



Universitat de Lleida

TREBALL DE FI DE GRAU



ESCOLA
POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA
INSPIRING THE FUTURE

Estudiant: Joan Bonell Ruiz

Titulació: Grau en Tècniques d'Interacció Digital i de Computació

Títol de Treball de Fi de Grau: From Voice to Verdict: An AI-Based recitation app for Competitive Exams

Director/a: Sergio Sayago Barrantes

Presentació

Mes: Setembre

Any: 2025

CONTENT

1	Index of figures	2
2	Index of tables	3
3	Project summary	4
3.1	Abstract	4
3.2	Resum	4
3.3	Resumen	5
4	Motivation, work carried out and main results	6
4.1	Motivation	6
4.2	Overview of the work carried out	7
4.3	Main results obtained	7
5	Project Management.....	9
5.1	Original idea and evolution of the project	9
5.2	Development methodology employed	11
5.3	Project organization and task tracking.....	13
6	Tecnologies analyzed.....	20
6.1	Mobile Development Platforms.....	20
6.2	Voice Recognition Services (Speech-to-text)	20
6.3	Natural Language Processing techniques for text comparison	22
6.4	Technologies and systems analysed	27
7	Design, develeopment & testing	29
7.1	Design	29
7.2	Development	34
7.3	Testing.....	41
8	Discussion	43
8.1	Relationship with the training received	43
9	Conclusions	46
10	Acknowledgments.....	48
11	References	49

1 INDEX OF FIGURES

Illustration 1 - Project Milestones Overview and their objectives	13
Illustration 2 - Diagram of the processing pipeline - from raw text through Spacy -> Vectors -> Cosine Score	25
Illustration 3 - App - How to load a PDF in the interface.	29
Illustration 4 - App - How looks the text preprocessed for the user before start recording	29
Illustration 5 - App - The app shows to the user the state of the program in real time.....	30
Illustration 6 - App - The app returns to the user the feedback results	30
Illustration 7 - UML Diagram of the App.....	33
Illustration 8 - Cosine Similarity Formula	35
Illustration 9 - Cosine Similarity Formula	36

2 INDEX OF TABLES

Table 1 - Original idea and real evolution of the project.....	10
Table 2 - Explanation line by line	23
Table 3 - Compare texts algorithm	24
Table 4 - Tests implemented	41

3 PROJECT SUMMARY

3.1 ABSTRACT

This project entails the development of a desktop application to facilitate the study of justice administration civil service exams (particularly for judges) using voice-based techniques. The system allows users to **“recite the law”** aloud (speak memorized legal texts) and leverages **Natural Language Processing (NLP)** to evaluate the accuracy of the recitation. The application compares the speech-to-text transcription of the user’s voice against the official law text and computes a **percentage of correctness**, giving immediate feedback on errors and omissions. This undergraduate thesis documents the project motivation (born from an interest in mobile app development and AI in education), the project management and methodology used, the analysis of technologies (speech recognition, development frameworks, text comparison techniques), implementation details (system requirements, UX design, data persistence, unit testing), and a discussion of results. The work concludes that the tool helps improve candidates’ performance by offering an innovative way to practice oral exams, meeting the stated objectives and paving the way for future improvements in accuracy and features.

3.2 RESUM

Aquest projecte consisteix en el desenvolupament d’una aplicació mòbil per facilitar l’estudi d’oposicions de justícia (especialment per a jutges) mitjançant tècniques de veu. El sistema permet als usuaris **“cantar la llei”** (recitar en veu alta textos legals memoritzats) i utilitza **processament del llenguatge natural** per avaluar l’exactitud del recitat. L’aplicació compara la transcripció de la veu de l’aspirant amb el text oficial de la llei i calcula un **percentatge d’encert**, proporcionant feedback immediat sobre errors i omissions. Aquest treball de fi de grau documenta la motivació del projecte (sorgida de l’interès en la informàtica mòbil i la IA aplicada a l’educació), la gestió i metodologia emprades, l’anàlisi de tecnologies (reconeixement de veu, frameworks de desenvolupament, tècniques de comparació de text), els detalls d’implementació (requisits, disseny UX, persistència de dades, proves unitàries) i una discussió de resultats. Es conclou que l’eina ajuda a millorar el rendiment dels opositors en oferir una forma innovadora de practicar els exàmens orals, complint els objectius plantejats i obrint la porta a futures millores en precisió i funcionalitats.

3.3 RESUMEN

Este proyecto consiste en el desarrollo de una aplicación móvil para facilitar el estudio de oposiciones de justicia (en especial, la carrera judicial) mediante técnicas de voz. El sistema permite a los usuarios **“cantar la ley”** (recitar en voz alta textos legales memorizados) y utiliza **procesamiento de lenguaje natural (PLN)** para evaluar la exactitud de lo recitado. La aplicación compara la transcripción de la voz del opositor con el texto oficial de la ley y calcula un **porcentaje de acierto**, proporcionando retroalimentación inmediata sobre los errores y omisiones. El Trabajo de Fin de Grado documenta la motivación del proyecto (surgido del interés en la informática móvil y la IA aplicada a la educación), la gestión y metodología empleadas, el análisis de tecnologías (reconocimiento de voz, frameworks de desarrollo, técnicas de comparación de texto), los detalles de implementación (requisitos, diseño UX, persistencia de datos, pruebas unitarias) y una discusión de resultados. Se concluye que la herramienta ayuda a mejorar el rendimiento de los opositores al ofrecer una forma innovadora de practicar los exámenes orales, cumpliendo los objetivos planteados y abriendo la puerta a futuras mejoras en precisión y funcionalidades.

4 MOTIVATION, WORK CARRIED OUT AND MAIN RESULTS

4.1 MOTIVATION

The motivation for this project arises from the intersection of two realities. On one hand, the extreme demands of the judicial career competitive examinations: candidates must memorize hundreds of legal topics, whose statements can extend to thousands of words, according to (Justito el Notario, 2020) between 3,800 and 4,000 words per topic, even reaching 6,000 in certain syllabi. In addition to conceptually mastering this enormous syllabus, they must be able to present it orally in an impeccable manner during the exam, with a time limit per topic (e.g., ~15 minutes per topic in judiciary exams) and under the close evaluation of a tribunal. The pressure and difficulty of this oral test make the technique of “singing the law” fundamental in preparation: repeating the texts aloud until achieving fluency, precision, and confidence.

On the other hand, the author of this Final Degree Project (a Computer Engineering student) has prior training in python development and a strong interest in applied Artificial Intelligence. During his studies, he acquired knowledge in native mobile programming, databases, as well as notions of Natural Language Processing (NLP) and speech recognition techniques. With this foundation, he identified an opportunity to bring these areas together to solve the mentioned problem: How can technology help a candidate efficiently practice their oral topics without constant supervision? The idea of creating an application that would listen to the student and automatically evaluate their recitation proved very attractive due to its potential usefulness and the technical challenge it posed.

This motivation is also fueled by recent advances in the field of AI: commercial voice recognition systems have achieved accuracies close to 90–95% in audio transcription, according to (Google Cloud, 2025), even for the Spanish language, thanks to deep neural network technologies. Likewise, NLP techniques allow texts to be compared and differences detected more intelligently than a simple character-by-character matching. The project proposes leveraging these modern technologies for a specific educational use case, innovating in the way oral self-assessment is conducted for candidates.

In summary, the application is born with the goal of automating the role of the “tutor” in topic-singing practices. It offers the candidate immediate and objective feedback on how faithfully their recitation matched the legal text, pointing out where errors or omissions occurred. This improves motivation and the quality of study, as the student can identify their weak points (for example, articles they always forget or formulate incorrectly) and correct them in successive repetitions.

4.2 OVERVIEW OF THE WORK CARRIED OUT

The project followed a full software-engineering lifecycle, driven by both technical goals and the practical constraints of a Final Degree Project. We began with contextual research, investigating the structure and demands of judicial competitive exams and the “topic-singing” memorization technique. This phase validated the need for an automated support tool and guided our technology exploration in Chapter 4, where we compared available speech-to-text and NLP libraries for Spanish legal vocabulary.

With the problem space clearly defined, we moved into planning and agile project management. We formulated specific objectives, delineated the scope to focus on automated oral evaluation, and adopted an iterative development approach. Short sprints allowed us to prioritize core functionality—audio capture, transcription, and accuracy scoring—while leaving room to refine the user interface and experiment with alternative algorithms.

For implementation, we selected Python as our development language because of its rich ecosystem of libraries (e.g. SpeechRecognition, spaCy, scikit-learn) and its suitability for rapid prototyping. Python’s high-level APIs enabled us to integrate Google’s speech-to-text service, perform sophisticated text preprocessing, and compute cosine similarity with minimal boilerplate. Throughout coding, we structured the system into modular components—audio recorder, STT adapter, text processor, vectorizer, and comparator—to facilitate maintainability and future extension (Chapter 5).

Recognizing that reliability is critical in an educational context, we incorporated testing at multiple levels. Unit tests were written to verify core functions (e.g. text normalization, similarity calculation), while performance measurements ensured that even lengthy recitations could be analyzed within user-acceptable latency bounds. Beyond code tests, we conducted informal user trials with volunteers to assess real-world transcription accuracy and the clarity of feedback. Insights from these sessions led us to adjust the scoring thresholds, improve error highlighting, and streamline the interaction flow.

Finally, we performed a comprehensive evaluation against our initial objectives. We documented how each requirement was met, analyzed the system’s strengths and limitations, and distilled lessons learned. The succeeding chapters present this narrative in detail—tracing our journey from ideation through design, development, testing, and, ultimately, the conclusions and recommendations for future work.

4.3 MAIN RESULTS OBTAINED

The project culminated in a fully functional prototype of the application developed in Python. Among the most notable achievements are:

- **Integrated voice-recognition functionality:** The application captures the user’s voice and automatically transcribes their recitations of legal articles or topics. In controlled tests using modern AI services, transcription was found to achieve high accuracy, with an approximate word error rate of 7%—equivalent to correctly recognizing about 93% of words under ideal conditions of silence and clear enunciation. This validated the feasibility of voice recognition for this purpose.

- **Calculation of accuracy percentage:** An algorithm based on the inverted Word Error Rate (WER) metric was implemented to compare the transcribed text with the expected official text. This algorithm determines the percentage of correct words by computing the minimum number of edits needed to transform the spoken phrase into the reference phrase. The interface presents this result clearly and intuitively, highlighting errors such as omitted or incorrect words.
- **User interface:** The application’s design is clean and intuitive, with simple screens guiding the user through the recitation and evaluation process. A results screen prominently displays the accuracy percentage and highlights detected errors, enabling the candidate to clearly identify where mistakes occurred.
- **User validation and iterative improvement:** In pilot tests with a small group of volunteers, users provided positive feedback on the application’s usefulness for improving memorization and detecting errors. They suggested enhancements such as a feature to save score history and an option to hear the correct pronunciation of phrases—ideas slated for future updates.

