产品报告

天涯咫尺

(实现聊天功能的 Mini App)

学　　校　：杭州电子科技大学

专　　业　：计算机科学与技术

学生姓名：　　黄江晔

目录

1. 产品功能介绍

天涯咫尺是一款即时通讯应用，旨在提供流畅、安全、方便的聊天体验。通过简洁直观的界面和丰富的功能，帮助用户轻松与朋友、家人及同事保持联系，随时随地进行沟通。

具体功能实现包括：登录功能、注册功能、实时消息传递功能、在线状态显示功能、个性化设置等功能。

2. 程序概要设计

2.1 系统架构设计与组成

该应用的系统架构基于客户端-服务器模型，其中客户端（Android App）与 Firebase 进行通信。Firebase 提供了数据库、实时消息传递和用户身份验证等服务。系统架构的设计旨在提供高效、可靠和可扩展的实时聊天功能，支持多个用户同时在线进行消息交流。

2.1.1 客户端（Android App）

（1）开发工具：使用 Android Studio 开发，采用 Java 编程语言。

（2）主要组件：

- UI 层：用于展示聊天界面，包括消息列表、输入框、发送按钮、聊天窗口等。
- 数据层：通过 Firebase SDK 与 Firebase 后端进行通信，处理数据的读取、写入、实时更新等操作。
- 消息处理：处理消息的发送、接收以及通知管理（如推送通知）。
- 用户认证：使用 Firebase Authentication 进行用户登录和注册。
- 实时数据监听：使用 Firestore 监听实时数据变化，更新 UI。

2.1.2 后端（Firebase）

（1）Firebase Authentication：提供用户认证和身份验证服务，确保每个用户都具有唯一的身份，并通过令牌控制访问权限。

（2）Firebase Firestore：

- 存储和管理聊天记录，包括消息、聊天室信息、用户资料等。
- 支持实时数据同步，确保消息能够即时传输到客户端。
- 数据的存储结构为 JSON 格式。

（3）Firebase Cloud Messaging (FCM)：用于实现推送通知，提醒用户有新消息。

2.2 功能模块设计

2.2.1 模块列表

- 用户注册与登录模块：处理用户的注册、登录和身份验证。
- 实时消息传递模块：实现消息的发送、接收、同步等功能。
- 聊天记录存储与管理模块：负责存储聊天记录和管理历史消息。

2.2.2 模块间关系

- 用户注册与登录模块为实时消息传递模块提供用户信息。
- 聊天记录存储与管理模块与实时消息传递模块相互依赖，以确保消息的持久化。

2.3 数据流程设计

- 用户数据流：用户信息首先通过 SignUpActivity 进行创建并上传到 Firebase Firestore，然后通过 SignInActivity 进行身份验证。成功登录后，用户信息被存储在 SharedPreferences 中，方便在不同页面间共享。

- 聊天数据流：聊天消息通过 ChatActivity 进行发送和接收，Firebase Firestore 实时更新会话数据，并展示在 MainActivity 和 ChatActivity 中。
- 会话数据流：每次聊天发生时，conversations 集合中的会话信息会更新，并保存最后一条消息。这个过程保证了用户始终可以看到最新的聊天会话。

