Pronunciation Coach – First Milestone

Team 1: Alondra Arce, Fabian Velez, Julian A. Toro, Aryam Z. Diaz, Jaydiemar Vazquez, Kevin Lara, Kevin Ruiz

Update: October 14, 2025 (In Progress)

Table of Contents

1. Informative Part

1.1 Teams

Team Leader & Architecture - Alondra Arce

Research & Documentation Lead - Fabian Velez

Security & Authentication - Julian A. Toro

Quality Assurance & Agile Processes - Aryam Diaz

Architecture & Visual Design - Jaydiemar Vazquez

Backend & Data Research - Kevin Lara

UI Components & Features - Kevin Ruiz

1.2 Current Situation, Needs, Ideas

1.2.1 Current Situation

For native Spanish speakers, learning and pronouncing English words and phrases presents a significant challenge due to fundamental differences between the two languages. Mobile applications often suffer from processing delays during authentication and data operations, leading to user frustration. **Existing applications typically use generic loading indicators that provide no educational value during wait times.**

Source: "Why is English pronunciation difficult for a Spanish speaker?", London Speech Workshop, https://londonspeechworkshop.com/why-is-english-pronunciation-difficult-spanish-speaker/

1.2.2 Needs

- Need for accessible pronunciation practice outside classroom settings
- Requirement for immediate, objective feedback on pronunciation attempts
- Need for structured progression through phonetically challenging sounds
- **Need for engaging visual feedback during application processing delays**
- **Requirement for educational content utilization during loading periods**
- Need for tracking personal progress and identifying persistent difficulties
- Requirement for motivational systems to maintain consistent practice habits

1.2.3 Ideas

- Interactive pronunciation exercises with instant feedback mechanism
- Phoneme-focused practice modules targeting specific sound challenges

- Visual comparison interface between learner and native speaker pronunciation
- Progressive difficulty system that adapts to user improvement
- **Advanced loading system with multiple visual strategies and pronunciation facts**
- **Enterprise design patterns for maintainable loading component architecture**
- Achievement system to encourage regular practice and milestone completion
- Personalized practice recommendations based on performance analytics

1.3 Scope, Span, and Synopsis

1.3.1 Scope and Span

Scope: Mobile-based language learning application specializing in pronunciation improvement. The first milestone focuses exclusively on building the front-end visual and architectural foundation. **including an advanced loading component system**

Span: English pronunciation training for Spanish-speaking adults. This initial phase spans the development of the application's shell, including login and home screens, a reusable component library, **enterprise-grade loading system**, and core app infrastructure (routing, state management).

The project encompasses mobile development with Flutter, initial architecture setup, UI/UX design, **and implementation of design patterns for enhanced user experience**.

1.3.2 Synopsis

Pronunciation Coach is a Flutter-based mobile application designed to help Spanish-speaking adults improve their English pronunciation through targeted exercises and AI-driven feedback. The project involves developing a robust architecture for handling authentication flows, creating a comprehensive component system for consistent UI/UX, **implementing an advanced loading system using enterprise design patterns**, and designing an engaging activity-based learning progression.

The solution aims to make pronunciation practice accessible, effective, and engaging through technology-enabled learning tools **that maintain user engagement even during processing delays**.

1.4 Other Activities (Beyond Coding)

- Domain Engineering: Research on English phonetics and common Spanish speaker challenges.
- Requirements Analysis: User needs assessment and feature prioritization for the UI/UX.
- Architecture Design: Design of application routing structure, state management, and project organization following clean architecture principles.
- **Component System Design: Implementation of enterprise design patterns for loading states**

- Research: Comprehensive analysis of secure authentication, backend solutions, data caching, CI/CD, and visual design.
- Documentation: Management of project plans, research findings, and technical specifications.

1.5 Derived Goals

- Development of a reusable Flutter component library for educational applications.
- Establishment of a scalable and maintainable codebase using clean architecture principles.
- Creation of a robust authentication flow that can be integrated with a secure backend.
- **Implementation of an advanced loading system using Factory, Strategy, Singleton, Observer, Decorator, and Template Method patterns**
- Implementation of a responsive and accessible design system.