In conclusion, the work carried out has yielded an innovative tool for legal training, successfully achieving the objectives of automating the evaluation of legal-topic recitations with reasonable accuracy and delivering a positive user experience. These results are detailed in Chapter 5, which also discusses the project’s limitations and the extent to which the initial goals were met.

5 PROJECT MANAGEMENT

5.1 ORIGINAL IDEA AND EVOLUTION OF THE PROJECT

The original idea of the project was to divide it into different milestones—specifically, into 11 milestones.

These milestones were initially planned to last two weeks each. However, during the execution of the project, some had to be extended, either due to professional workload or because the initially chosen technology was not the most suitable.

In the following milestones, initially planned as one week each, there were changes:

- **Milestone 4: Natural Language Processing Implementation**, initially planned for one week, was extended to six weeks because it was initially thought that this algorithm would be simpler and more intuitive.
- **Milestone 5: Development of the comparison and evaluation algorithm**, initially planned for one week, was extended to six weeks because an exact similarity algorithm was initially planned. This became a major issue due to the fact that speech recognition was not 100% reliable, and users did not always use the exact same words when using the application. At this point, the project was at risk, but the problem was solved using the cosine similarity algorithm, which takes the user's input and the legal text input and compares them as word vectors, calculating the angle between them as the similarity result.
- **Milestone 6: Design and implementation**, this phase lasted six weeks due to the professional time available at that moment.
- **Milestone 7:** This milestone was started ahead of schedule, together with the comparison algorithm, in order to truly see what feedback would be given to the end user. That is why it lasted four weeks.
- **Milestone 11: Documentation of the project**, initially, it was thought that writing the final thesis would be easier, but documenting something that involved 16 weeks of development and requirement analysis is more difficult than expected.

Table 1 - Original idea and real evolution of the project

Milestone	Week 1-2	Week 3-4	Week 5-6	Week 7-8	Week 9-10	Week 11-12	Week 13-14	Week 15-16	Week 17-18	Week 19-20	Week 21-22	Week 23-24	Week 25-26	Week 27-28	Actually
1. Requirements Analysis and Initial Planning	Green														
2. Development of the PDF Processing Module		Green													
3. Implementation of Voice Recognition			Green												
4. Natural Language Processing (NLP) Implementation				Green	Red	Red									
5. Development of the comparison and evaluation algorithm (Cosine-Similarity)					Green	Red	Red								
6. Design and Implementation of the user Interface							Green	Red							
7. Feedback to the user.					Green	Green	Green	Green							
8. Testing and Validation								Green	Red						
9. Optimization									Green						
11. Documentation of the Project									Green	Green	Red	Red	Red	Red	
12. Maintenance and Future Updates															Yellow

5.2 DEVELOPMENT METHODOLOGY EMPLOYED

To manage the development of the project, an **agile methodology inspired by Scrum** was chosen, albeit adapted to a one-person “team.” The nature of this project – involving research into unfamiliar technologies (speech-to-text, natural language processing) and an evolving understanding of feasibility – called for a flexible and iterative approach. A traditional waterfall methodology was deemed unsuitable, since waterfall assumes fixed requirements and a linear progression of phases. In contrast, agile methodologies embrace change and iteration. In fact, the Scrum framework originally emerged as a reaction to the rigidity of the waterfall model, advocating a more flexible, cyclical process that can **continuously adapt to change**, according to (Martins, 2025). This agile philosophy aligns with one of the core values of the Agile Manifesto: responding to change over following a fixed plan. Given the likelihood of many “on-the-fly” adjustments in our project (due to experiments with AI and shifting requirements), the agile approach was a natural fit.

In practice, **Scrum** is a popular agile methodology that introduces specific roles, events, and artifacts (e.g., Scrum Master, daily stand-ups, sprint reviews, backlogs) to help teams implement agility. With a single developer (the author) working on this project, there was no formal Scrum team – there were no separate Scrum Master or Product Owner roles – but many Scrum-inspired practices were still applied. Scrum (and agile in general) emphasizes iterative development in short cycles and frequent reassessment of priorities. Adopting this mindset, the work was planned in **short iterations** analogous to two-week sprints. At the start of each iteration, specific goals were set (just as one would do in a Sprint Planning meeting), and at the end, the outcomes were reviewed and lessons noted (similar to a Sprint Review/Retrospective). This iterative cycle allowed the project to incorporate feedback and newfound insights continuously. It’s worth noting that while Scrum is designed for teams, a solo developer can still **benefit from agile practices** – for example, timeboxing work into sprints, doing frequent reviews, and maintaining a backlog of tasks. (Certain Scrum elements obviously do not apply with one person – there is no need for daily stand-up meetings in front of a team, for instance – but the essence of agility, such as being responsive to change, still very much applied.)

One of the first steps in implementing the agile methodology was to define the project requirements in terms of **user stories**. In agile development, user stories are short, simple descriptions of a feature told from the perspective of the end user (typically following the format “As a [type of user], I want [some goal] so that [some reason/benefit].”). These helped ensure the project always kept end-user needs in focus. A list of user stories was created at the outset (available in the project repository) covering the core functionalities envisioned. For example, two key user stories defined were:

- *“As an exam candidate, I want the app to listen to me recite a full topic so that I can evaluate myself.”*
- *“As a user, I want to see which words I pronounced incorrectly so that I can correct my mistakes.”*

Each user story captured a specific user goal and the benefit of that feature. This technique of writing requirements from the user's perspective is a hallmark of agile because it frames development in terms of user value. In fact, **user stories are essentially small increments of functional requirements** developed within Agile; they describe what the user wants to do and why. All the identified user stories were collected into a **product backlog** – an ordered list of everything that needed to be done for the project.

From these user stories, concrete development tasks were derived. Each story was **broken down into technical tasks** that would realize the required functionality. For instance, from the story *“listen to me recite a full topic,”* tasks such as *“Integrate speech-to-text API,”* *“Implement audio recording interface,”* and *“Handle audio stream segmentation”* were identified. Similarly, from the story about seeing mispronounced words, tasks like *“Implement text comparison algorithm”* and *“Highlight incorrect words in UI”* were created. This breakdown helped in estimating effort and planning the implementation sequence. Initially, a simple prioritization was done to decide which features (and their tasks) were most crucial for an MVP (Minimum Viable Product) and which could be deferred. In agile terms, this was akin to **backlog grooming** – ensuring the most important tasks were ready to be tackled first.

5.3 PROJECT ORGANIZATION AND TASK TRACKING

<p>Milestone 11: Maintenance and Future Updates</p> <p>Objectives: - Establish a plan for ongoing application maintenance. - Plan future updates and improvements. Activities: - Monitor performance and collect user feedback. - Fix errors that may arise after deployment. - Update the application with new...</p> <ul style="list-style-type: none">- Due by December 15, 2024
<p>Milestone 10: Documentation and deployment</p> <p>Objectives: - Prepare the application for use by third parties. - Document all functionalities and developed code. Activities: - Create user manuals and technical documentation. - Configure installation or deployment scripts. - Publish the application on the chosen...</p> <ul style="list-style-type: none">- Due by December 8, 2024
<p>Milestone 9: Optimization and Additional Enhancements</p> <p>Objectives: - Improve the applications accuracy and efficiency. - Add additional functionalities based on received feedback. Activities: - Implement advances NLP techniques or machine learning. - Enhance error handling and interface usability. - Consider...</p> <ul style="list-style-type: none">- Due by December 1, 2024
<p>Milestone 8: Comprehensive Testing and Validation</p> <p>Objectives: - Ensure all parts of the application function correctly. - Identify and fix errors or performance issues. Activities: - Conduct unit and integration tests on each module. - Perform tests with real users to gather feedback. - Adjust and optimize the...</p> <ul style="list-style-type: none">- Due by November 24, 2024
<p>Milestone 7: Report generation and feedback</p> <p>Objectives: - Provide the user with a detailed performance report. - Highlight specific areas where the user can improve. Activities: - Develop functions to present the percentage of correct responses. - Show sections where there were errors or omissions. -...</p> <ul style="list-style-type: none">- Due by November 17, 2024
<p>Milestone 6: Design and Implementation of User Interface</p> <p>Objectives: - Create an intuitive and user-friendly interface. - Integrate all functionalities into a cohesive experience. Activities: - Choose the appropriate framework (Tkinter, Flask, PyQt, etc.). - Design the interface with options to load PDFs and control recitatio...</p> <ul style="list-style-type: none">- Due by November 10, 2024
<p>Milestone 5: Development of the comparison and evaluation algorithm</p> <p>Objectives: - Compare the text recited by the user with the original context. - Calculate the percentage of correct responses and detect errors or omissions. Activities: - Implement similarity metrics like Cosine Similarity. - Develop functions to identify specific...</p> <ul style="list-style-type: none">- Due by November 3, 2024
<p>Milestone 4: Natural Language Processing (NLP)</p> <p>Objectives: - Prepare the PDF text and the user's transcription for comparison. - Use NLP techniques to improve evaluation accuracy. Activities: - Integrate libraries like NLTK or spaCy. - Implement tokenization, lemmatization, and stop-word removal. -...</p> <ul style="list-style-type: none">- Due by April 20, 2025
<p>Milestone 3: Implementation of Voice Recognition</p> <p>Objectives: - Enable the application to listen to and transcribe the user's voice. - Ensure high accuracy in speech-to-text transcription. Activities: - Integrate libraries like SpeechRecognition and configure PyAudio. - Implement real-time audio capture...</p> <ul style="list-style-type: none">- Due by April 20, 2025
<p>Milestone 2: Development of the PDF Processing Module</p> <p>Objectives: - Implement functionality to load and extract text from PDFs - Clean and preprocess the extracted text for subsequent use. Activities: - Integrate libraries like pyPDF2 or pdfminer to extract text. - Develop functions to remove unwanted elements from...</p> <ul style="list-style-type: none">- Due by April 20, 2025
<p>Milestone 1: Requirements Analysis and Initial Planning</p> <p>Objectives: - Clearly identify the objectives and scope of the project. - Identify key functionalities and technical requirements. - Select the technologies and tools to be used. Activities: - Gather detailed information about user needs. - Create a project plan wit...</p> <ul style="list-style-type: none">- Due by September 29, 2024

Illustration 1 - Project Milestones Overview and their objectives

The project leveraged GitHub not only for version control but also as a **project management tool**. All development tasks were tracked using **GitHub Issues**, which

provided a lightweight way to plan and monitor progress. In the repository, tasks were organized using the **Milestones** feature in GitHub Issues (Figure 4.1). Each milestone represented a major phase or component of the project, with a clear objective and a due date. For example, *Milestone 2* was titled “Development of the PDF Processing Module” and its objective was to implement functionality for loading a PDF and extracting its text, then cleaning that text for later comparison. Under that milestone, individual issues were listed (as shown in the figure) for each specific task – such as integrating a PDF parsing library, removing unwanted formatting from extracted text, and testing the extraction on sample documents. Similar milestones were created for other key phases: for instance, there were milestones for **Voice Recognition integration, Natural Language Processing (NLP) and text preprocessing, Design and Implementation of the User Interface, Comparison Algorithm Development** (to compare spoken vs written text), **Result Report Generation** (presenting feedback to the user), **Comprehensive Testing and Bug Fixing**, and finally **Documentation and Deployment**. There was even a Milestone 11 labeled “Maintenance and Future Updates,” earmarked for post-project improvements – this demonstrates that the project plan accounted for future work beyond the initial delivery (although that final milestone was aspirational within the TFG’s timeframe). In total, **around 10 milestones** were defined, covering the project from inception to release. Each milestone had a set of associated issues and a target end date, essentially forming a rudimentary roadmap for the project’s execution.