2.3.1 用户注册（SignUpActivity）

（1）数据输入：用户提供姓名、电子邮件、密码、确认密码以及头像图片。

（2）数据验证：

- 检查所有字段是否为空。
- 检查邮箱是否有效。
- 确认密码与输入的密码一致。
- 确保头像已上传。

（3）数据处理：

- 将头像图片编码为 Base64 字符串。
- 将用户输入的所有信息（包括头像编码）作为一条记录上传到 Firebase Firestore 数据库。

（4）数据存储：用户信息（包括头像编码）存储到 users 集合中。

（5）跳转：注册成功后，跳转到 MainActivity，并将用户登录状态保存到本地 SharedPreferences 中。

2.3.2 用户登录（SignInActivity）

（1）数据输入：用户输入电子邮件和密码。

（2）数据验证：

- 检查邮箱是否有效。
- 检查密码是否为空。

（3）数据处理:检索 Firebase Firestore 中的用户数据，通过电子邮件和密码验证用户身份。

（4）数据存储：如果验证成功，将用户信息（如名称、头像、用户 ID 等）存储到本地 SharedPreferences 中。

（5）跳转：登录成功后，跳转到 MainActivity，并保存登录状态。

2.3.3 用户列表（UsersActivity）

（1）数据获取:从 Firebase Firestore 中的 users 集合获取所有用户的数据。排除当前用户，确保不显示自己。

（2）数据存储:将所有其他用户的信息（姓名、头像、ID 等）保存在本地列表中，并显示在 RecyclerView 中。

（3）点击事件：用户点击其他用户时，跳转到 ChatActivity 并传递用户信息。

2.3.4 主界面（MainActivity）

（1）数据加载：

- 加载当前登录用户的名称和头像。
- 获取当前用户的 FCM 推送令牌，并将其更新到 Firebase Firestore 中。
- 监听当前用户的会话数据（即与其他用户的消息）。

（2）数据存储:从 Firebase Firestore 中的 conversations 集合获取与当前用户相关的消息记录，并通过 RecyclerView 展示最近的会话。

（3）消息更新：

- 监听 Firebase Firestore 中的会话数据变化，实时更新会话列表。
- 如果有新的消息，更新会话记录中的最后一条消息，并在界面上实时显示。

2.3.5 聊天界面（ChatActivity）

（1）数据加载:

- 加载接收方用户的名称和头像信息。
- 显示历史聊天记录（如果有）。

（2）数据存储:

- 用户输入的消息会被上传到 Firebase Firestore 中的 chat 集合。
- 如果没有现有的会话记录，创建新的会话记录并存储到 conversations 集合。
- 每次发送消息时，更新 conversations 集合中的最后一条消息。

（3）实时更新:

- 使用 SnapshotListener 监听消息变化并实时更新聊天记录。
- 每次新消息到达时，将其添加到聊天记录中，并滚动到最新消息。

2.3.6 数据存储与管理

（1）Firebase Firestore:

- users 集合：存储所有用户的信息（包括姓名、邮箱、头像、FCM 令牌等）。
- chat 集合：存储每条聊天消息的记录（包括发送者、接收者、消息内容、时间戳等）。
- conversations 集合：存储每对用户之间的会话记录，包括最后一条消息、会话更新时间、参与者信息等。

（2）本地存储:

- 使用 SharedPreferences 存储当前用户的登录状态、用户 ID、姓名、头像等基本信息。
- 本地缓存用户信息和会话信息，以提高性能和用户体验。

数据流图示意：

1. **注册/登录**：
   - 用户输入注册/登录信息 → 校验信息 → 保存用户信息到Firebase → 更新本地 `SharedPreferences` → 跳转到主界面
2. **主界面**：
   - 加载用户数据（从Firebase读取） → 获取会话记录 → 显示最近会话
   - 监听会话更新（Firebase实时更新）
   - 点击用户 → 跳转到聊天界面
3. **聊天界面**：
   - 加载聊天记录（从Firebase读取）
   - 用户输入消息 → 发送消息 → 存储消息到Firebase → 更新会话记录
   - 监听新消息 → 实时更新聊天记录

2.4 数据库设计

（1）Users Collection：包含应用用户的基本信息，包括名字、头像、电子邮件、密码、FCM

Token（用于推送通知）和在线状态（availability）。Firebase Authentication 用于管理用户登录。

存储应用的用户信息，包括用户的 ID、姓名、电子邮件、密码、头像等。每个用户对应一个文档，文档 ID 为用户的唯一标识（例如，使用 Firebase Authentication 用户 ID 或其他唯一标识符）。

```json
users (Collection)
    └── userId (Document)
        ├── name: "John Doe"
        ├── email: "johndoe@example.com"
        ├── password: "hashed_password"
        ├── image: "base64_encoded_image"
        ├── fcmToken: "user_fcm_token"
        └── availability: 1  // 1 表示在线，0 表示离线
```

