Representations from Transformers) is an excellent choice.

- BERT’s Strength in Context Understanding.
- High Accuracy for Emotion & Sentiment Analysis.
- Pre-Trained Models for Faster Implementation.
- Multilingual Support.
- Ability to Generate Text Suggestions.
- Compatibility with iOS and Backend APIs.
- Secure & Private Processing.

BERT is a bidirectional transformer model, meaning it understands the full context of a word based on surrounding words. This is crucial for tone and emotion detection, where meaning can shift based on context. Multilingual BERT is trained on 104 languages, making it ideal for detecting tone across different linguistic and cultural contexts. BERT can be fine-tuned for text generation and paraphrasing, allowing it to rewrite user input with a more positive or neutral tone.

While BERT is heavier than DistilBERT, it can still be optimized for mobile using TensorFlow Lite or ONNX. BERT can be deployed in the cloud (reducing mobile device load) while maintaining secure API communication.

- BERT achieved 92.31% accuracy in classifying movie reviews as positive or negative.
- When analyzing tweets about airline services, a BERT-based model attained 87.1% accuracy.
- BERT-based models reached 93% accuracy in identifying sentiment in pandemic-related tweets.

These results demonstrate BERT's state-of-the-art performance in natural language processing, making it an excellent choice for VibeCheck’s emotion recognition and text analysis features.

Implemented Views

Using SwiftUI and Xcode, the planned screens from the prototype were successfully implemented. Additionally, some features that were not originally considered were added, such as the ability to click on a recommendation and view the saved text.

For future updates, there are plans to change the color of the pop-up that appears when clicking on the underlined text and add the functionality to edit the text.

Figura 1. Home view

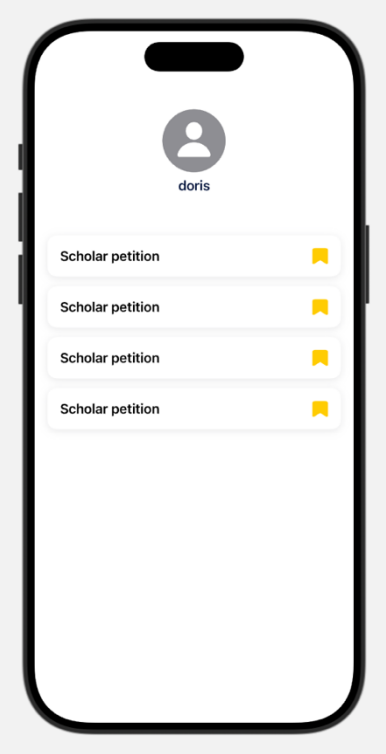

Figura 2. Profile View

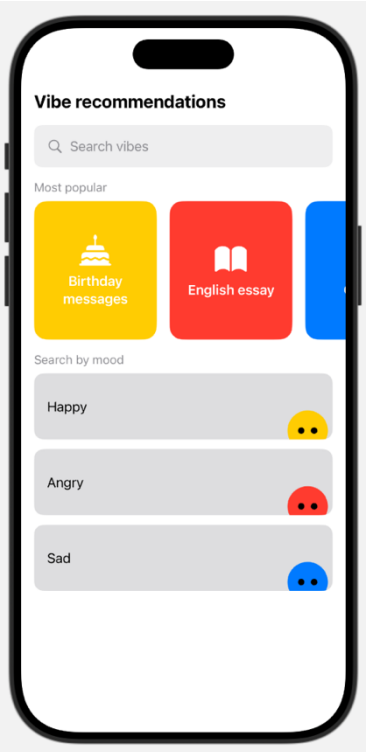

Figura 3. Recommendations View

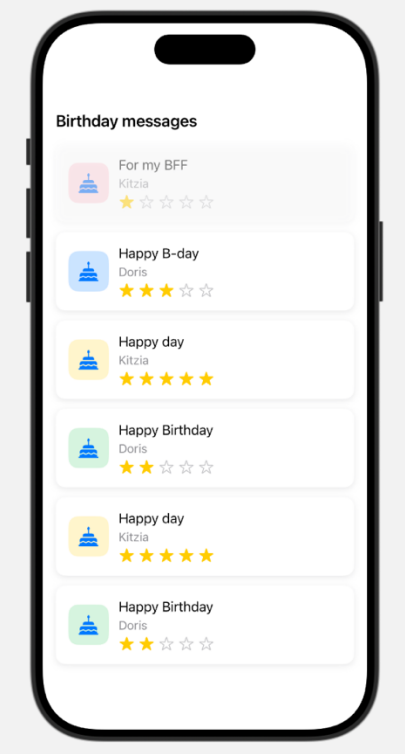

Figura 4. Recommendations Part 2 View

Figura 5. Save text View

API Choice

We chose the GPT-4o mini model for the OpenAI API because of how affordable it is considering our limited budget.