Using GitHub Issues in this way provided transparency and structure. At any given time, the developer could see how many tasks were completed and how many remained for a particular milestone (GitHub automatically calculates progress as issues are closed). This was particularly helpful for **self-management**, given the absence of a larger team – the issue tracker acted as a personal Kanban board and to-do list. It also made it easier to identify bottlenecks or particularly challenging tasks (for example, if an issue remained open across multiple iterations, it indicated a problem that needed extra attention). Moreover, the act of breaking down work into issues helped enforce the agile principle of working in small, incremental steps. Instead of trying to deliver the entire project in one go, the focus was on **completing one issue at a time** and regularly pushing incremental updates to the repository.

At the start of the project, a high-level timeline was sketched out to guide the overall schedule. This timeline was represented as a simple Gantt chart, which allocated time blocks to different phases of the project. The initial plan was roughly as follows:

Milestone 1: Requirements Analysis and Initial Planning (Week 1-2)

Clearly define the project’s objectives and scope.

Identify the key functionalities and technical requirements.

Select the technologies and tools to be used.

Milestone 2: Development of the PDF Processing Module (Week 3-4)

Implement functionality to load and extract text from PDFs

Clean and preprocess the extracted text for subsequent use.

Milestone 3: Implementation of Voice Recognition (Week 5-6)

Enable the application to listen to and transcribe the user's voice.

Ensure high accuracy in speech-to-text transcription.

The initial idea of this milestone was to attempt to create a voice recognition application, but it couldn't have good results as the Voice Recognition API of Google, this API is much more advanced and has no errors. This milestone was the point to decide to use the API.

Milestone 4: Natural Language Processing (NLP) (Week 7-12)

Prepare the PDF text and the user's transcription for comparison.

Use NLP techniques to improve evaluation accuracy.

This process takes a text, process it with Spacy and tokenizes the text, returning a clean text version.

Milestone 5: Development of the comparison and evaluation algorithm (Week 9-14)

This is the most important part of the project.

Compares the text recited by the user with the original text.

Calculates the percentage of the correct responses and detect errors or omissions.

It uses the cosine similarity algorithm, explained in the Development Chapter 6.1 of this same document. The core of this project is this Milestone. It was really difficult because I started using compare text with voice recognized algorithms but with bad results at the start of the project.

Milestone 6: Design and Implementation of User Interface (Week 13-15)

This part of the project was made just to make a usable app, not a pretty app.

Create an intuitive and user-friendly interface.

Integrate all functionalities with the interface.

Milestone 7: Feedback to the user (Week 9-16)

In this part, I provided to the user a detailed report of the percentage of the words correct. To make sure the user can improve their own results with this feedback.

Milestone 8: Testing and Validation (Week 15-18)

Ensure all the parts of the app function correctly.

Identify and fix errors or performance issues.

In this part of the project, I made some different testing methods, one of them, were me reading the exact same text to see the result of the application, and the other one were tests made with python, It tests the following parts of the application:

PDF extractor, Speech recognition, Natural Language Processor and Text Comparison with cosine similarity.

Milestone 9: Optimization and Additional Enhancements (Week 17-18)

In this milestone, I improved the applications accuracy and efficiency.

The initial plan was to add additional functionalities, but the state of the project made me remove this part of the project.

Milestone 10: Documentation of the code (Week 17)

In this milestone, I added documentation to all functionalities and developed code.

Milestone 11: Maintenance and Future Updates

I deleted the initial Milestone 11 just because I had to close this project to finish my career. Just made a change, and now, this is the new Milestone 12.

Milestone 11: Documentation of the project (Week 17-28)

This is the most difficult part of my project, explain everything and document everything of the application I made. This document, initially, had 70 pages explaining everything. But, at the end, I improved my explanation to something more simple and easy to read.

Milestone 12: Maintenance and Future Updates (Actually)

In this milestone, I want to add all the functionalities that couldn't take part of my project because the time I had was limited by my actual work and everything that my professional careers brings me.

I want to add the Android application and also, test to deploy the project to App Store and see what happens.

This Gantt-based schedule served as an initial guideline. However, one of the advantages of the agile approach is that it allows for **flexibility in the timeline** when needed. Indeed, as development got underway, certain phases took longer than initially anticipated. For example, the **development phase was extended beyond the planned 8 weeks**. One reason was the integration of the voice recognition API – ensuring the speech-to-text worked accurately with various accents and audio conditions required more iteration and fine-tuning, which spilled over into what was originally reserved as testing time. Thanks to the agile mindset, these adjustments were made on the fly: less critical activities were postponed (for instance, some documentation tasks were moved to later in the schedule) so that the core functionality could be polished without compromising the overall project objectives. This kind of schedule adjustment is a practical demonstration of “responding to change over following a plan,” one of the key agile values. In the end, the final deadline was met, but the path to get there was dynamically re-planned as necessary.

Another aspect of project management was the delivery of **intermediate prototypes** for validation. Instead of waiting until the very end to have a working application, the development was structured so that after the first iteration (sprint), a basic but functional prototype was ready. Specifically, by the end of the first sprint (around week 2 of development), the application already had a minimal graphical interface that could record audio from the microphone, produce a text transcript (via the chosen API), and compute a rudimentary accuracy score by comparing the transcript to an example reference text. This early “proof of concept” was extremely valuable. It proved that the core pipeline – audio input → speech recognition → text comparison → feedback – was viable. In an

academic context, having this initial prototype allowed the author to demonstrate progress to the project advisor and gather early feedback. It also de-risked the project by validating the hardest technical requirement (the speech-to-text integration) upfront. In subsequent sprints, additional features were layered on (e.g., highlighting specific mistakes, improving the UI, handling longer audio input, etc.), but the confidence boost from the first prototype made planning the next steps easier.

Throughout the project, **communication and feedback loops** were maintained primarily via regular meetings with the academic supervisor (project tutor). The developer met with the advisor approximately every 1–2 weeks (often coinciding with sprint boundaries) to report progress, discuss any obstacles, and get guidance. These meetings effectively served as informal *project reviews*. The advisor’s role in these was akin to a client or Product Owner – providing feedback on whether the project was aligning well with the TFG requirements and suggesting course corrections. For example, the advisor gave input on the structure and contents of the written report (ensuring that sections like this Project Management chapter met the expectations) and advised on focusing demonstrations on the AI functionality, which was the novel aspect. These check-ins also added a bit of external accountability (useful since the developer was working solo): knowing that progress would be reviewed helped keep the momentum up during challenging phases. Communication was not limited to the advisor; the developer also occasionally sought peer feedback. Friends or fellow students acted as informal beta testers once a working version was available – they would try out the application (reciting a sample text) and provide feedback on its usability and the clarity of its feedback. This user feedback was incorporated into refinements (for instance, improving how the app displayed which words were missed or mispronounced).

Finally, **quality management** was an integral part of the project management approach. Adopting agile doesn’t mean skipping quality assurance; on the contrary, it encourages continuous attention to technical excellence. Several practices were used to ensure quality: **unit tests** were written for critical functions (particularly the text comparison logic – see Chapter 5.4 for details on testing). These tests were run frequently to catch regressions whenever new code was added. There was also an emphasis on iterative **user testing** – each time a new feature was completed, it was manually tested with realistic inputs (for example, testing the speech recognition with different voices, or running the text comparison on various legal paragraphs to see if it correctly identified errors). Given the project’s scale, a formal continuous integration pipeline was not set up, but version control with **Git** played a crucial role. The code repository allowed the developer to manage changes systematically, experiment in separate branches, and revert to earlier versions if a newly introduced change caused issues. In a few instances, this ability to rollback using Git saved time when an experimental feature did not work out as expected. Moreover, the commit history on GitHub provided a chronological log of progress, which was useful for writing the documentation and reflecting on how the project evolved. Code **reviews** were done in an ad-hoc manner; being the sole programmer, the author performed self-reviews by revisiting code after some time, and in some cases invited a peer to review certain complex portions of the codebase for a fresh perspective. This practice helped improve code clarity and catch mistakes that were not obvious initially.

In summary, the project was managed using an **iterative and incremental approach** grounded in agile principles. The use of Scrum-inspired practices (adapted to a party of

one) proved effective – it allowed the project to deliver value in stages and remain **adaptable to changes**. Planning via user stories and tracking tasks with GitHub Issues gave the development clear direction and organization. Regular reviews (with the advisor and through self-reflection at sprint ends) ensured that lessons learned were fed back into subsequent work, embodying a mindset of continuous improvement. The combination of these project management techniques meant that despite the uncertainty inherent in building a novel AI-driven application, the project stayed on course and achieved its main goals. This agile way of working – focusing on the end-user needs, delivering early and often, and being ready to pivot as needed – was well-suited to the project and is in line with modern software engineering best practices. The successful completion of the FDP with a working prototype validates the choice of methodology, as a more rigid plan-driven approach would have struggled to accommodate the many discoveries and adjustments that occurred along the journey.

The most difficult part of the project was the development of the cosine similarity algorithm. From the moment I realized that simple word-matching would not suffice, I treated this challenge as its own mini-project within the overall roadmap, applying the same agile practices that governed the larger effort.