（2）Chats Collection：存储每一条具体的聊天记录。通过 senderId 和 receiverId 来区分消息的发送者和接收者。每一条消息都记录了消息内容、发送时间等信息。

存储实际的聊天消息。每条消息对应一个文档，记录了发送者、接收者、消息内容和时间戳等信息。

```json
chat (Collection)
    └── chatId (Document)
        ├── senderId: "userId1"
        ├── receiverId: "userId2"
        ├── message: "Hello, how are you?"
        ├── timestamp: "2024-12-20T10:15:00Z"
        └── type: "text"  // 消息类型：文本、图片等
```

- **字段说明**：
  - `senderId`：发送者的用户 ID。
  - `receiverId`：接收者的用户 ID。
  - `message`：消息内容。
  - `timestamp`：消息的发送时间，通常以 UTC 时间格式存储。
  - `type`：可选，指定消息的类型（如文本消息、图片消息等）。

（3）Conversations Collection：存储用户之间的对话概要信息，用于快速查询聊天列表。每个对话文档包含了发送者和接收者的详细信息（如姓名、头像）、最后一条消息的内容、最后更新时间和在线状态。

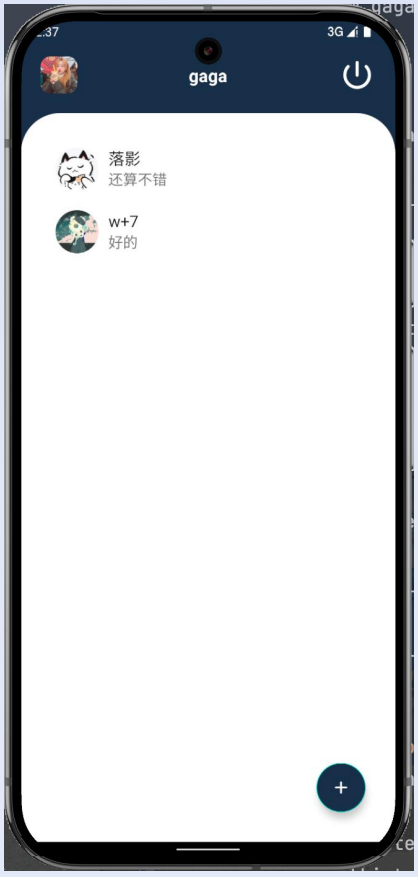
存储用户之间的对话信息，包括对话的基本信息（如发送者和接收者信息、最后一条消息等）。每个对话都有一个唯一的 `conversionId`。

```json
conversations (Collection)
    └── conversationId (Document)
        ├── senderId: "userId1"
        ├── senderName: "John Doe"
        ├── senderImage: "base64_encoded_image"
        ├── receiverId: "userId2"
        ├── receiverName: "Jane Smith"
        ├── receiverImage: "base64_encoded_image"
        ├── lastMessage: "Hey, let's catch up!"
        ├── timestamp: "2024-12-20T10:15:00Z"
        └── availability: 1  // 1 表示接收者在线，0 表示离线
```

- **字段说明**：
  - `senderId`：发送者的用户 ID。
  - `senderName`：发送者的姓名。
  - `senderImage`：发送者的头像（以 base64 编码的字符串存储）。
  - `receiverId`：接收者的用户 ID。
  - `receiverName`：接收者的姓名。
  - `receiverImage`：接收者的头像（以 base64 编码的字符串存储）。
  - `lastMessage`：对话中最后一条消息的内容。
  - `timestamp`：对话的最后更新时间，通常是最后一条消息的时间戳。
  - `availability`：表示接收者的在线状态，1 表示在线，0 表示离线。