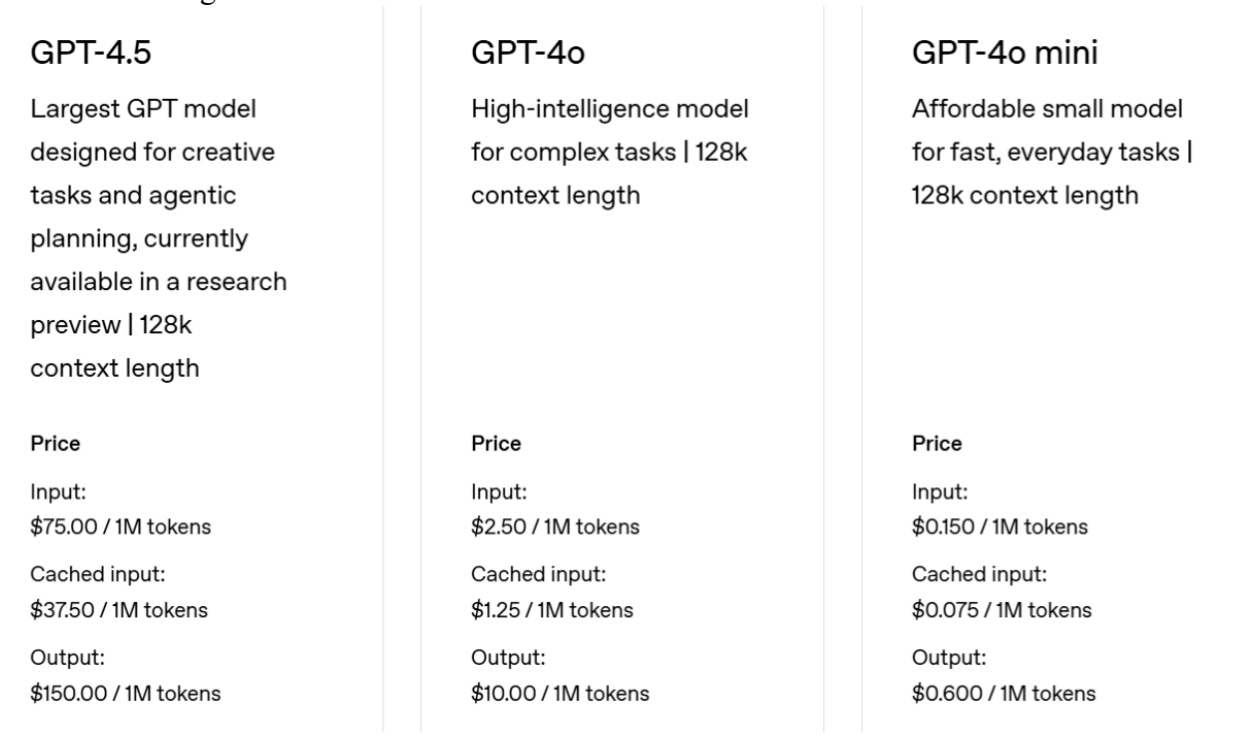

Figure 6: API choise in GPT model

Summary

User Interface Development

- Designed and implemented various views in SwiftUI.
- Fixed errors in the home view.
- Added a feature that highlights identified sentiments with colors.

Machine Learning Integration

- Trained multiple BERT model versions for emotion recognition.
- Converted the BERT model to CoreML for SwiftUI integration.
- Researched different BERT implementations and available models.

Backend & Security

- Researched and selected bcrypt for password encryption in Swift.
- Investigated OpenAI API implementation for text processing.
- Updated project documentation on GitHub.

Future Development

The next phase of development for VibeCheck will focus on building out the backend infrastructure to support scalable and secure natural language processing. This includes integrating our emotion recognition models into a robust backend system and developing custom APIs to handle real-time text analysis, tone suggestions, and user interactions. The backend will be responsible for receiving user input, running inference using our fine-tuned models, and returning structured responses that the mobile frontend can display efficiently. Additionally, we plan to

implement API endpoints for account management, sentiment tracking history, and multilingual support. Depending on performance and privacy considerations, we will deploy models either on-device via CoreML or server-side using optimized frameworks like TensorFlow or ONNX. This backend development will ensure the application is modular, maintainable, and ready for future scaling, including cloud deployment, user analytics, and integration of additional NLP features.

Tasks Accomplished by Each Team Member by March 27th

Team member 1: Cecilia Soriano Tochimani

- Using SwiftUI and Xcode, the color scheme was modified, adopting a traffic light logic—green, yellow, and red—to classify the text based on its tone.
- Additionally, when the pop-up appears, it now displays different options, and upon selecting one, the corresponding text replaces the underlined text dynamically.
- As part of the development, the SignInView and LoginView screens were created, ensuring a seamless and intuitive user authentication experience.