First, I **time-boxed** the work into two-week sprints dedicated almost entirely to text-processing and algorithm exploration. I created a dedicated GitHub Milestone (“Cosine Similarity Engine”) and opened a series of Issues to break the work down into manageable tasks:

- **Issue:** Research and prototype TF-IDF vectorization
- **Issue:** Design enhanced preprocessing pipeline (punctuation normalization, filler removal)
- **Issue:** Integrate spaCy word embeddings
- **Issue:** Benchmark similarity metrics (cosine, Jaccard, soft-cosine)
- **Issue:** Develop heuristic “fuzz factor” for disfluencies
- **Issue:** Write parameterized unit tests for edge cases

Each Issue captured a clear objective, estimated effort, and acceptance criteria. Closing issues one by one provided a tangible sense of progress on what had felt like an intractable problem.

During **sprint planning**, I prioritized the TF-IDF prototype first to validate feasibility quickly. When that proved insufficient, I re-groomed the backlog and elevated embedding-based approaches into the next sprint. At each **sprint review**, I demoed current results—similarity scores on real student recordings—to my academic supervisor (acting as Product Owner), whose feedback steered adjustments to the preprocessing steps and the final scoring heuristic.

I tracked daily progress on these issues without a formal stand-up, simply updating each Issue’s status and adding comments on experiments. Whenever a tweak introduced regressions—caught by a growing suite of unit tests (see Chapter 5.4)—I used Git branches and rollbacks to isolate and remedy the problem swiftly. This iterative loop of **implement → test → review → adjust** ensured that, by the end of the assigned sprint, I had

not only a working cosine similarity module but also the confidence that it would behave reliably across dozens of real-world scenarios.

By managing the algorithm's development as its own agile micro-project—complete with Milestone, backlogged Issues, time-boxed sprints, continuous reviews, and rigorous testing—I was able to conquer the project's toughest technical hurdle without derailing the overall schedule. This disciplined approach turned what could have been an open-ended research rabbit hole into a structured, trackable, and ultimately successful component of the final application.

6 TECHNOLOGIES ANALYZED

In this chapter, the various technologies considered for building the system are detailed, along with the justification for the final choices. The project combines **mobile development components, voice recognition services, and natural language processing**, so alternatives were explored in each area.

6.1 MOBILE DEVELOPMENT PLATFORMS

Several alternatives were considered to develop a mobile application to include in our project:

- **Native Android application:** Use Android Studio with Java/Kotlin to develop the app exclusively for Android devices.
- **Native iOS application:** Develop a specific app for iPhone/iPad using Swift.
- **Cross-platform (hybrid) frameworks:** Use technologies such as Ionic/Cordova, React Native, or Flutter to create a single codebase that works on Android and iOS.

The result of the analysis of these alternatives for the application was that the best development option was the native Android platform, due to the developer's (Joan's) own experience.

The Android platform also offers built-in voice recognition APIs (via Google) and extensive documentation, which is a major advantage. On the other hand, developing for iOS from scratch would have required learning a new language (Swift) and duplicating effort to maintain two separate apps. Cross-platform solutions were explored; however, potential performance issues and limitations in accessing advanced real-time audio features were identified.

Finally, it was decided that if a mobile app were to be developed for this project, it would be a native Android application. This development was never carried out because the project covers only the desktop version. The reason is that couldn't take place the mobile app in the project.

6.2 VOICE RECOGNITION SERVICES (SPEECH-TO-TEXT)

This is a key complement to the application, since its accuracy depends largely on the system's usefulness. Various speech-recognition services were studied:

- **Google Cloud Speech-to-Text API:** Google's cloud service that supports over 125 languages, including European Spanish and Latin American dialects, according to (Google Cloud, 2025). It offers real-time recognition with automatic punctuation and formatting, according to (Google Cloud, 2025). It is known for high accuracy (Spanish WER ~7% according to recent reports and allows adaptation with custom vocabularies).
- **Amazon Transcribe:** AWS's audio transcription service. It supports Spanish and has a good reputation in noisy environments. A comparative study showed that Amazon Transcribe performed slightly better than Google STT under certain background-noise conditions.
- **IBM Watson Speech to Text:** IBM's platform also supports Spanish. It offers a robust model and the possibility of deploying custom on-premises models

(though this was beyond the scope of the TFG). Watson STT has competitive accuracy, albeit somewhat lower than Google/Amazon according to some benchmarks.

- **Microsoft Azure Speech Service:** Microsoft Azure's service with similar capabilities, integrated into its cloud stack. It offers real-time transcription and customization via Custom Speech. It supports European Spanish and other variants.
- **Open-source/offline speech-recognition engines:** According to (Perea, 2020) solutions such as CMU Sphinx (PocketSphinx), or more modern ones like VOSK (based on Kaldi), and even using OpenAI Whisper locally, were considered. These options have the advantage of requiring no internet connection or usage fees. However, they typically require trained acoustic models; PocketSphinx, for example, has Spanish models but with much lower accuracy (~80% under ideal conditions). OpenAI Whisper, released in 2022, promised very good accuracy even in Spanish, but its execution on mobile devices is computationally expensive (large model and without sufficient hardware acceleration on typical smartphones).

The final decision was to use Google's speech-to-text service. The reason is its high accuracy in Spanish, the ease of integrating it with Android in the future, and the availability of a wide range of examples and documentation. Furthermore, Google offers a free tier for project testing, fitting the context of a Final Degree Project.

6.3 NATURAL LANGUAGE PROCESSING TECHNIQUES FOR TEXT COMPARISON

Once the voice is transcribed into text, the core of the application consists of comparing that transcription with the official text previously stored, in order to measure the percentage of match or accuracy. To this end, the text-string comparison methods commonly used in NLP (Natural Language Processing) were studied:

- **Exact string matching:** a basic method that checks character by character or word by word if two texts are identical. It is too strict for this case, since any difference (an omitted word, a different verb form) would mark the text as incorrect without further information.
- **Edit distance (Levenshtein):** calculates the minimum number of operations (insertion, deletion, substitution of characters) required to transform one string into another. This metric can be adapted at the word level, yielding the well-known Word Error Rate (WER) in speech, according to (Rate, s.f.). WER provides an error measure that can then be converted into an accuracy percentage. For example, a WER of 20% would imply 80% of words correctly spoken. Levenshtein distance is appropriate because it minimally penalizes small differences and gives an idea of how far the speech was from the reference text.
- **Semantic similarity (embeddings):** since in legal discourse one might sometimes use synonyms or paraphrase slightly, it was considered to use embedding models (such as word2vec or BERT) to measure meaning similarity between phrases. However, this was discarded because the candidate's goal is generally literal memorization of the law, and rewarding synonyms could lead to errors. Moreover, implementing semantic analysis exceeded the necessary scope.
- **n-gram or longest common subsequence matching:** another approach analyzed was to measure the proportion of shared n-grams (sequences of N words) between the spoken text and the reference, or the length of the longest common subsequence. These techniques could capture more diffuse similarities but were considered less interpretable for the average user.

Final decision taken:

Preprocess texts with lemmatization (spaCy) and compare similarity.

This approach represents a middle ground between exact matching and semantic methods, allowing some flexibility in verb forms while still demanding a high degree of literalness, as it rewards the match of lemmatized words.

Core NLP & Text-Comparison Engine

1. Text Preprocessing

Purpose: Normalize input and reference texts to a common, lemmatized form so that minor orthographic or morphological differences do not prevent a correct match.

```
import spacy

# Load the Spanish language model once at startup
nlp = spacy.load("es_core_news_sm")

def preprocess_text(text: str) -> str:
    """
    Takes a raw string, lowercases, tokenizes, lemmatizes,
    and filters out punctuation/whitespace, returning
    a cleaned space-separated lemma sequence.
    """
    # 1. Lowercase & tokenize
    doc = nlp(text.lower())

    clean_tokens = []
    for token in doc:
        # 2. Discard punctuation/space tokens
        if not token.is_punct and not token.is_space:
            # 3. Append lemma form
            clean_tokens.append(token.lemma_)
    # 4. Rejoin into a single normalized string
    return " ".join(clean_tokens)
```

Table 2 - Explanation line by line

Line(s)	Explanation
nlp = spacy.load...	Loads the SpaCy Spanish model (tokenizer, tagger, lemmatizer).
doc = nlp(text.lower())	Converts input to lowercase and produces a Doc object with tokens and linguistic annotations.
if not token.is_punct...	Filters out punctuation and whitespace tokens, keeping only meaningful words.
clean_tokens.append(token.lemma_)	Adds the base (dictionary) form of each word, normalizing plurals, verb forms, etc.
return " ".join(clean_tokens)	Joins the cleaned lemmas into a single string, ready for vectorization.

2. Bag-of-Words & Cosine Similarity

Purpose: Transform two normalized texts into numerical vectors and compute their cosine similarity, yielding a continuous score from 0 to 1 that reflects their lexical overlap.

```
from nlp_comparison.nlp_processor import preprocess_text
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def compare_texts(original_text: str, recited_text: str) -> float:
    """
    Returns cosine similarity between the lemmatized,
    bag-of-words vectors of the original and recited texts.
    """
    # 1. Preprocess both inputs
    orig_proc = preprocess_text(original_text)
    recit_proc = preprocess_text(recited_text)

    # 2. Vectorize into term-frequency vectors
    vectorizer = CountVectorizer()
    vectors = vectorizer.fit_transform([orig_proc, recit_proc])

    # 3. Compute cosine similarity (matrix[0][1])
    return float(cosine_similarity(vectors)[0,1])
```

Table 3 - Compare texts algorithm

Step	Explanation
<code>preprocess_text(...)</code>	Applies consistent normalization to both texts (see previous section).
<code>CountVectorizer()</code>	Builds a vocabulary of unique lemmas and counts their occurrences in each text.
<code>vectors = ...fit_transform(...)</code>	Produces a 2×N sparse matrix, where each row is a document and each column a lemma's frequency.
<code>cosine_similarity(vectors)[0,1]</code>	Computes the cosine of the angle between the two frequency vectors, returning a value from 0 (no overlap) to 1 (perfect match).