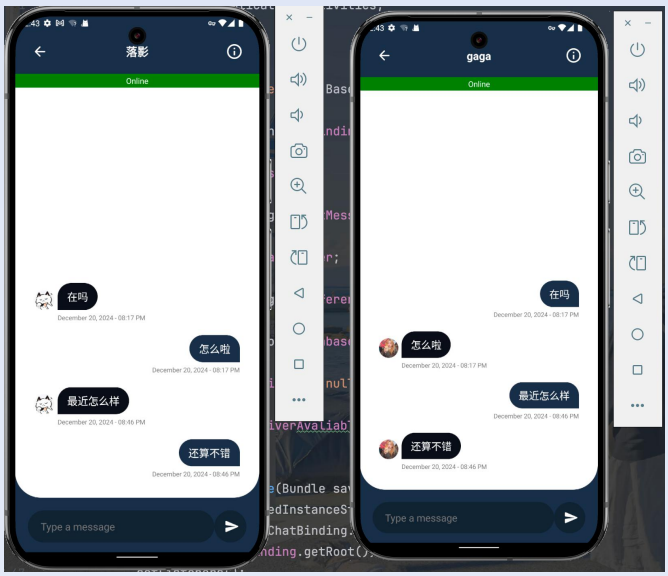
2.5 界面设计


Select User
w+7
2201748734@qq.com
落影
1535145283@qq.com
无关风月
1471499711@qq.com
超能炫彩蛋挞
1305927094@qq.com

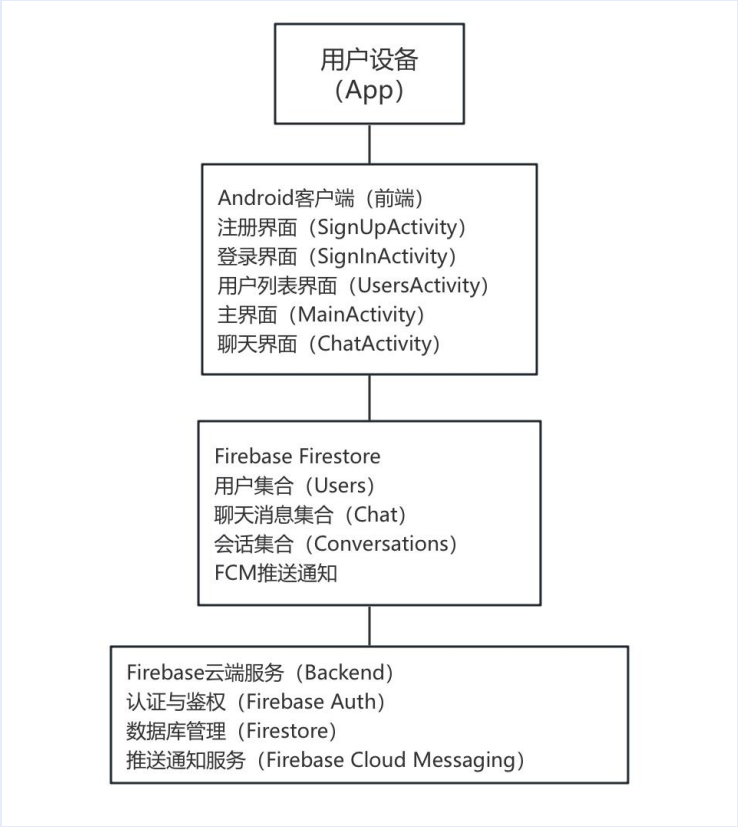

gaga
落影
还算不错
w+7
好的


w+7
你好
December 20, 2024 - 08:42 PM
你好啊
December 20, 2024 - 08:42 PM
食堂
December 20, 2024 - 08:43 PM
中午吃什么
December 20, 2024 - 08:43 PM
食堂
December 20, 2024 - 08:43 PM
好的
December 20, 2024 - 08:43 PM
Type a message


