|  |  |
| --- | --- |
| **MOBILE APPLICATION DEVELOPMENT**  ChessKel | Brief description  Allow users to create profiles, register their location, and play matches against a local AI or an opponent on the same LAN, managing all user and match information through CRUD operations with SQLite.  Ing.Caleb Rodríguez  Mobile Application Development |

Contend

[Introduction 1](#_Toc211631359)

[What the Application Does 1](#_Toc211631360)

[How is it Built? 1](#_Toc211631361)

[Development of the ChessKel Project 2](#_Toc211631362)

[1. Architecture and Data Persistence (SQLite and Accounts) 2](#_Toc211631363)

[1.1 Local Account Management and Authentication 2](#_Toc211631364)

[1.2 Structure and CRUD with SQLite 2](#_Toc211631365)

[2. Domain Logic (The "Brain" of the Game) 3](#_Toc211631366)

[3. Communication Layer (LAN Multiplayer) 3](#_Toc211631367)

[Movement Exchange Protocol: 4](#_Toc211631368)

[4. Key Utilities 4](#_Toc211631369)

# Introduction

Hello! This is the ChessKel project, a chess application that we built for Android phones using the Kotlin language.

Our goal was to create a complete chess app that goes beyond just moving pieces. We want people to be able to play, register, and save their progress.

## What the Application Does

* **Your Profile and Location:** You can create an account with your name, password, and even a photo. The application is smart and can automatically register your country and region when you sign up.
* **Flexible Gameplay:** You have two ways to play:
  + **Against the Computer (AI Bot):** We created an automatic opponent. If you want a challenge, the bot can use the Minimax algorithm to think through its moves.
  + **Against a Friend at Home (LAN):** If you are on the same Wi-Fi network, the app allows you to connect and play against another person using Socket technology or Nearby Connections.
* **Memory and History:** We use a small database called SQLite inside the phone. This allows us to:
  + Save and update all your user information (your data, your photo).
  + Record all your played games so you can view your history.

## How is it Built?

The application is organized in layers (like a sandwich 🥪) to make it easy to maintain:

* **Screens (UI):** This is what you see (Menus, the Chessboard, the Login screen).
* **Logic (Domain):** This is where the chess rules and the Bot's intelligence reside.
* **Data (SQLite):** Handles saving and reading all the information from the database.
* **Communication (Network/Utilities):** Here we manage the connection for multiplayer and the geolocation function.

In summary, ChessKel is proof that we can combine complex functions like geolocation, multimedia, local networking, and an AI into a single, functional mobile application.

# Development of the ChessKel Project

The development of ChessKel is based on a modular four-layer architecture (UI, Domain, Data, and Communication), using Kotlin as the primary language and SQLite for local data persistence.

## 1. Architecture and Data Persistence (SQLite and Accounts)

The management of users and games is entirely local, with no dependency on cloud services.

### 1.1 Local Account Management and Authentication

The creation and validation of accounts are handled internally:

* **Creation and Security:** When a user registers (RegisterActivity), the entered password is immediately converted into an unreadable string (passwordHash) using a hashing function. Only this hash is saved in the usuarios (users) table in SQLite.
* **Login Mechanism:** In the LoginActivity, the application takes the password entered by the user and generates a temporary hash on the spot. If this temporary hash matches the passwordHash stored in SQLite, access is granted.
* **Session Maintenance:** After a successful login, the application stores the user's unique ID in a local configuration file (Preferences.kt). This ID is used to quickly load the profile and keep the session active between application restarts.

## 1.2 Structure and CRUD with SQLite

The data layer is managed using the DAO (Data Access Object) pattern with the central class DBHelper.kt, which implements all CRUD (Create, Read, Update, Delete) operations for the two main tables:

|  |  |  |
| --- | --- | --- |
| Table | Purpose | CRUD Operations |
| Usuarios (users) | Stores profile data (name, email, passwordHash, location, profile picture). | **Create** (Registration), **Read** (Load Profile), **Update** (Edit Profile), **Delete** (Delete Account). |
| Partidas (games) | Stores game history, including player IDs and the final/current game state (saved using FEN notation). | **Create** (New Game), **Read** (Load History), **Update** (Resume/Finish Game), **Delete** (Clear History). |

## 2. Domain Logic (The "Brain" of the Game)

The chess logic resides in the Domain layer and is the foundation of the game's functionality.

* Tablero.kt**(Board.kt):** This is the central model that represents the 8x8 matrix of squares and keeps track of the pieces (Pieza.kt). It uses FEN (Forsyth-Edwards Notation) to serialize and deserialize the complete board state into a string, allowing games to be easily saved and loaded.
* AjedrezController.kt**(ChessController.kt):** Acts as the rules engine. It is responsible for:
  + Validating moves (mover(m: Movimiento)) based on the rules of the pieces.
  + Detecting Check, Checkmate, and Stalemate.
  + Synchronizing the current state with the local database (DBHelper) to save the game.
* BotIA.kt**(AI Bot.kt):** Manages the automatic opponent. At its basic level, it uses the Board to generate and select a valid move at random. For advanced levels, it implements the Minimax algorithm to search for the optimal move to a limited depth.

## 3. Communication Layer (LAN Multiplayer)

The multiplayer feature uses a temporary Client-Server model based on TCP Sockets to connect two devices on the same Wi-Fi or LAN network.

|  |  |  |
| --- | --- | --- |
| Component | Role in the Connection | Task |
| Server Phone | Executes SocketServer.kt. It is responsible for opening a port and waiting for the other player's connection. | Sends/receives moves and manages the network session. |
| Client Phone | Executes SocketClient.kt. It connects to the Server Phone's local IP address. | Sends/receives moves to/from the server. |

### Movement Exchange Protocol:

The communication focuses on the Movimiento.kt (Move.kt) object. The phone making a move serializes the move object into JSON format (e.g., {"from":"e2","to":"e4", ...}) and sends it over the socket. The receiving phone deserializes it and applies it to its local Board, ensuring both devices remain synchronized.

## 4. Key Utilities

* **Geolocation (**LocationUtils.kt**):**
  + Uses the FusedLocationProviderClient to obtain latitude and longitude coordinates.
  + Employs Geocoder to convert these coordinates into human-readable information (country and region), which is stored in the user's profile.
* **Image Handling (**ImageUtils.kt**):**
  + Allows the user to select a profile picture using an Intent.
  + Saves the local file path of the selected image, and this path is what is stored in the foto\_perfil (profile\_picture) field of the usuarios database table.
  + The app uses this file path to load and display the image in the ProfileFragment.