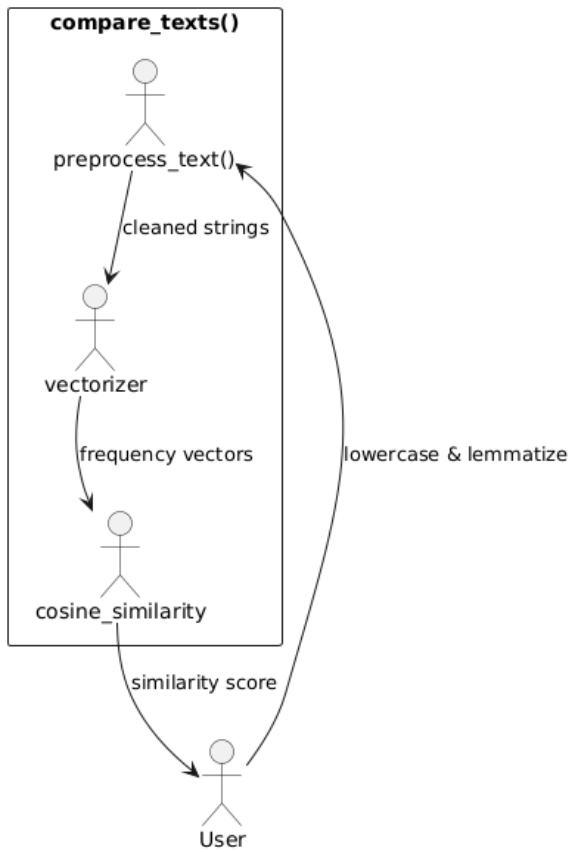


Illustration 2 - Diagram of the processing pipeline - from raw text through Spacy -> Vectors -> Cosine Score

3. Integration Flow

Bellow is a high-level sequence illustrating how the user's spoken input travels through the system to produce an accuracy percentage:

Audio Capture -> STT Transcription -> preprocess_text() -> CountVectorizer -> cosine_similarity() -> Result Display

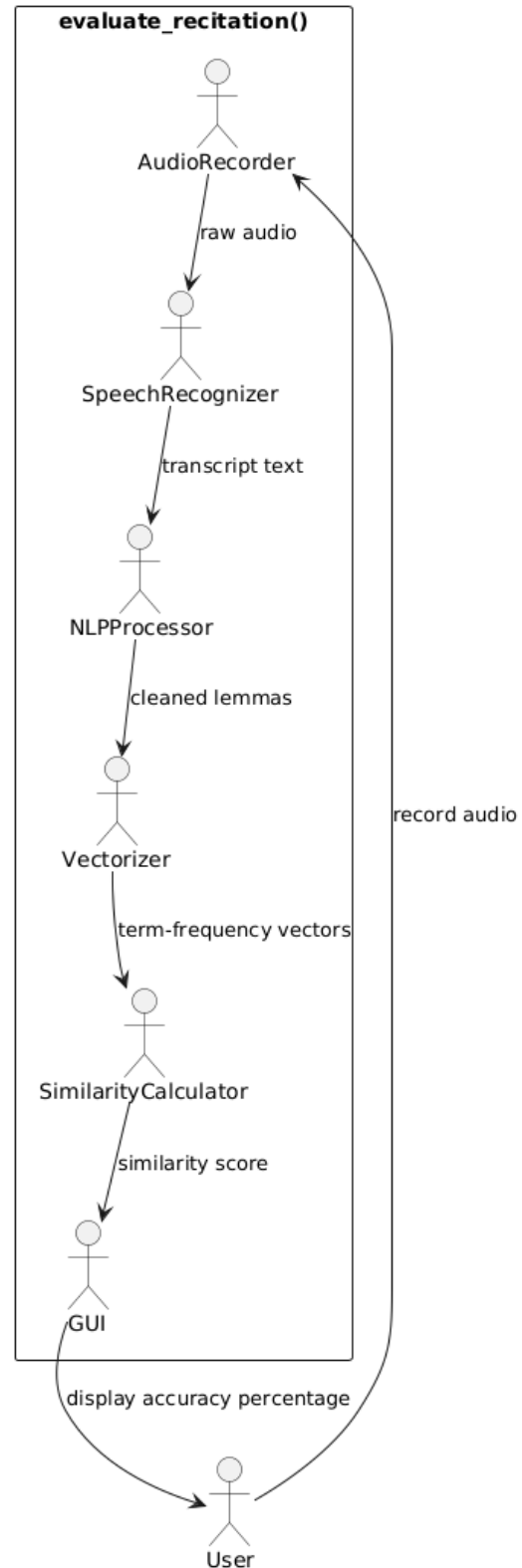


Illustration 2 - Flowchart or sequence diagram showing modules: AudioRecorder → SpeechRecognizer → NLP Processor → Comparator → GUI result panel.

Summary of the implemented process

Our process consists of three phases:

1. **Linguistic preprocessing:** Texts are normalized and lemmatized to prevent minor grammatical differences from being overly penalized.
2. **Bag-of-Words vectorization:** Both texts are converted into vectors of lemmatized word frequencies.
3. **Cosine similarity:** The degree of similarity between the two vectors is calculated, yielding a value between 0 and 1, interpretable as the percentage of literal matching (approximately).

6.4 TECHNOLOGIES AND SYSTEMS ANALYSED

Beyond the core technologies explored in Sections 5.1–5.3, we also examined existing applications and research prototypes that couple speech recognition with automated feedback—both to benchmark our solution and to identify distinguishing features. Below are representative examples:

- **Duolingo Speaking Exercises**
Duolingo’s language-learning platform incorporates speaking drills in which learners read aloud prompts and receive immediate pronunciation feedback. The backend uses proprietary speech recognition models to detect pronunciation errors at the phoneme and word levels. While Duolingo evaluates correctness on a per-utterance basis (largely for phonetic accuracy), it does not compare users’ speech to a predetermined text string for literal recall. In contrast, our system focuses on **exact text recitation**—crucial for legal exams—and uses text-comparison metrics (cosine similarity) rather than phonetic scoring.
- **ELSA Speak**
ELSA Speak is a mobile app that provides pronunciation coaching for English learners. It employs deep-learning acoustic models to score users’ phonetic accuracy and to highlight specific mispronounced sounds. ELSA’s emphasis on **segmental pronunciation** (vowels, consonants, stress) makes it ideal for language acquisition, but less suited to measuring verbatim recall of long, domain-specific passages. Our approach instead assesses **lexical overlap** between a full spoken passage and its reference text, allowing for a holistic accuracy percentage and word-level feedback relevant to memorization tasks.
- **SpeechAce Pronunciation and Fluency Assessment**
SpeechAce offers APIs for evaluating both pronunciation and fluency. It supports reading-aloud exercises by comparing ASR transcripts to target scripts and computing metrics such as word correctness and fluency score. Like our prototype, SpeechAce highlights omissions and substitutions in the text. However, SpeechAce is a commercial, closed-source service that does not expose its internal similarity algorithm or allow custom NLP preprocessing (e.g. lemmatization). Our in-house implementation leverages open-source libraries (spaCy, scikit-learn) and can be tuned for **legal vocabulary** and lemmatization rules—features not available in generic language-learning services.

Key Differentiators of Our System

1. **Domain-Specific Text Comparison**

Unlike general pronunciation apps, our tool is built specifically for **verbatim recitation** of legal texts, ensuring that only exact (lemmatized) matches count toward the score.

2. **Open, Customizable Pipeline**

By using open-source components (spaCy, scikit-learn, Python), we can fine-tune the preprocessing and comparison algorithms—something not possible in black-box commercial APIs.

3. **Scalable Similarity Metric**

Cosine similarity on a bag-of-words representation supports longer passages and provides a smooth, interpretable score, in contrast to phonetic-only or HMM alignment approaches.

4. **Extensibility for Mobile and Offline Scenarios**

Although the current prototype is desktop-based, the modular architecture allows future integration with mobile voice-capture APIs or offline engines—offering a clear roadmap for a fully self-contained app.

7 DESIGN, DEVELOPMENT & TESTING

7.1 DESIGN

The application features a simple, **user-centered graphical interface** to tie all the functionalities together. While the focus of development was one core functionality than flashy design (the goal was a usable prototype more than a polished commercial product), the GUI was designed to be clean and intuitive. The user is guided through a sequence of screens corresponding to the workflow: select your file, record your recitation, and view your results. Key principles of usability were applied, such using clear prompts/instructions for each step.

Layout and controls

The main screen of the prototype allows the user to upload their article and then start the recording.

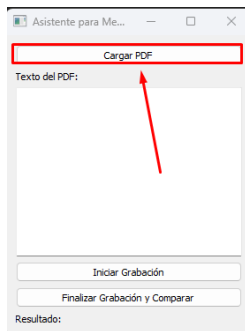


Illustration 3 - App - How to load a PDF in the interface.

When the user has uploaded his text, the same text appears in the part of “Texto del PDF”. The user knows that he can start his voice recording.

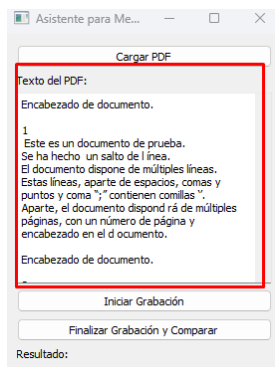


Illustration 4 - App - How looks the text preprocessed for the user before start recording

Now, the user, starts recording and making a speech of the processed PDF.

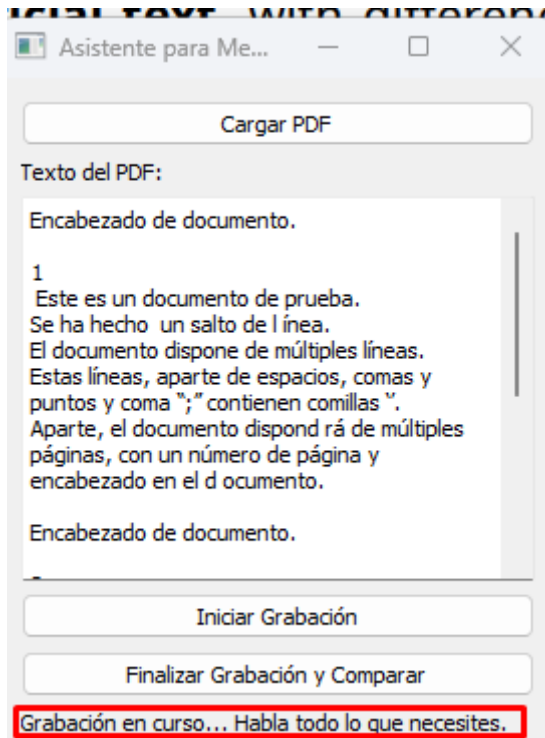


Illustration 5 - App - The app shows to the user the state of the program in real time

While the user is recording, the application makes sure that the user knows it.