2. Descriptive Part

2.1 Domain Description

2.1.1 Domain Rough Sketch

(This section is a preliminary sketch based on the research made by team 1)

- User: Spanish-speaking adult, motivated to learn, may be frustrated with current tools **and application loading times**
- Goal: Improve English pronunciation.
- Activity: Logs into app, **sees engaging loading animations with pronunciation tips**, sees progress, selects a practice module, records their voice, receives feedback, tracks improvement.
- System: Mobile app, requires login, has home dashboard, practice sections, profile **advanced loading component system**.
- Data: User account, authentication tokens, progress data, practice history **loading strategies and pronunciation facts**.

2.1.2 Terminology

- Phoneme: The smallest unit of sound in a language that can distinguish words (e.g., /θ/ in "think" vs. /s/ in "sink").
- Authentication: The process of verifying a user’s identity (e.g., via email and password).
- JWT (JSON Web Token): A compact, URL-safe means of representing claims to be transferred between two parties, used for securing authentication.
- State Management: The handling of the state of the application (e.g., whether a user is logged in or not) in a predictable way.
- Component Library: A collection of reusable UI elements (buttons, input fields, cards) that

ensure design consistency.

- Routing/Navigation: The mechanism for moving between different screens in the application.
- **LoadingStrategy**: A design pattern implementation for interchangeable loading animations
- **Factory Pattern**: A creational pattern that provides an interface for creating objects in a superclass
- **Observer Pattern**: A behavioral pattern for notifying multiple objects about state changes
- **Decorator Pattern**: A structural pattern that adds behavior to objects without altering their structure

2.1.3 Domain Terminology vs Rough Sketch

The terminology was derived from analyzing the needs of the domain (language learning) and the technical solution (a Flutter app). Terms like **Phoneme** and **Progress** come from the educational domain, while **JWT**, **State Management**, and **Routing** are technical concepts required to build a secure and functional application shell. **The loading system terminology (LoadingStrategy, Factory Pattern, etc.) emerged from addressing user frustration with processing delays and the need for engaging wait states.**

2.1.4 Narrative

A user, Maria, wants to improve her English pronunciation. She downloads the Pronunciation Coach app. Upon opening it, she is presented with a clean login screen. She enters her credentials **and sees an engaging pulsating wave animation with a pronunciation fact about the 'th' sound while the system authenticates her**. She arrives at her home page, which welcomes her and shows her current learning streak, recent progress, and suggests a new sound to practice. The app is intuitive, responsive, **and maintains engagement even during processing through varied loading strategies**, and makes her feel confident to start her practice session.

2.1.5 Events, Actions, Behaviors

- Event: User presses the "Login" button.
- Action: The system validates the input fields and sends credentials to the authentication service **while displaying a loading strategy**.
- Behavior: If authentication is successful, the application's state changes to "authenticated," and the user is navigated to the Home screen.
- **Event: Processing operation begins**
- **Action: Loading system selects and displays appropriate strategy**
- **Behavior: User views pronunciation facts while waiting for operation completion**

2.1.6 Function Signatures

(High-level domain operations, not final code)

- `authenticateUser(credentials: Credentials): AuthenticationResult` - Validates user credentials.

- `navigateTo(screen: ScreenName)` - Changes the current view of the application.
- `getUserProfile(userId: UserID): UserProfile` - Retrieves the user's data for display on the home screen.
- **`createLoadingStrategy(type: StrategyType): LoadingStrategy` - Factory method for loading animations**
- **`displayLoadingFeedback(message: String, strategy: LoadingStrategy)` - Shows engaging loading state**
- **`getRandomPronunciationFact(): String` - Retrieves educational content for loading displays**

2.2 Requirements

2.2.1 User Stories, Epics, Features

Epic: User Authentication * As a new user, I want to log in with my email and password so that I can access my personalized learning content. * As a user, I want to see clear error messages if my login fails so that I can correct my information. * As a user, I want my session to be managed securely so that my account remains protected. * **As a user, I want to see engaging loading animations during authentication so that wait times feel shorter**

