Technical Project Report - Android Module

Wizer

Subject:	Universidade de Aveiro - Computação Móvel
Date:	30/06/2025
Students:	106093: Caroline Ribeiro 106082: Mateus da Fonte 128660: Victor Milhomem
Project abstract:	The Wizer app is a mobile educational platform designed to support students and teachers in managing study groups, quizzes, and interactive exercises, fostering academic engagement in collaborative environments. The application supports user authentication with role differentiation (student and teacher), integrates with Supabase for database and auth services, and offers role-specific features.

Report contents:

1 Application concept

Overview

Wizer is a mobile app designed to support academic collaboration and self-improvement among students and educators. The platform provides a centralized space where users can join or manage study groups, participate in quizzes, and complete exercises tailored to their learning topics.

Who are the typical users?

- **Students** (primarily high school or university level)
- **Teachers or tutors** responsible for organizing study activities

How do they benefit?

- **Students** can:
    - Join multiple study groups and access quizzes shared by their teachers.
    - Track their own performance through scores and submissions.
    - Practice with randomly generated exercises tailored to the subject of their study group.
    - Stay connected with their peers and improve study consistency.
- **Teachers** can:
    - Create and manage study groups for different subjects.
    - Share quizzes with specific groups using unique access codes.
    - Track student engagement through group membership and quiz submissions.

The app fosters a more organized, interactive, and motivating learning environment, bridging the gap between educators and learners through structured academic tools accessible directly from a smartphone.

Essential journey map

“Joining a Study Group and Completing a Quiz” (Student Persona)

Stage	**User Actions**	**User Thoughts & Feelings**	**Touchpoints**	**Opportunities**
1. Discover App	Student downloads Wizer and creates an account	“I hope this helps me stay organized in my studies.”	App store, Onboarding screens	Offer an engaging onboarding experience to build trust and clarity
2. Join Group	Student enters a group code provided by the teacher	“Will I find the group I need? Is the code correct?”	“Enter Group Code” dialog	Provide immediate feedback for success/failure of group join
3. View Group Page	Sees subject name, teacher’s name and group members	“Great, I found my math group. Who else is here?”	Group Detail Screen	Show relevant info (teacher name, list of participants, upcoming activities)
4. Take a Quiz	Taps on the quiz tab and selects a quiz available to their group	“Let’s see how much I remember from class…”	Quiz List Screen, Quiz Detail Screen	Provide a progress indicator and feedback at the end
5. Review Results	Submits the quiz and sees the score and correct answers	“Nice! I scored 7/10. I know what to review now.”	Quiz Submission Screen	Display detailed feedback; encourage retry or practice exercises
6. Practice More	Goes to the Exercises section to try 10 random questions from their subject	“These help me practice the tough parts.”	Exercises Screen	Allow daily practice sessions and track progress over time

7. Ongoing Use	Keeps using Wizer regularly, joining new groups, checking rankings, etc.	“I feel more confident for my tests now.”	Navigation Bar, Home Page	Add gamification (badges, streaks), reminders, and a leaderboard for engagement

2 Implemented solution

Architecture overview (technical design)

The architecture of our mobile application follows a clean MVVM (Model–View–ViewModel) pattern, where the UI (View) layer remains fully decoupled from the business logic and data access layers. This modularity was essential to manage the app's growing complexity as we integrated multiple services such as authentication, group management, quizzes, and exercise modules.

At the Model layer, we defined all data classes (e.g., `User`, `Group`, `Quiz`, `Question`, `QuizSubmission`) using Kotlin's `@Serializable` annotation to ensure compatibility with Supabase's JSON-based PostgREST API. These models reflect the structure of the underlying Supabase tables and facilitate seamless serialization and deserialization during data exchange.

The ViewModel layer acts as the central logic handler for each screen, managing UI state, coordinating data fetches from services, and handling events such as quiz submissions or group joining. Although we did not use Jetpack's LiveData, we utilized Kotlin's `mutableStateOf` and `remember` to maintain reactive state across composables in Jetpack Compose. Coroutine scopes (`rememberCoroutineScope`) were used to manage asynchronous data fetching in a lifecycle-aware manner.

The Repository/Service layer encapsulates Supabase interactions. We followed a Repository pattern, centralizing all Supabase queries inside service classes like `UserService`, `GroupService`, `QuizSubmissionService`, `QuestionService`, and `QuizzesService`. Each service is responsible for connecting to a specific table and exposing high-level operations (e.g., `getGroupsForUser`, `joinGroupWithCode`, `getSubmissionsByUserId`). This abstraction allows the UI layer to remain agnostic to how data is retrieved or stored, making testing and maintenance much easier.