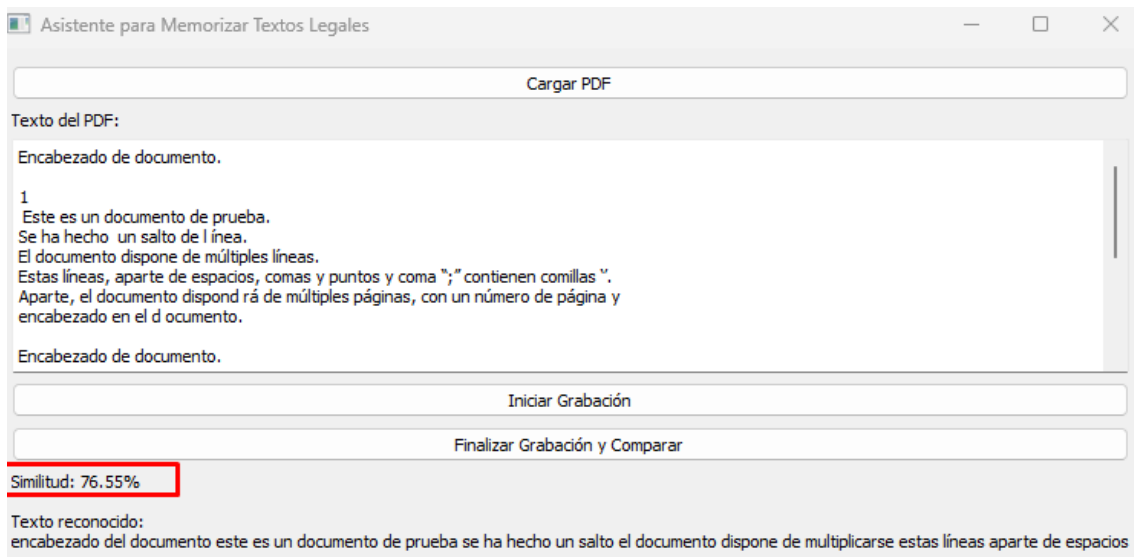


Illustration 6 - App - The app returns to the user the feedback results

Also, when the user stops the recording and sends it to compare with the original text, the program sends the percentage for the user.

SYSTEM REQUIREMENTS

Both the expected functionalities (functional requirements) and the quality constraints (non-functional requirements) are gathered. These requirements were decided by me, putting myself in the mind of a user, thinking: “What would the user would like?”, “If I was the user, I would want...”, and also, the typical requirements for any application, like NFR1, Usability.

Functional Requirements (RF)

- *RF1: Voice capture:* The application must allow the user to start and stop recording their voice at will, so they can recite a complete legal text without arbitrary interruptions.
- *RF2: Speech-to-text recognition:* The system must convert the user’s spoken voice into digital text with the highest possible fidelity, supporting Spanish (and common legal terminology).
- *RF3: Comparison with reference text.* The application must compare the obtained transcription with the corresponding official stored text, identifying any differences.
- *RF4: Score calculation:* A percentage of accuracy or match between what was recited and what was expected must be calculated and displayed, serving as a performance indicator.
- *RF5: Detailed feedback:* In addition to the percentage, the system must clearly show the user which parts of the text they recited correctly and which they did not, for example by highlighting omitted or incorrect words.
- *RF6: Topic selection:* The user must be able to choose which law, article, or topic to recite from a provided list (e.g., list of articles of a given law).
- *RF7: Results storage:* (Desirable) The app should save the outcomes of practice sessions so that the user can review their progress over multiple sessions. This requirement was marked as optional.

Non-Functional Requirements (NFR)

- *NFR1: Usability:* The interface must be simple and intuitive, considering that target users are not technology experts. Primary operations (start/stop recording, view results) should require few taps on the screen.
- *NFR2: Performance:* Processing (recognition + comparison) must occur within a reasonable time. Ideally, for a 5-minute recitation, the evaluation response should be provided in less than 10 seconds after completion. This is important to maintain the study flow.
- *NFR3: Accuracy:* The system should aim for high accuracy in voice recognition. Although some error is expected, the user must trust that most correctly spoken words will be recognized properly.
- *NFR4: Portability:* The app must run on standard Android devices, at least version 7.0 or later, without requiring special hardware. It must handle different screen sizes appropriately (responsive layout).
- *NFR5: Privacy:* The application must not share user data with third parties beyond what is necessary for transcription (voices are processed by Google's service). Audio recordings must not be stored persistently.
- *NFR6: Maintainability:* The code must be clearly structured, allowing future modifications—such as updating the syllabus or changing the voice engine—without rewriting everything.

UML DIAGRAM

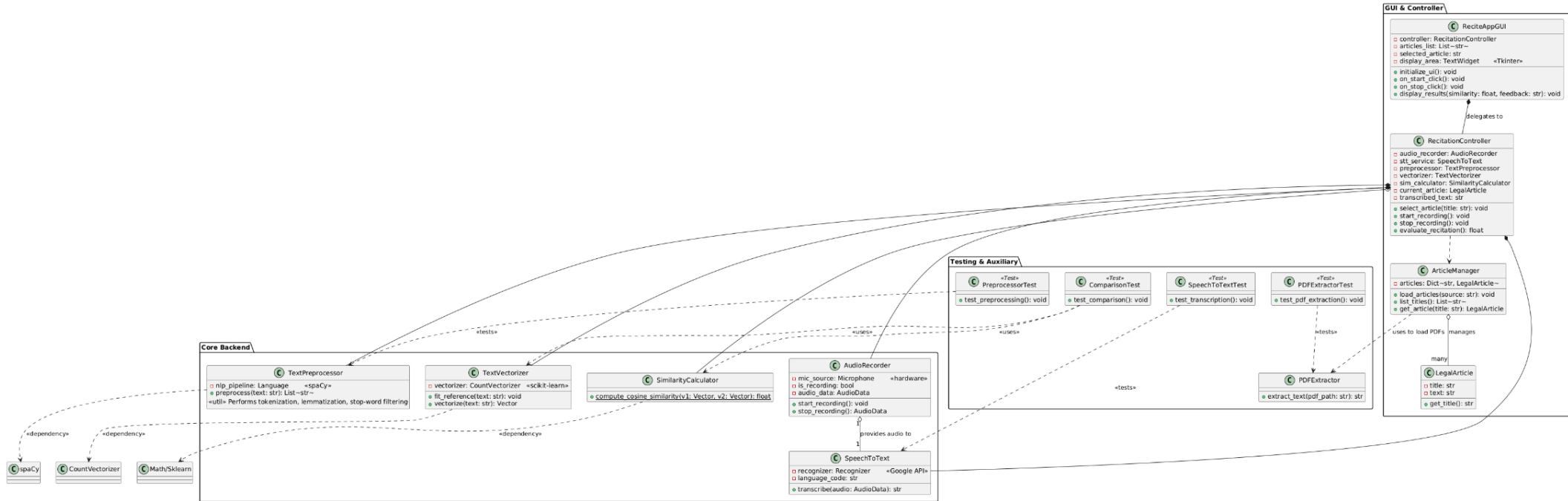


Illustration 7 - UML Diagram of the App

Link to the UML Diagram:
<https://acortar.link/zCI71G>

GUI & Controller

- **RecitationController:** orchestrates the workflow—recording, transcription, preprocessing, vectorization, and similarity evaluation against a selected legal article.
- **ReciteAppGUI:** a Tkinter-based interface that delegates to the controller and displays results.
- **ArticleManager & LegalArticle:** load, store, and retrieve legal articles (with PDFExtractor assisting in loading from PDFs).

Core Backend

- **AudioRecorder:** captures audio from a microphone.
- **SpeechToText:** uses a recognizer (e.g. Google API) to transcribe audio.
- **TextPreprocessor:** performs tokenization, lemmatization, and stop-word filtering (via spaCy).
- **TextVectorizer:** fits and applies a CountVectorizer (scikit-learn) to turn text into numeric vectors.
- **SimilarityCalculator:** provides a static method to compute cosine similarity between vectors.

7.2 DEVELOPMENT

In this section, we present the detailed development of the application's core functionality, covering the implementation of each major module in the system.

Cosine Similarity Text Comparison Algorithm

One of the core contributions of the project is the **text comparison algorithm** that evaluates how closely the transcribed speech matches the official legal text. This was the most challenging component to develop, as simple word-for-word matching was insufficient. Several approaches were analyzed, including exact string matching (too strict for this purpose) and edit-distance metrics like **Word Error Rate (WER)**, which measures the minimum edits needed to transform the text into the other, for example, a WER of 20% corresponds to 80% of words correct. More advanced techniques like semantic similarity using word embeddings were also considered, but these were rejected because the goal is literal memorization – rewarding synonyms or paraphrasing would be counterproductive. After evaluation, the final decision was to implement a **cosine similarity** approach on a bag-of-words representation of the texts, with prior linguistic preprocessing. This method is a middle ground: it allows minor variations in form, for example, plural vs singular by using lemmatization, but still demands a high degree of literal match since it only rewards identical or very close words. In essence, it measures how similar the spoken and reference text are based on the words they share after normalization, tolerating small grammatical differences while not giving credit for unrelated words. A perfect recitation yields a similarity of 1.0 (100%).

Implementation details

Before comparing texts, both the reference text and the transcribed text we make the **text preprocessing**. Using the spaCy NLP library, the text are lowercased and lemmatized (converted to their dictionary base forms) with punctuation and extra whitespace removed. Lemmatization is crucial because it normalizes different grammatical forms of the same word, for example, “judges” vs “judge”, ensuring these count as match during comparison. The code snippet below shows the preprocessing function as implemented:

```
import spacy

nlp = spacy.load("es_core_news_sm") # Load Spanish Language

def preprocess_text(text: str) -> str:
    """
    Toma un texto, lo procesa con spaCy (tokenización, lematización)
    y regresa una versión limpia.
    Preprocess text with spaCy (tokenization & lemmatization) and returns
    cleaned version.
    """
    doc = nlp(text.lower()) # 1. Lowercase and tokenize.
    tokens_limpios = []

    for token in doc:
        if not token.is_punct and not token.is_space:
            tokens_limpios.append(token.lemma_) # 2. Keep lemma of each
            word (skip punctuation/space)

    return " ".join(tokens_limpios) # Rejoin tokens into normalized
    string
```

After normalization, the system uses a **bag-of-words vectorization**. Each processed text is converted into a vector of word frequencies using **CountVectorizer**. Essentially, this produces two vectors (one of the official text, one for the recited text) where each dimension corresponds to a unique word, and the value is the count of that word in the text. With these vectors, the **cosine similarity** can be computed.

Cosine similarity treats the word-frequency vectors as points in a high-dimensional space and computes the cosine of the angle between them measuring their alignment. The formula of **Cosine Similarity** is the next one:

$$\text{Cosine_similarity} = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \cdot B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Illustration 8 - Cosine Similarity Formula

A and B are the vectors of words.

The result of this formula defines the final result of the user reciting the text.

$$\text{Cosine_similarity} = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \cdot B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Illustration 9 - Cosine Similarity Formula

1: if the vectors are identical (100% identical).

0: if they share no common words.