Epic: Application Foundation * As a developer, I want a well-organized project structure following clean architecture so that the codebase is maintainable and scalable. * As a developer, I want a central state management solution so that the user's authentication state can be shared across the app. * As a developer, I want a library of reusable UI components so that we can ensure design consistency and speed up development. * **As a developer, I want a reusable loading system with multiple strategies so that I can implement engaging loading states easily**

*Epic: **Enhanced User Experience** * **As a user, I want to learn pronunciation tips during loading screens so that I can make productive use of waiting periods** * **As a user, I want to see varied loading animations so that the application remains visually engaging** * **As a user, I want consistent loading behavior across the app so that I have a predictable experience**

2.2.2 Personas

- **Maria, the Motivated Learner**: A 28-year-old professional from Mexico. She uses English at work but is self-conscious about her accent. She is tech-savvy and uses her phone for most tasks. She needs structured, feedback-driven practice she can fit into her busy schedule **and gets frustrated by unproductive waiting times in applications**.
- **Carlos, the Consistent Student**: A 45-year-old teacher from Colombia preparing to move to an English-speaking country. He is dedicated but has limited time. He needs clear goals, progress tracking, and motivation to practice daily **and appreciates educational content even during application loading periods**.

2.2.3 Domain Requirements

1. The system must restrict access to user-specific data until identity is verified (authentication).
2. The system must provide a clear and intuitive path for the user to begin their learning activities.

3. The system must present information (progress, goals) in a motivating and visually clear way.
4. **The system must provide engaging visual feedback during processing operations to maintain user engagement**.
5. **The system must educate users during waiting periods through relevant pronunciation tips**.

2.2.4 Interface Requirements

- The login screen shall have input fields for email and password.
- The login screen shall have a button with the label "Login".
- The home screen shall display a welcome message containing the user’s name.
- The application shall transition from the login screen to the home screen upon successful authentication.
- **The system shall display engaging loading animations during authentication processing**
- **The system shall show pronunciation facts during loading states**
- **The system shall maintain consistent loading behavior across all processing operations**

2.2.5 Machine Requirements

- The application shall render correctly on iOS and Android devices.
- The initial app startup time shall be under 400ms on a mid-range device.
- The UI shall respond to user input (e.g., button presses) within 16ms for a smooth 60fps experience.
- **Loading animations shall maintain 60fps smoothness during operation**
- **The loading system shall have minimal performance impact on core application functionality**
- **Authentication with loading feedback shall complete within 3 seconds**

2.3 Implementation

2.3.1 Selected Fragments of Implementation

The application is organized using a feature-based modular architecture that separates concerns and promotes maintainability. The structure follows Flutter best practices with clear separation between core functionality, feature modules, and shared components.

Application Architecture Overview:

The project is divided into three main sections: core infrastructure, feature modules, and the advanced loading system. The core section contains shared utilities and base components used across the entire application. This includes application constants for consistent styling, reusable widget components, and state management providers.

Welcome Screen Architecture Flow:

The application launch sequence has been transformed from a simple direct structure to a multi-layered, modular system designed for scalability and professional user experience. Previously, the app followed a linear architecture that immediately displayed the MainNavigationScreen after launch, with no authentication, onboarding, or responsive design layers.

The new architecture introduces a sophisticated flow:

- Application launches and executes main.dart wrapped with Sizer for responsive layout
- Global ChangeNotifierProvider manages state for user sessions and authentication
- Animated WelcomeScreen displays as the entry point for approximately 4 seconds
- Automatic navigation transitions to the LoginPage for authentication
- Upon successful authentication, navigation proceeds to MainNavigationScreen

This architectural evolution represents a significant enhancement from the previous flat tab-based navigation to a hierarchical multi-level system that supports onboarding, authentication flows, and conditional routing based on user session state.