For backend integration, we used Supabase as our BaaS (Backend-as-a-Service) provider. Supabase provides RESTful APIs (via PostgREST), authentication, and role-based access control. Supabase’s tables are used for all persistent data, and queries are made via the official Supabase-KT library, which handles authentication tokens, JSON parsing, and API errors natively.

We also followed Android Architecture Guidelines where applicable:

- **Model:** Kotlin data classes annotated with `@Serializable`.
- **View:** Built exclusively using Jetpack Compose, providing a reactive and declarative UI layer.
- **ViewModel (Controller):** Handled in each screen using `rememberCoroutineScope` and `mutableStateOf`.
- Although we did not use Room for local persistence, all stateful data from Supabase is cached in memory and retained during recompositions.

In terms of data persistence and update strategy, the application relies entirely on Supabase's hosted PostgreSQL database. Updates are triggered manually by user actions and reflect immediately on-screen due to reactive state observation. No offline mode or local caching is currently implemented, as the app assumes consistent internet access.

Advanced Design Elements

While the app is mostly CRUD-oriented, we implemented several design strategies to improve user experience and scalability:

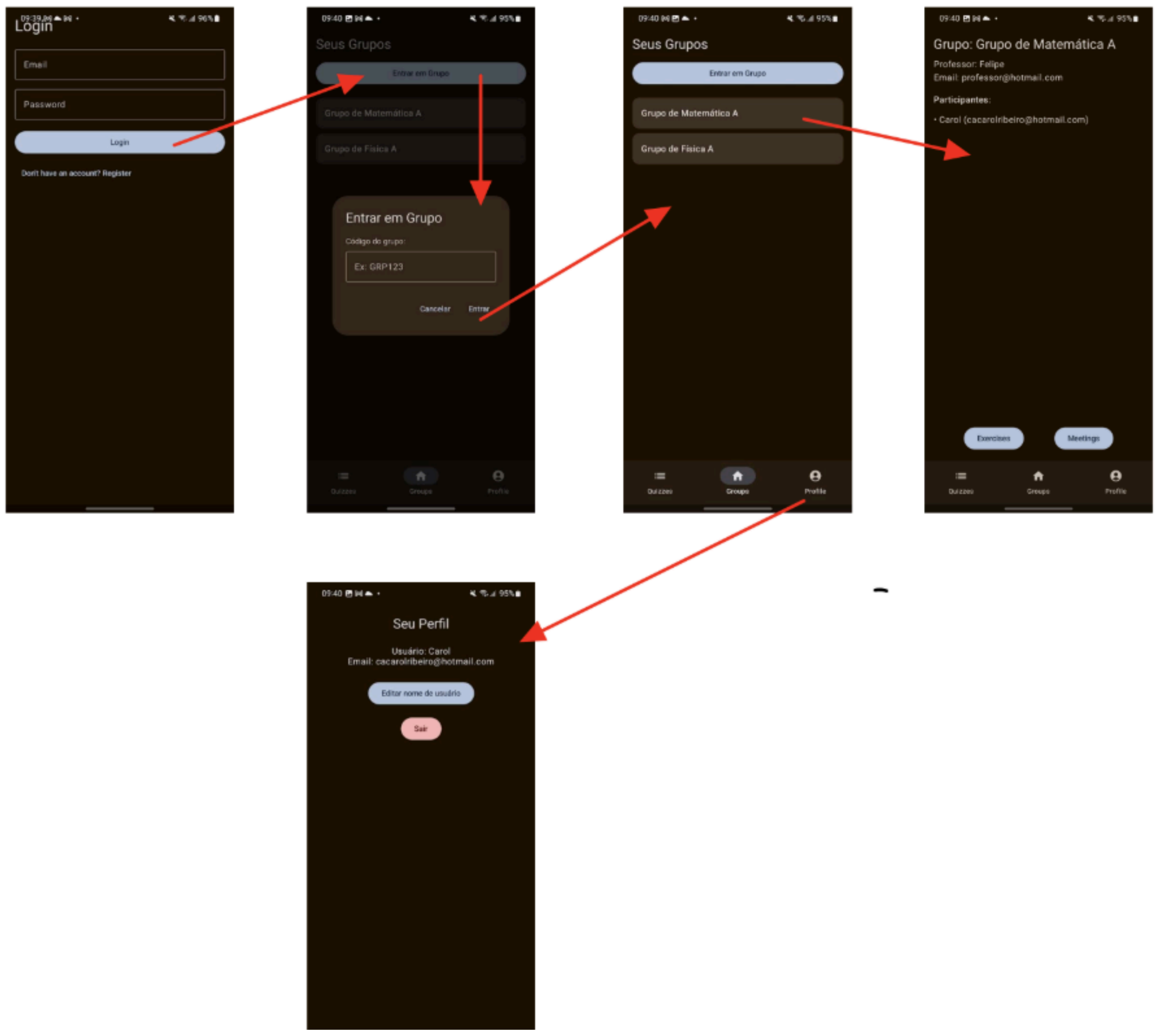
- **Remote service integration:** All core features like group membership, quiz participation, and question loading are integrated with Supabase in real time.
- **Secure authentication:** Supabase's built-in email/password authentication ensures that user sessions are persisted securely.
- **Randomized exercise generation:** The "Exercises" screen retrieves a random sample of 10 questions from the subject associated with the user's group, making use of Kotlin's `shuffled().take(10)` to simulate a personalized learning experience.
- **Navigation with arguments:** Dynamic routing is used to pass group IDs, subject IDs, and user context between screens using Compose's `NavHost` and route parameters.
- **UI responsiveness:** Jetpack Compose enables fast feedback, real-time updates, and lightweight UI design, greatly improving performance and user interaction.