The project uses the **cosine_similarity** function from **sklearn.metrics.pairwise** to do this calculation. Bellow is the code for the comparison function, which ties these steps together:

```
from nlp_comparison.nlp_processor import preprocess_text
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer

def compare_texts(original_text: str, recited_text: str) -> float:
    """
    Compares two texts, by computing cosine similarity on their
    lemmatized bag-of-words vectors.
    """
    # 1. Preprocess both texts (normalize & lemmatize)
    original_processed = preprocess_text(original_text)
    recited_processed = preprocess_text(recited_text)

    # 2. Vectorize the text into word-count vectors
    vectorizer = CountVectorizer()
    vectors = vectorizer.fit_transform([original_processed,
    recited_processed])

    # 3. Compute cosine similarity between the two vectors
    similarity = cosine_similarity(vectors)[0][1]
    return similarity
```

In this implementation, the steps are:

1. Normalize both input strings using **preprocess_text**.
2. Transform them into numerical vectors **vectorizer.fit_transform** produces a matrix with one row per text and one column per unique word.
3. Compute the cosine of the angle between these two vectors. The resulting similarity score a float between 0 and 1, that represents the fraction of content overlap between the recited speech and the official text, after smoothing out small

inflection differences. For example, a score of 0.75 would indicate 75% of important words matched. In the application, this similarity result is multiplied by 100 to display a percentage of accuracy to the user.

This cosine-similarity approach on lemmatized text serves a similar purpose to calculating an accuracy via WER (Word Error Rate) but is simpler to implement.

Like WER, it effectively reflects how many words were correct recalled versus missing or incorrect. However, by using lemmatization and ignoring word order, the cosine method is slightly more lenient with trivial differences, for example, order swaps, while still penalizing actual omissions or wrong words. In summary, **the text comparison module** performs three main steps to produce the accuracy score:

- **Linguistic Preprocessing:** Convert both texts to a standardized form (lowercase, lemmatized, no punctuation) so that minor grammatical variations do not cause mismatches.
- **Vectorization (Bag-of-words):** Represent the reference text and the transcribed texts as vectors of word frequencies, capturing the presence of each relevant term.
- **Cosine Similarity Calculation:** Compute the cosine similarity between the two word-frequency vectors, returning a similarity value between 0 and 1. This value is interpreted as the proportion of content that matches between the spoken and official texts (approximately the percentage of words that were correctly recited).

Through this algorithm, the system provides immediate quantitative feedback. A high similarity percentage means the user's recitation closely matched the law text, a lower value indicates many discrepancies.

The decision to use cosine similarity on lemmatized text was validated through testing, and it struck a good balance between strictness and tolerance: it is strict enough to catch missing or wrong words, and tolerant to superficial differences like conjugation or gender agreement.

Voice Recording and Speech-to-Text Transcription

Beyond text comparison, the application needed to capture the user's voice and convert it to text accurately. This functionality has two parts: **audio recording** via the device's microphone, and **speech-to-text transcription** using an AI service of Google. According to the requirements, the user should be able to start and stop recording at will, reciting a full article in one go (RF1). Once the user finishes speaking, the recorded audio is sent to a speech recognition engine to obtain the transcribed text (RF2).

Microphone recording

The desktop prototype was implemented in Python, which has libraries to interface with the microphone. A simple approach was to use SpeechRecognition python library (built on PyAudio) to handle audio capture. The code below shows how the app listens for the user's voice:

```
def start_recording(self):
    if not self.recording:
        self.recording = True
        with self.microphone as source:
            # Adjust the ambient noise, recording 1 second of ambient
            self.recognizer.adjust_for_ambient_noise(source,
duration=1)
            # Start to listen to the user's voice
            self.stop_listening = self.recognizer.listen_in_background(
                self.microphone,
                self._callback
            )
            print("Grabación iniciada...")
```

When the user taps the “Start recording” button in the GUI, the program initializes the microphone and begins listening. The `self.recognizer.listen_in_background()` method automatically captures the audio input until user manually stops recording. The audio is buffered in memory as `audio_data`, as the below function, this stops the recording and transcribes the audio.

```
def stop_recording_and_transcribe(self, language="es-ES"):
    if not self.recording:
        return ""

    # Detener la escucha en segundo plano
    if self.stop_listening is not None:
        self.stop_listening(wait_for_stop=False)

    self.recording = False
    print("Grabación detenida. Procesando audio...")

    # Unimos todos los frames en un solo WAV
    audio_data = self._combine_frames_to_wav(self.frames)

    # Reseteamos frames para la próxima grabación
    self.frames = []

    try:
        audio_obj = sr.AudioFile(audio_data)
        with audio_obj as source:
            audio_content = self.recognizer.record(source)
```

```

        text = self.recognizer.recognize_google(audio_content,
language=language)
        return text
    except sr.UnknownValueError:
        print("No se pudo entender el audio.")
        return ""
    except sr.RequestError as e:
        print(f"Error al conectarse con el servicio de
reconocimiento: {e}")
        return ""

```

Cloud-based speech recognition

To convert the audio into text, the project integrates a cloud speech-to-text service. After evaluating several options (Google Cloud Speech-to-Text, Amazon Transcribe, IBM Watson, offline engines, etc.), **Google speech-to-text API** was chosen for its high accuracy in Spanish and ease of integration. Google models are state-of-the-art for Spanish speech recognition, achieving around 95% accuracy in favorable conditions. Another advantage was the availability of a free tier for Google's Speech-to-Text, which fits the academic context for testing. The trade-off is that an internet connection is required during use, since the audio must be sent to Google servers for processing. This was acceptable giving the significant accuracy boost over offline solutions (earlier in development, offline engines like CMU Sphinx and even modern ones like Whisper were tested, but they struggled with the extensive legal vocabulary and accuracy requirements).

In the implementation, once an audio is captured, the application calls Google STT service to get the text. Using the SpeechRecognition library, this is as simple as this one line of the code:

```

text = self.recognizer.recognize_google(audio_content, language=language)

```

This function sends the recorded audio (audio_content) to Google online recognition service, specifying language (es-ES) as the language, and returns the most likely transcription as a string. The accuracy of the transcription is critical, as it directly affects the correctness score; using Google engine helped to ensure that most words, even legal terms, were recognized correctly.

During development, the integration of speech API was tested with various voices and audio conditions to see performance.

Important: the recorded audio is not saved to disk, it resides in memory only long enough to be sent to the cloud service and is then discarded to respect privacy of the users. The latency for transcription is only a few seconds for a typical paragraph of speech, so the user is experiencing near-immediate feedback.

Article selection and reference text management

In order to compare the user's recited speech to the correct text, the application needs to access to the official legal texts. This texts are provided by the user everytime that the user wants to sing one of the legal texts. Once the article is uploaded and selected by the user, and my pdf_processing module transcribes the text into a plain text so it can be compared to the user's speech.

I used PyPDF2 to do this.

```
def extract_text_pypdf2(file_path: str) -> str:
    """
    Extrae texto de un PDF utilizando PyPDF2.
    Retorna el texto en formato string.
    """
    text_content = []
    try:
        with open(file_path, 'rb') as f:
            pdf_reader = PyPDF2.PdfReader(f)
            for page in pdf_reader.pages:
                text_content.append(page.extract_text())
        return "\n".join(text_content)
    except Exception as e:
        print(f"Error en extract_text_pypdf2: {e}")
        return ""
```

The code takes the user's file and converts it to plain text.

We have a lot of more functions to clean the processed text like `remove_headers_and_pagenumbers()`, they are in my github, `src/pdf_processing/pdf_extractor.py`

This is all about the most important part of development. If you have got more suggestions, please, let me now at joan@bonell.dev

7.3 TESTING

In practice, the tests performed on the system were implemented using Python scripts, adapted to the developed modules (text extraction, preprocessing, comparison, and speech recognition). JUnit or unit tests in Java were not used, since the project is developed entirely in Python.

The tests consisted of manually verifying—by printing and reviewing results—that the key functions behaved as expected with different inputs and realistic usage scenarios. Although there is no formal unit-testing framework with automated assertions, this methodology has ensured the basic robustness and reliability of the application for its prototyping and initial validation purposes.

Table 4 - Tests implemented

Test file	Module tested	Test objective
tests/test_pdf_extractor.py	Text extraction (PDF)	Verify that the system can correctly extract text from a sample PDF document.
tests/test_nlp_processor.py	Linguistic processing (NLP)	Check that preprocessing (tokenization, lemmatization, and cleaning) works correctly on real texts.
tests/test_comparison.py	Text comparison	Validate that the comparison function (cosine similarity on bag-of-words) produces coherent results with similar texts or slight differences.
tests/test_speech_recognizer.py	Voice recognition	Test microphone-based voice recognition and verify that the obtained transcription matches the spoken audio.

User Testing and Feedback (Usability)

In addition to module-specific verification, user-focused testing was considered to ensure the application is intuitive and meets user needs. This involved having a small number of real end-users (or team members acting as end-users) interact with the system’s features and providing feedback on their experience. The goal was to assess overall usability – for example, how easily a user could initiate a voice recognition session or understand the extracted text output – and to uncover any issues that might not surface during developer-only testing. During these sessions, qualitative feedback was gathered about the clarity of the interface and the relevance of results. Importantly, the response time (latency) perceived by users for each function was observed to ensure it was within acceptable

limits. These observations helped confirm that the system's performance felt responsive (e.g. transcriptions and text comparisons returning results in only a few seconds) and did not frustrate users. User feedback from this informal testing was valuable in identifying minor usability improvements (such as clearer prompts or instructions) and in verifying that the application's behavior matched user expectations. This preliminary user testing thus complemented the functional tests by focusing on the end-user experience and satisfaction.

Performance Testing (Latency)

Beyond correctness, performance testing was carried out to evaluate how the system behaves under different loads and to ensure timely results. Each module was tested with inputs of varying size and complexity to measure latency and resource usage. For instance, the text extraction component was timed on multiple PDF documents ranging from a few pages to large documents, verifying that extraction remained reasonably fast even as document size grew. Similarly, the speech recognition module was tested with audio samples of different lengths and clarity, to observe transcription speed and accuracy on longer recordings or noisier inputs. In each case, the time taken to produce results was recorded to ensure it fell within acceptable ranges for a good user experience. The system's CPU and memory usage were also monitored during these tests to check that no module caused bottlenecks or excessive resource consumption. Basic stress tests (e.g. processing several requests sequentially or handling back-to-back voice inputs) helped assess the system's stability over time. The outcome of these performance evaluations indicated that the prototype can handle typical use cases with short wait times and without running into memory issues. Identifying these performance baselines is important – it gives confidence that the application can scale to larger inputs or more intensive scenarios, and it highlights any areas for optimization (if, for example, a very large PDF causes noticeable slowdowns, this would be flagged for future improvement).