Team member 2 & 3: Melisa Sampieri Espinoza & Luis Felipe Jarquin Romero

Main Components of the Architecture:

Client (iOS Application developed with SwiftUI):

- Responsible for the user interface and interaction experience.
- Sends text analysis requests and receives processed results from the backend.

Backend (Firebase and complementary cloud services):

- User authentication management through Firebase Authentication.
- Secure and scalable data storage using Firestore and Firebase Storage.
- Execution of cloud functions via Firebase Functions, including communication with external APIs for emotion analysis.

Natural Language Processing API:

- Specialized service that detects emotions in text and suggests corrections or improvements.
- Direct communication with backend functions to ensure security and efficiency in data flow.

Technical and Strategic Justification of the Selected Architecture:

Scalability and Efficient Performance:

- Firebase provides a cloud infrastructure capable of scaling vertically and horizontally based on user base growth, without requiring physical server management.
- Serverless functions allow complex processes to be executed quickly and without overloading the client, ensuring a smooth user experience.

Reduction of Development Time and Costs:

- Using managed services like Firebase minimizes the effort required to implement core features such as authentication, data synchronization, and storage.
- Extensive documentation and native compatibility with SwiftUI facilitate integration and reduce the learning curve.

Ease of Maintenance and Updates:

- The separation between client and server allows backend improvements and fixes to be made without directly affecting the already distributed application.
- Firebase enables function updates or access rule changes without the need for new versions of the application.

Compliance with Security and Privacy Regulations:

- The proposed architecture includes secure authentication mechanisms, granular access control, and encryption of data both in transit and at rest.
- This setup facilitates compliance with regulations such as the General Data Protection Regulation (GDPR) and other local privacy laws.

Flexibility for Integration with Intelligent Services:

- The modularity of the architecture makes it easy to incorporate external APIs for text processing and to evolve toward proprietary artificial intelligence solutions in future phases of the project.

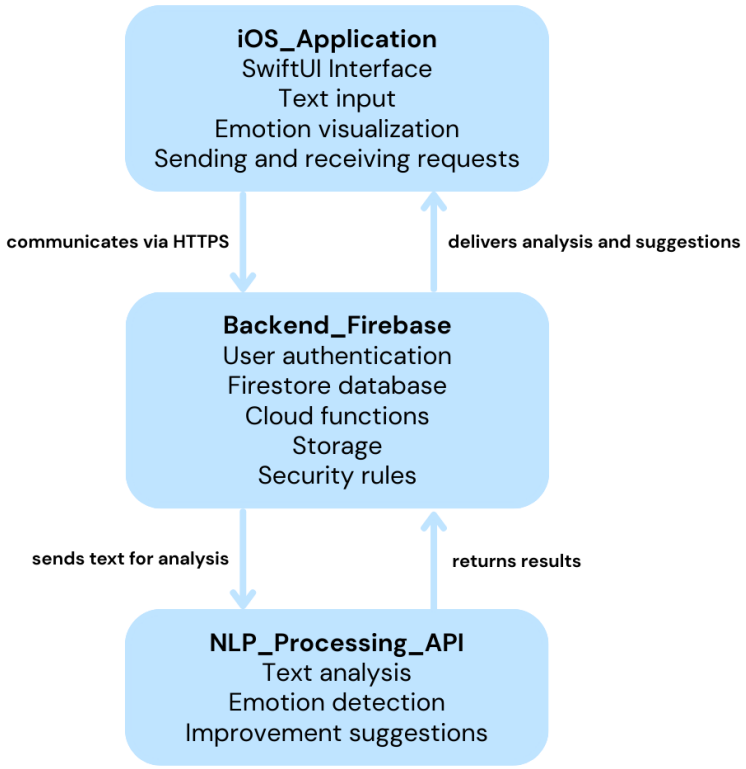

Figure 7: Proposed Architecture Diagram

The presented UML diagram represents the logical architecture of the VibeCheck system, based on a client-server model supported by cloud services to ensure a modular, scalable, and easily maintainable structure. This architecture is composed of three main components: the iOS application, the Firebase-based backend, and an external natural language processing (NLP) API.

The iOS application serves as the system's client and is the direct point of contact with the end user. Developed with SwiftUI, this interface allows users to input text, visualize the results of emotional analysis, and receive suggestions to improve the phrasing of their messages. All operations between the app and the server are carried out through secure communication using the HTTPS protocol.

The backend, implemented with Firebase, acts as the intermediary between the mobile application and the processing API. This server manages user authentication through Firebase Auth, stores texts and results in a structured manner using Firestore and executes automated functions via Firebase Functions. It also enforces access rules and security mechanisms to safeguard sensitive information and ensure compliance with privacy regulations such as GDPR.