Although not yet implemented, the project could evolve to include push notifications (via

Firebase Cloud Messaging), offline caching (Room or local JSON storage), and even Bluetooth/Nearby communication for local group sessions — depending on future needs. The modular architecture already in place would allow these additions with minimal disruption to the existing codebase.

Implemented interactions

“Enter a group and see details”



Project Limitations

While the current version of the application provides essential functionality for student group management, quiz tracking, and exercise generation, there are several planned features and improvements that remain unimplemented:

- **No Offline Support:** The app currently requires a constant internet connection, as all data is retrieved directly from Supabase. Implementing local caching (e.g., using

Room or DataStore) would allow users to access group data, past submissions, and exercises even without internet access.

- **No Push Notifications:** Students do not currently receive real-time updates about new quizzes, meetings, or changes to groups. A future integration with Firebase Cloud Messaging (FCM) or Supabase Edge Functions could enable personalized and timely notifications.

- **Missing Admin Interface:** Professors currently have no dedicated interface within the app to create quizzes or manage groups dynamically. Adding role-specific UIs and permissions would improve the teacher experience.

- **Limited Error Feedback:** Many of the app's error messages are generic (e.g., "Código inválido"), and there is no robust mechanism for users to know *why* an operation failed (e.g., invalid group code, connection issue, or permission error). More detailed error handling and user-friendly messages would improve UX.

- **No Quiz Solving or Correction Flow:** While students can view their quiz scores, the actual quiz-solving interface and question-by-question feedback are still under development. A complete quiz-taking UI with timers, interactive questions, and result breakdown is planned.

- **No Real-Time Sync or Collaboration:** Features like collaborative meetings, chat, shared study materials, or group progress visualization are currently outside the scope but could greatly enhance engagement and community building.

- **Basic Design and Accessibility:** Although the UI is responsive and built with Compose, it lacks customization for dark mode, accessibility support (e.g., screen reader compatibility), and visual refinement.

- **Limited Sensor Integration:** While there was an initial plan to explore device sensors (e.g., accelerometer for attendance), that part of the scope was deprioritized for this release but remains a potential area of innovation.

Overall, the app lays a strong foundation with a scalable architecture, and the integration of the missing features listed above will significantly enhance its educational value and usability in real-world academic settings.

New features & changes after the project presentation

Not applicable.

3 Conclusions and supporting resources

Lessons learned

Throughout the development of this mobile application, our team faced a series of technical and conceptual challenges that contributed significantly to our learning experience. One of the most notable difficulties involved the integration of Supabase with Kotlin and Jetpack Compose. While Supabase offers a robust backend-as-a-service solution, adapting it to work seamlessly with Kotlin Coroutines and Compose required special attention. For example, some queries failed silently due to type mismatches or missing serializers. We overcame this by explicitly annotating all models with `@Serializable` and using decoding methods like `decodeList<T>()` and `decodeSingleOrNull()` to ensure safe and accurate data handling.

Another major issue occurred with the handling of UUIDs in Supabase queries. We discovered that navigation arguments passed between screens could contain trailing spaces, which caused the Supabase queries to break unexpectedly. These issues were difficult to debug at first, as there was no clear error message. To address this, we implemented `.trim()` operations on route arguments and added detailed logging throughout the codebase, which allowed us to detect and resolve these bugs more quickly.

Synchronizing state across different screens was also a challenge, especially for group memberships, session management, and quiz submissions. To solve this, we moved shared logic to centralized ViewModels and ensured that important identifiers like `userId`, `groupId`, and `subjectId` were passed explicitly between composables and route arguments. This approach improved both maintainability and reliability across the app.