注册新账号
点击选择头像
昵称
邮箱
密码
确认密码
注册
登录



3. 软件架构图



3.1 用户设备（App）

客户端应用，运行在用户的手机上，提供所有的交互界面和功能。包含多个界面，如注

册、登录、用户列表、主界面、聊天界面等。

3.2 Android 客户端（前端）

运行在安卓设备上的应用层。它处理用户输入、界面展示、数据请求等。主要由不同的 Activity（例如：SignUpActivity、SignInActivity、UsersActivity、MainActivity、ChatActivity）构成，每个界面对应一个特定的功能。

3.3 Firebase Firestore（数据库）

后台数据库，存储所有应用数据。包含：

- 用户集合（Users）：存储用户信息（如姓名、邮箱、头像、token 等）。
- 聊天消息集合（Chat）：存储聊天记录，每条消息都包括发送者、接收者、消息内容、时间戳等。
- 会话集合（Conversations）：存储最近的对话信息，包括对话双方、最后一条消息、时间戳等。

3.4 Firebase 云端服务（Backend）

Firebase 提供的云端服务，包括：

- Firebase Auth：用于用户的认证和身份验证（例如，用户登录、注册、会话保持等）。
- Firestore：作为应用的数据库管理系统，保存和查询数据。
- Firebase Cloud Messaging (FCM)：用于推送通知服务，通知用户收到新的消息。

模块关系：

- Android 客户端（前端）与 Firebase Firestore 之间通过网络进行通信。客户端发起请求（如注册、登录、获取用户列表、获取聊天记录等），Firestore 返回数据。
- Firebase Auth 提供用户的身份验证和授权，确保只有经过认证的用户能够进行操作。
- FCM 提供实时通知功能，当有新消息时，系统会通过推送通知提醒用户。

3.5 更详细的模块交互

3.5.1 用户注册与登录（Firebase Auth）：

用户输入注册信息（如邮箱、密码等），通过 FirebaseAuth 进行身份验证，成功后将信息保存到 Firestore 的用户集合（Users）。登录过程类似，通过 FirebaseAuth 进行认证并获取用户信息。

3.5.2 获取用户列表：

用户登录后，客户端请求 Firestore 中的用户集合，获取所有其他用户的详细信息（排除当前用户自己）。

3.5.3 发送消息与获取消息：

客户端通过 Firestore 的聊天集合（Chat）发送和接收消息。消息存储在 Firestore 中，当有新消息时，应用会通过 SnapshotListener 监听数据变化，实时更新 UI。

3.5.4 会话管理：

聊天消息会在会话集合（Conversations）中存储，记录最后一条消息和对话双方。当用户启动聊天界面时，首先检查是否已经存在会话记录。

3.5.5 推送通知（FCM）：

当用户收到新消息时，应用会通过 FCM 发送推送通知，提醒用户有新消息。

4. 技术亮点及其实现原理

4.1 Firebase 实时数据库与云存储（Firestore）

（1）技术亮点：

实时数据同步：Firestore 提供了一个实时数据库，它允许客户端实时同步数据，确保多个设备之间的数据始终保持一致。这对于即时消息应用（如聊天应用）非常重要，确保用户在任意时刻看到的是最新的消息。

（2）实现原理：

Firestore 的实时监听器 (addSnapshotListener) 会监听特定文档或集合的变化。当数据发生变化时（例如，新的聊天消息添加到数据库），客户端会自动接收到通知，并更新界面，无需刷新或重载数据。

