## 实验一：通过文件：实现查询历史记录的存储

### 1. 文件存储的目的

在移动应用中，用户的输入数据（如车票信息）需要被持久化，以便在用户下次打开应用时能够恢复。通过将这些数据存储在设备的本地文件中，用户可以查看历史记录，提高用户体验。

### 2. 文件存储的实现逻辑

#### a. 读取文件

**方法：loadTicketInfoFromFile()**

* **目的**：从存储中读取车票信息并显示在界面上。
* **步骤**：
  1. **检查文件存在性**：
  + val file = File(getExternalFilesDir(null), "ticket\_info.json")
      
    if (file.exists()) {
  + 使用 getExternalFilesDir(null) 方法获取应用的外部文件目录，然后尝试查找名为 ticket\_info.json 的文件。
  1. **读取文件内容**：
  + val content = file.readText()
  + 如果文件存在，使用 readText() 方法读取文件内容。
  1. **解析 JSON 数据**：
  + val gson = Gson()
      
    val listType = object : TypeToken<List<TicketInfo>>() {}.type
      
    ticketInfoList = gson.fromJson<MutableList<TicketInfo>?>(content, listType)?.toMutableList() ?: mutableListOf()
  + 使用 Gson 库将 JSON 字符串解析为 MutableList<TicketInfo>，如果解析结果为 null，则使用一个空的可变列表 mutableListOf()。
  1. **更新适配器**：
  + ticketInfoListAdapter = TicketInfoListAdapter(ticketInfoList)  
    binding.ticketInfoList.adapter = ticketInfoListAdapter  
    ticketInfoListAdapter.notifyDataSetChanged()
  + 创建一个新的适配器实例并将其设置为 RecyclerView 的适配器，然后调用 notifyDataSetChanged() 通知 RecyclerView 数据已更新。
  1. **显示第一个车票信息**：
  + if (ticketInfoList.isNotEmpty()) {  
     val firstTicket = ticketInfoList[0]  
     binding.startingStation.setText(firstTicket.startingStation)  
     binding.terminal.setText(firstTicket.terminal)  
    }
  + 检查列表是否为空，如果不为空，则将第一个车票的起点站和终点站显示在输入框中。

#### b. 保存文件

**方法：saveCurrentTicketInfo()**

* **目的**：将用户输入的车票信息保存到文件中。
* **步骤**：
  1. **获取用户输入**：
  + val startingStation = binding.startingStation.text.toString()  
    val terminal = binding.terminal.text.toString()
  + 从界面获取用户输入的起点站和终点站。
  1. **创建新的车票信息对象**：
  + val newTicketInfo = TicketInfo(startingStation, terminal)  
    ticketInfoList.add(newTicketInfo)
  + 创建一个 TicketInfo 对象，并将其添加到 ticketInfoList 中。
  1. **更新适配器**：
  + ticketInfoListAdapter.notifyItemInserted(ticketInfoList.size - 1)
  + 通知适配器插入了新的项，以便更新 RecyclerView。
  1. **序列化数据并保存文件**：
  + val gson = Gson()  
    val json = gson.toJson(ticketInfoList)  
    val file = File(getExternalFilesDir(null), "ticket\_info.json")  
      
    try {  
     FileWriter(file).use { writer -> writer.write(json) }  
     Toast.makeText(this, "车票信息已保存", Toast.LENGTH\_SHORT).show()  
    } catch (e: Exception) {  
     e.printStackTrace()  
     Toast.makeText(this, "保存失败", Toast.LENGTH\_SHORT).show()  
    }
  + 将 ticketInfoList 序列化为 JSON 字符串，并使用 FileWriter 保存到文件中。使用 try-catch 块捕捉可能的异常，如果发生异常，打印堆栈信息并显示错误提示。

#### c. 清除文件

**方法：clearTicketInfoFile()**

* **目的**：清空历史记录。
* **步骤**：
  1. **检查文件存在性**：
  + val file = File(getExternalFilesDir(null), "ticket\_info.json")  
    if (file.exists()) {
  1. **清空文件内容和列表**：
  + file.writeText("") // Clear the file  
    ticketInfoList.clear() // Clear the list  
    ticketInfoListAdapter.notifyDataSetChanged()
  + 如果文件存在，则清空文件内容，并清空 ticketInfoList，然后通知适配器数据已更改。
  1. **显示成功提示**：
  + Toast.makeText(this, "历史记录已清除", Toast.LENGTH\_SHORT).show()

### 3. 异常处理

在文件读写过程中，可能会出现各种异常（例如：文件未找到、权限问题或IO异常），因此在保存车票信息时，使用了 try-catch 块来捕捉异常。如果发生异常，程序不会崩溃，而是打印堆栈信息并向用户显示错误消息。这种做法增强了应用的健壮性。