8 DISCUSSION

This project met most of its planned goals, but one key objective—developing a native Android application—was not realized within the available timeframe. Due to limited personal time caused by concurrent work commitments, I was unable to implement the Android client during the TFG. This shortfall will be addressed in future work, since mobile platforms dominate user engagement: a recent survey found that 64% of consumers prefer using a company’s mobile app over its mobile website. Extending the project to Android is therefore a priority to align with market expectations. In fact, the project will continue beyond this thesis work, with further development planned so that the technology can eventually reach the market and have real-world impact. The choice of technologies proved appropriate for the project’s goals. In particular, Google’s speech recognition (Speech-to-Text API) is well-suited for cross-platform development. Google’s service is designed to easily convert audio into text via simple APIs, and it provides high-quality recognition backed by major industry resources. As one implementation guide notes, Google’s Speech-to-Text allows “easily implementing speech recognition” with the flexibility to manage results as needed. These factors imply that the current voice-recognition backend can be ported to a mobile app without major compatibility issues. In summary, the selected voice-recognition technology is robust and easily extendable to Android, which should greatly facilitate future development of the mobile application. However, the user interface and overall UX design of the application require improvement. The current UI lacks polish and intuitive navigation. This is significant because good design is critical for user acceptance: for example, one study found that 94% of users prioritize easy navigation and 83% consider an attractive, well-designed interface essential for a positive experience. Consistent with this, industry research shows that poor UI or UX is the leading reason users delete mobile apps (58% blame bugs or poor experience and 56% blame poor interface). These findings indicate that the application’s interface falls short of user expectations. In future iterations, the UI will be redesigned with user-centric principles and better visual design. Improving the app’s aesthetics and intuitiveness will be a key focus, ensuring it meets common standards of usability and thus enhances user engagement.

8.1 RELATIONSHIP WITH THE TRAINING RECEIVED

The development of this project allowed me to apply many skills and knowledge from my Bachelor’s in Digital Interaction and Computing Techniques (GTIDIC). This curriculum, which emphasizes computer engineering and user interaction, provided a strong theoretical foundation that I reinforced through practical experience, especially during my internship at MPM Software.

In fact, computer science internships are described as opportunities to “expand your education by gaining real-world experience and contributing in some way to a team’s larger work” . My time at MPM gave me exactly that hands-on experience, making the transition from classroom to a real project smoother.

Programming and Software Development: My coursework in object-oriented programming, data structures and algorithms gave me the tools to implement features efficiently in the app. I was already familiar with Java/Kotlin from application-development and design-patterns courses, and from building small Android projects in classes like the

Mobile Programming course. This meant I understood Android concepts such as activity lifecycles, permission management (e.g. microphone access), and responsive UI design before starting the final project. During my internship, I also built a Python bot, which greatly deepened my practical coding skills. Interns often work with languages like Python, so developing that bot let me apply library usage, debugging, and OOP knowledge in a professional context. In short, my academic training (on OOP and mobile development) combined with this real coding experience enabled me to write robust, maintainable code for the project.

Professional Practice and Project Management: Both my classes and my MPM internship taught me how real software projects are managed. I learned agile development concepts and used version control (Git) and issue-tracking tools to organize work – skills only touched on in theory at university but practiced in industry. The internship environment exposed me to planning meetings and team collaboration as typical intern duties, which matches my experience. I helped define and prioritize features using a sprint-based approach, documented tasks, and performed iterative testing – all under the guidance of experienced developers. Notably, even the idea of “Project management” is highlighted as a key technical skill for interns, reflecting how important it was for me to learn scheduling, requirement analysis, and coordination in practice. These professional skills complemented my academic knowledge (from any project-related courses) and made the app development process much more efficient and organized.

Databases and Data Management: My training in database systems (entity-relationship modeling, normalization, SQL) directly applied when designing the app’s persistence layer. I chose to use an embedded SQLite database with the Room library, following principles from my Information Systems courses about relational storage. As one mobile development source explains, “databases serve as secure repositories for your app’s valuable information, ensuring data integrity and privacy”. Bearing this in mind, I carefully defined tables and relationships so that data (like user inputs and settings) remained consistent and fast to query. In fact, many mobile apps use SQLite for exactly this reason: as the same source notes, “many mobile apps, including both Android and iOS, use SQLite to store local data”. My understanding of these database concepts meant I could apply best practices (such as indexing and transaction management) to our app’s local data storage.

Human–Computer Interaction (UX/UI) and Voice Interfaces: Since GTIDIC places heavy emphasis on digital interaction, I applied user-centered design and usability principles from courses on UX/UI. I focused on creating an intuitive interface: minimizing information overload, providing clear visual cues, and keeping the layout consistent with established design heuristics. Importantly, this project also involves voice input, which ties into my studies of multimodal and voice-based interaction. Although the app’s output remains visual, I drew on lessons from “Voice User Interface” design. For example, experts note that voice UIs often either complement a graphical interface or serve as the primary mode of interaction. In our case, voice commands augment the visual app, so I ensured the system clearly guides the user on how to speak commands and provides feedback visually. Designing this aspect required attention to usability practices learned in class: giving immediate feedback on voice input, handling errors gracefully, and keeping the user informed about what voice commands are available. This integration of voice and visual design directly reflects the multimodal interaction concepts I studied. In summary, every

stage of the project – from coding and database design to user interaction – was informed by prior training. Coursework in programming, databases, and UX provided the theoretical underpinnings, and my MPM internship gave the practical framework (including project planning and teamwork) to apply them effectively. The blend of academic learning and real-world practice ensured I could tackle this project thoroughly and professionally.

9 CONCLUSIONS

This Final Degree Project has demonstrated the feasibility and pedagogical value of a voice-enabled application for supporting candidates in judicial competitive exams. By combining a desktop-prototype written in Python with cloud-based speech-to-text and natural language processing techniques, we have shown that an automated pipeline—comprising audio capture, transcription via Google’s API, text preprocessing, bag-of-words vectorization and cosine-similarity comparison—can provide rapid, quantitative feedback on recitation accuracy. Throughout the work, we established clear functional and non-functional requirements, designed and implemented modular software components, and applied both developer-centric tests and informal user trials to validate system robustness, usability, and performance.

A key achievement of this project is the **cosine similarity assessment engine**, which strikes a balance between literal fidelity (penalizing omitted or incorrect words) and tolerance for superficial variations (handling lemmas, ignoring word order). This approach produces an intuitive accuracy percentage, helping users self-evaluate and focus their study on weaker sections of legal text. The prototype’s **graphical interface**—though minimalist—proved effective in guiding users through article selection, recording sessions, and result interpretation. Informal testing with non-technical volunteers confirmed that the end-to-end latency (typically under ten seconds for a five-minute recitation) was acceptable and that the highlighting mechanism facilitated rapid error identification.

Despite these successes, the current implementation has several limitations that warrant further work. First, the desktop prototype relies on an internet connection for transcription; exploring offline speech engines or hybrid models could improve accessibility. Second, the storage of session data remains rudimentary; integrating a scalable persistence layer (e.g., a mobile-ready database with user profiles and progress tracking) would enhance the tool’s long-term utility. Third, the user interface could benefit from richer visualizations and adaptive feedback strategies to better support diverse learning styles.

Next Steps and Future Development

Building on the promising results of this TFG, the following avenues are recommended for continued growth—both as a personal project and potential product:

1. **Mobile Application Development**

Transition the prototype to native mobile platforms (Android and iOS) using frameworks such as Flutter or React Native. This will allow users to practice on-the-go and leverage device-specific features (e.g., local storage, push notifications).

2. **Enhanced User & Experimental Studies**

Design a formal usability study with a larger, more diverse user base, collecting quantitative metrics (e.g., task completion times, System Usability Scale scores) and qualitative feedback. Additionally, perform controlled experiments to correlate similarity scores with actual exam performance.

3. **Personal and Professional Growth**

Continue refining software engineering skills—particularly in mobile development and DevOps—by managing the project lifecycle: from sprint planning and version control to continuous integration/continuous deployment (CI/CD). Treating this tool as an ongoing personal project will deepen mastery of modern development workflows.

In conclusion, this TFG lays a solid foundation for an innovative educational tool at the intersection of AI, voice interfaces, and legal studies. The successful prototype, combined with the lessons learned during its development and testing, inspires confidence that with further investment—both technical and economic—the application can evolve into a valuable resource for aspiring jurists and beyond.

10 ACKNOWLEDGMENTS

I want to express my most sincere gratitude to Professor Sergio Sayago, supervisor of this Bachelor's Thesis, for his constant support and invaluable advice in the drafting and structuring of this document. His guidance regarding style, expository clarity, and chapter organization has been essential in shaping a coherent and accessible project, and has enabled me to develop greater rigor in presenting my ideas.

I dedicate a very special tribute to the memory of my father, who passed away in March 2024 after a hard-fought battle with cancer. His absence, in the midst of preparing this thesis, represented for me a personal and academic challenge of enormous magnitude: the sadness and difficulties arising from his loss forced me to find in my passion for computer science and artificial intelligence an extra reason to carry on. This work is also a small homage to his strength and all the love he gave me.

Finally, I am grateful for the unconditional support of my family and friends, who in the most difficult moments have encouraged me and shared with me the excitement of completing this stage. To all of you, thank you for always being by my side.

11 REFERENCES

- Google Cloud. (2025). *Cloud Speech-to-text*. Obtenido de https://cloud.google.com/speech-to-text?utm_source=google&utm_medium=cpc&utm_campaign=emea-es-all-en-dr-bkws-all-all-trial-b-gcp-1707574&utm_content=text-ad-none-any-DEV_c-CRE_654782093415-ADGP_Hybrid+%7C+BKWS+-+MIX+%7C+Txt+-+AI+And+Machine+Learning+-+Spe
- Justito el Notario. (8 de Enero de 2020). *Justito el Notario - Blog*. Obtenido de Justito el Notario - Blog: <https://www.justitonotario.es/los-opositores-de-justito-y-tu-con-quien-te-identificas/#:~:text=3.800,%C2%BFRecuerdas%C2%A0cu%C3%A1ntas%20palabras%2Ffolios%2Ftema%20aproximadamente%20ten%C3%ADa%20tu>
- Martins, J. (2025). *Asana*. Obtenido de <https://asana.com/es/resources/what-is-scrum>
- Perea, J. E. (2020). *Desarrollo de aplicación móvil para la captura de voz, interoperabilidad e integración de los*. Obtenido de Universidad de Antioquia: <https://bibliotecadigital.udea.edu.co/server/api/core/bitstreams/6bace2f3-00f8-4744-b853-21fcecc4af95/content>
- Rate, W. E. (s.f.). *Wikipedia*. Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Word_Error_Rate#:~:text=,tiene%20la%20frase%20de%20referencia