On the topic of Android development, we were pleasantly surprised by the expressiveness and power of Jetpack Compose. Compared to traditional XML layouts, Compose allowed us to build UIs faster and with more clarity. Its built-in support for reactive state and declarative design significantly reduced the complexity of managing UI state and user interactions. However, we also found that navigating with dynamic route arguments in `NavHost` could be error-prone. Strings such as UUIDs needed careful parsing and validation, and we learned that even minor formatting issues could cause entire screens to fail loading.

In terms of backend infrastructure, Supabase exceeded our expectations. Its combination of authentication, PostgREST, and real-time capabilities gave us everything we needed to build a full-stack application without managing our own servers. This allowed us to focus more on app logic and user experience rather than infrastructure.

Reflecting on the course “Computação Móvel”, we believe it offers an excellent foundation for students interested in mobile development. However, based on our experience, we recommend that future editions dedicate more time to backend integration and asynchronous programming, especially when using services like Supabase. Many students begin the course with a solid understanding of UI components but encounter difficulties when dealing with data synchronization and remote queries. We also advise future students to start small and iterate — building a minimum viable product early and gradually expanding its functionality helped us manage complexity and maintain focus.

In conclusion, this project provided valuable hands-on experience with modern Android development practices, highlighted key architectural lessons, and reinforced the importance of clean code, debugging strategies, and user-centered design.

Work distribution within the team

- **Authentication**: The authentication component using Supabase was developed by Victor Milhomem.
- **Student**: The student screens integrated with Supabase were developed by Caroline Ribeiro. This includes the screens for quiz, groups, profile, group detail, exercises, and meetings.
- **Services**: All services were developed by Victor Milhomem.
- **Teacher**: The professor-related screens integrated with Supabase were developed by Victor Milhomem and Mateus da Fonte.
- **View Model**: The view models for the professor section were implemented by Victor Milhomem.
- **Supabase**: The creation of the Supabase tables was shared between Victor Milhomem and Caroline Ribeiro, according to the needs that arose on the student or professor side.
- **Models**: The data models were created by Victor Milhomem and Caroline Ribeiro based on the needs of both the student and professor sides, and in alignment with the database structure.
- **Report:** The report was made by Caroline Ribeiro.

Taking into consideration the overall development of the project, the contribution of each team member was distributed as follows: Caroline Ribeiro did 40% of the work, Mateus da Fonte contributed 20% and Victor Milhomem did 40%.

Reference materials

This application was built using a variety of powerful tools, libraries, and web services that played a fundamental role in enabling the development of core functionalities. These resources are especially recommended for other students pursuing similar educational or collaborative mobile projects.

Libraries and Frameworks

- **Jetpack Compose (Android)**
    - Used for building the entire UI using modern declarative programming.
    - Benefit: Cleaner UI code, improved performance, and real-time previews.
    - Jetpack Compose official docs
- **Supabase (supabase-kt)**
    - Used as the backend-as-a-service (authentication, database, and API).
    - Supabase client (`supabase-kt`) for Kotlin handles authentication and real-time PostgREST API access.
    - supabase-kt GitHub repo
- **Kotlin Serialization**
    - For converting JSON data to Kotlin objects (`@Serializable` annotation).
    - Integrated seamlessly with Supabase and Jetpack Compose.
- **Coroutines and Flows**
    - For asynchronous database calls and state management in the UI.
    - Ensures UI remains responsive during network operations.
- **Navigation-Compose**
    - For handling in-app navigation between screens using route-based navigation.

Web APIs and Backend

- **Supabase PostgREST**
    - RESTful API provided automatically for every table in the Supabase PostgreSQL database.
    - Allows secure filtering, inserting, and retrieving records with minimal setup.

- **Supabase Auth**
  - Email and password-based authentication.
  - Easy to integrate, with built-in user session management.

Other Useful References

- **Supabase Docs** – https://supabase.com/docs
  - Clear, developer-friendly documentation covering setup, authentication, API access, and table management.
- **Compose Academy Blog** – https://compose.academy
  - Useful for Jetpack Compose tips, patterns, and design suggestions.
- **Blog: “Supabase with Jetpack Compose” (dev.to)**
  - A hands-on post that helped during early stages of Supabase integration.

Project resources

Resource:	Available from:
Code repository:	<URL to shared git repo; be sure that teacher can access!>
Ready-to-deploy APK:	<put URL for apk; may be inside the code repo; the debug apk, automatically generated, would be Ok>
App Store page:	Not applicable
Demo video:	Not applicable