### 4. 数据结构

* **TicketInfo 类**：这个类是用于表示车票信息的数据模型，通常包含起点站和终点站的属性。
* **Gson**：用于将对象序列化为 JSON 格式，或将 JSON 格式反序列化为对象。Gson 库的使用使得数据存储和读取过程简单直观。

### 5. UI 更新

每次数据变化时（如添加新车票或清空历史记录），都需要调用适配器的通知方法（notifyDataSetChanged() 或 notifyItemInserted()），以确保 UI 反映最新的数据状态。这样可以保证用户看到的内容与存储的数据保持一致。



## 实验二：12306登录界面的实现

### 1. 文件存储的目的

文件存储的主要目的是为了持久化用户的登录信息，使得用户在下次启动应用时可以自动填充登录界面，提供更好的用户体验。通过“记住密码”功能，用户无需每次都输入用户名和密码。

### 2. 代码结构与实现逻辑

#### a. 数据模型 UserCredentials

首先，代码中使用了一个数据类 UserCredentials，该类用于存储用户的登录凭证信息：

data class UserCredentials(
  
 val username: String,
  
 val password: String,
  
 val remember: Boolean
  
)

该类包含三个属性：

* username：用户名。
* password：密码。
* remember：一个布尔值，表示用户是否选择记住密码。

#### b. 文件操作

在 LoginActivity 中，使用了以下方法来处理文件的读写操作：

1. **文件创建与初始化**：

* private val credentialsFile: File by lazy {  
   File(getExternalFilesDir(null), "user\_credentials.json")  
  }
* 使用 getExternalFilesDir(null) 获取外部文件目录，并创建一个名为 user\_credentials.json 的文件，用于存储用户凭证。

1. **加载用户凭证**：
   * **方法**：loadUserCredentials()
   * **功能**：在应用启动时读取文件内容，如果文件存在，则解析 JSON 数据并填充到输入框中。
   * **实现**：
   * if (credentialsFile.exists()) {
       
      try {
       
      val content = credentialsFile.readText()
       
      val gson = Gson()
       
      val userCredentials = gson.fromJson(content, UserCredentials::class.java)
       
       
      binding.account.setText(userCredentials.username)
       
      binding.password.setText(userCredentials.password)
       
      binding.remember.isChecked = userCredentials.remember
       
      } catch (e: JsonSyntaxException) {
       
      e.printStackTrace()
       
      Toast.makeText(this, "加载凭证失败", Toast.LENGTH\_SHORT).show()
       
      }
       
     }
   * **步骤**：
     1. 检查文件是否存在。
     2. 使用 readText() 读取文件内容。
     3. 使用 Gson 将 JSON 字符串解析为 UserCredentials 对象。
     4. 填充用户名、密码和记住密码复选框的状态。
2. **保存用户凭证**：
   * **方法**：saveUserCredentials(username: String, password: String, remember: Boolean)
   * **功能**：在用户点击登录按钮时，根据“记住密码”的选择来保存用户凭证。
   * **实现**：
   * private fun saveUserCredentials(username: String, password: String, remember: Boolean) {
       
      val userCredentials = UserCredentials(username, password, remember)
       
      val gson = Gson()
       
      val json = gson.toJson(userCredentials)
       
       
      try {
       
      FileWriter(credentialsFile).use { writer -> writer.write(json) }
       
      Toast.makeText(this, "凭证已保存", Toast.LENGTH\_SHORT).show()
       
      } catch (e: IOException) {
       
      e.printStackTrace()
       
      Toast.makeText(this, "保存凭证失败", Toast.LENGTH\_SHORT).show()
       
      }
       
     }
   * **步骤**：
     1. 创建 UserCredentials 对象。
     2. 使用 Gson 将对象序列化为 JSON 字符串。
     3. 使用 FileWriter 将 JSON 字符串写入文件。
     4. 捕捉任何可能的 I/O 异常，确保程序不会崩溃。
3. **清空用户凭证**：
   * **方法**：clearUserCredentials()
   * **功能**：在用户取消勾选“记住密码”时，删除存储的凭证文件。
   * **实现**：
   * private fun clearUserCredentials() {
       
      if (credentialsFile.exists()) {
       
      credentialsFile.delete() // 删除凭证文件
       
      Toast.makeText(this, "凭证已清除", Toast.LENGTH\_SHORT).show()
       
      }
       
     }
   * **步骤**：
     1. 检查文件是否存在。
     2. 删除文件并显示确认消息。

### 3. 异常处理

在文件读写过程中，使用了 try-catch 块来处理可能的异常：

* 在读取凭证时，捕捉 JsonSyntaxException 以处理 JSON 解析错误。
* 在保存凭证时，捕捉 IOException 以处理文件写入错误。
* 这种异常处理机制确保了即使在出现错误时，应用也能优雅地处理，而不会崩溃。

### 4. UI 更新

每次加载或保存凭证后，应用会更新 UI 以反映当前的状态。例如，在加载凭证后，自动填充用户输入框；在保存凭证后，向用户显示成功保存的提示。