The third component is the natural language processing API, which is responsible for analyzing textual content, identifying the emotions present in the message, and generating improvement suggestions without altering the original meaning. The Firebase backend handles sending the text to this API and receiving the corresponding analysis, which is then processed and returned to the iOS application for user presentation.

The interaction between these three modules ensures a clear separation of responsibilities, thereby allowing each component to scale, update, or be maintained independently without disrupting the

overall functioning of the system. This architecture aligns with the principles of efficiency, modularity, security, and user experience—core pillars in the development of VibeCheck.

Tasks Accomplished by Each Team Member by April 4, 2025

Team member 1: Cecilia Soriano Tochimani

- Adjusted the app's color scheme to enhance visual consistency and user experience.
- Integrated Login and Sign Up screens into the main user flow of the application.
- Researched how to implement sign-in with Google and Apple ID:
    - Google Sign-In requires setting up a Firebase project, enabling Google sign-in in Firebase Authentication, and integrating the `GoogleSignIn` SDK into the app. The user is authenticated using their Google credentials and receives a secure token to be used within the app.
    - Apple Sign-In involves enabling "Sign in with Apple" in the app's capabilities in Xcode, using the `AuthenticationServices` framework. The system handles the login UI and returns a credential that can be used to authenticate the user securely.

Team member	Contributions	How it improves the current project status
Cecilia Soriano Tochimani	• Unified the app's colors for visual consistency and better UI/UX.	Creates a more professional and cohesive user interface, improving user perception and app polish.
	• Designed and implemented separate views for login and sign-up, added navigation flow.	Adds essential authentication functionality, allowing real users to access and engage with the app securely.
	• Google Sign-In Research	Prepares the app to support widely used login methods, reducing friction and increasing user adoption potential.
	• Apple ID Sign-In Research	Supports Apple ecosystem users natively, ensuring the app meets platform expectations and supports privacy-first login

Team member2: Melisa Sampieri Espinoza.

- Conducted in-depth research on the BERT and DistilBERT models to assess their suitability for mobile deployment, especially in the context of real-time emotion recognition and tone analysis.
- Investigated various methods to download and convert BERT-based models into CoreML format, making them compatible with SwiftUI for on-device inference.

- Explored multiple GitHub repositories and open-source implementations to understand best practices and identify functional CoreML conversion pipelines for BERT and DistilBERT models.
- Initiated the first conversion trials using tools like coremltools, evaluating model compatibility, output structure, and integration feasibility within the iOS app.

Team member	Contributions	How it improves the current project status
Melisa Sampieri Espinoza	• Research on BERT and DistilBERT models	Expands the app's potential for on-device machine learning, allowing faster, private, and offline processing of emotional tone in user messages.
	• Investigated methods for downloading and converting models to CoreML.	Lays the groundwork for integrating lightweight models like DistilBERT.
	• Explored GitHub repositories with existing implementations.	Improves understanding of cross-platform NLP deployment.
	• Started initial CoreML conversion tests for integration into the app	Strengthens app scalability and technical flexibility by validating CoreML workflows.

Team member 3: Luis Felipe Jarquin Romero

- Acquire the necessary licenses and permissions for the implementation of the ChatGPT API.
- Ensure all changes and modifications comply with Apple's Human Interface Guidelines and App Store Review Guidelines.
- Conduct regular code reviews and quality assurance checks to maintain high development standards.
- Validate accessibility features to ensure the app is inclusive and meets WCAG (Web Content Accessibility Guidelines) for iOS.

Team member	Contributions	How it improves the current project status
Luis Felipe Jarquin Romero	• Acquire the necessary licenses for the ChatGPT API	Ensures legal and uninterrupted access to AI functionalities essential for emotion analysis and rewriting.
	• Ensure compliance with Apple's Human Interface and App Store Guidelines	Increases the chances of successful App Store approval and ensures a smooth, native iOS user experience.

	• Perform extensive usability testing	Helps identify UX/UI issues early, enhancing overall user satisfaction and engagement.
	• Conduct regular code reviews and quality assurance checks	Improves code reliability, reduces bugs, and accelerates future updates or feature integration.
	• Validate accessibility features (WCAG compliance)	Makes the app more inclusive, expanding the potential user base and aligning with accessibility best practices.

References:

1. T. P. Batubara, S. Siburian, and R. T. A. Pasaribu, "Analysis Performance BCRYPT Algorithm to Improve Password Security from Brute Force," *J. Phys.: Conf. Ser.*, vol. 1811, no. 1, p. 012129, 2021. Available: https://doi.org/10.1088/1742-6596/1811/1/012129.
2. Stytch, "Argon2 vs. bcrypt vs. scrypt: Which password hashing algorithm should you use?" *Stytch Blog*, May 2022. Available: https://stytch.com/blog/argon2-vs-bcrypt-vs-scrypt.

GitHub link: https://github.com/LuisAtMcGill/vibecheck.git