Component Responsibilities and Data Flow:

Each architectural layer has defined responsibilities:

- WelcomeScreen: Provides animated branding and sets visual tone before user interaction
- LoginPage: Handles authentication UI and prepares for backend integration
- MainNavigationScreen: Manages the core application dashboard with tab-based navigation
- Sizer Layer: Ensures responsive design across mobile and tablet devices
- State Management: Coordinates authentication state and user session persistence

The data flow follows a clear sequence from user input through authentication to dashboard access, with each component maintaining separation of concerns while enabling seamless transitions between application states.

Loading System Implementation Process:

The loading system represents a significant architectural enhancement, implementing enterprise design patterns to create engaging user experiences during processing operations. This system is structured around six key design patterns that work together to provide varied, educational loading states.

The loading process begins when a user triggers an operation that requires processing time, such as authentication. The system uses a Factory Pattern to create appropriate loading strategies, selecting from four distinct visual approaches: pulsating wave animations, progress stage indicators, morphing shape transformations, and text typing simulations.

Each loading strategy follows a consistent Template Method Pattern that ensures all loading states display educational pronunciation facts while maintaining unique visual characteristics. The

Strategy Pattern allows these loading types to be interchangeable at runtime, providing variety and preventing user fatigue.

A Singleton Pattern manages the global loading state, ensuring consistent behavior across the entire application. Components throughout the app can observe loading state changes using the Observer Pattern, allowing for coordinated user interface updates during processing operations.

The Decorator Pattern enhances loading presentations with visual features like custom backgrounds and blur effects, creating polished, professional loading experiences. Throughout all loading states, the system displays relevant pronunciation facts from a curated database, turning waiting periods into educational opportunities.

Architectural Benefits and Future Readiness:

This combined architectural approach ensures that both the welcome screen flow and loading system work together to create a polished, professional user experience. The modular design allows independent development and testing of each component while maintaining clear separation of responsibilities.

The architecture lays groundwork for future enhancements including:

- Authentication integration with Firebase Auth or REST APIs
- Session persistence for returning users
- Conditional routing based on login state
- Dynamic theming based on user preferences
- Scalable user management systems

This architectural foundation transforms the application from a simple prototype into a production-ready system capable of supporting complex user flows while maintaining engaging experiences during all application states, including processing delays.

3. Analytic Part

3.1 Concept Analysis

The rough sketch identified key domain entities: User, Practice, and Progress. These abstractions led to the terminology that defines both the problem space (phoneme, feedback) and the solution space (authentication, routing, state). **The implementation of the loading system introduced new concepts derived from user experience challenges with processing delays, leading to enterprise design pattern applications.**

The narrative connects these concepts into a coherent user journey, validating that the initial implementation focus (authentication and home screen) **enhanced with advanced loading states** correctly establishes the foundation for the user's interaction with the app.

The technical research (auth security, architecture) **and loading system design patterns** ensures the solution space concepts are implemented robustly **while maintaining engaging user**

experience.

3.2 Validation and Verification

Testing Plans: As per Research #14 and #19, testing will include:

- Unit Tests: For the SessionController state changes (login/logout logic) **and LoadingStrategyFactory strategy creation**.
- Widget Tests: For Login Screen input validation and button enabling/disabling **and loading animation rendering**.
- Integration Tests: For the complete flow from Login to Home navigation **including loading state transitions**.

Walkthroughs: The team will conduct peer code reviews on all pull requests to verify architecture adherence and code quality **including loading system design pattern implementation**.

Scenarios used for validation:

1. Happy Path: Enter valid credentials → Login button enables → Press button → **Loading animation displays with pronunciation fact** → Navigates to Home.
2. Validation Error: Enter invalid email format → Error message appears under field → Login button remains disabled.
3. Authentication Error: Enter incorrect credentials → **Error-specific loading strategy displays** → SnackBar with generic error message appears.
4. Offline Scenario: With no internet, the Home screen should still render any cached data **with appropriate loading states for offline operations**.
5. **Loading Variety**: Multiple authentication attempts demonstrate different loading strategies and pronunciation facts.