每次数据库中的数据变化时（如消息、会话更新），应用会自动通过 WebSocket 或持久连接同步这些变化，确保用户始终看到最新状态。

（3）关键点：

查询性能：Firestore 提供强大的索引机制，查询性能可以在大规模数据量下保持高效。

灵活的数据结构：你可以灵活地使用集合和文档来存储用户、消息和会话等数据。

4.2 Firebase Cloud Messaging (FCM) 推送通知

（1）技术亮点：

实时消息推送：FCM 可以用来发送推送通知，提醒用户有新消息或应用活动。这是大多数即时通讯应用的核心特性。

（2）实现原理：

FCM 提供了一个平台，无论用户是否打开应用，服务器都可以发送通知消息到用户设备。当用户在聊天应用中接收到新消息时，后端可以通过 FCM 向用户设备推送通知。即使应用处于后台，FCM 也能保证用户及时收到消息。

（3）关键点：

消息优先级：FCM 支持设置消息的优先级，比如“高优先级”消息可以在用户设备上立即推送，而“低优先级”消息可以延迟推送，减少不必要的资源消耗。

自定义通知：FCM 支持自定义推送通知的内容，包括标题、内容、图标和操作按钮等。

4.3 Firebase Authentication（身份验证）

（1）技术亮点：

简化的认证流程：通过 Firebase Authentication，用户可以使用多种方式（如电子邮件、Google 登录、Facebook 登录等）轻松完成认证，且无需自己实现复杂的身份验证逻辑。

（2）实现原理：

Firebase Authentication 提供了一个简单的 SDK，它将用户的认证流程抽象化，支持密码登录、短信验证码登录、OAuth 登录等多种方式。

当用户登录后，Firebase 会创建一个认证会话，客户端可以通过 Firebase 提供的 API 获取用户的身份信息（如 UID、电子邮件、头像等），并将这些信息存储在应用的本地存储中，以便后续使用。

（3）关键点：

安全性：Firebase Authentication 提供了内建的安全机制，如验证码、邮箱验证等，确保身份验证过程的安全。

跨平台支持：支持 Android、iOS 和 Web 平台的统一身份验证流程，使得跨平台应用可以使用相同的认证系统。

4.4 基于事件驱动的架构设计

（1）技术亮点：

数据变更实时响应：应用内的每个操作（如发送消息、接收消息）都通过监听器和事件驱动方式触发。这种方式使得系统具备了实时响应能力，适用于即时通讯场景。

（2）实现原理：

Firestore 和 FCM 都是事件驱动的系统。Firestore 的实时监听器监听数据变化事件，当数据库中的数据发生变化时，自动推送更新至客户端。FCM 则通过云服务向设备推送通知，告知用户新的事件发生（如新消息）。

在聊天过程中，客户端不断监听用户在线状态和聊天消息的变化，这些事件通过 Firestore 的监听器（addSnapshotListener）和 FCM 通知（send()）机制驱动。

（3）关键点：

无轮询：由于采用了实时监听和推送通知，应用无需通过传统的轮询方式获取数据，从而大大减少了网络带宽和延迟。

高效的用户体验：即时响应、无需等待或刷新，提升用户体验，尤其适用于需要实时反馈的应用，如聊天、股票监控等。

4.5 Base64 编码与图像处理

（1）技术亮点：

头像存储与传输：应用使用 Base64 编码将图像转换为字符串，以便在 Firebase Firestore 中存储和传输用户头像。这种方式避免了直接存储文件而导致的存储空间占用，适合存储较小的图像。

（2）实现原理：

用户上传头像时，应用将图像转换为 Base64 字符串并将其存储在 Firestore 数据库中。

接收头像时，应用通过 Base64 解码将图像恢复为 Bitmap 格式进行显示。

这种方式虽然适用于小型图像的存储和传输，但对于大图像可能会有性能瓶颈，因为 Base64 编码后的数据量是原始图像的约 33%。

（3）关键点：

便于存储和传输：Base64 编码将图像转化为纯文本格式，便于在数据库和网络中传输。

适用于小图像：虽然 Base64 编码非常方便，但它对于大图像或文件并不是最佳解决方案，可能需要采用更高效的文件存储和传输方案（如 Firebase Storage）。

4.6 RecyclerView 与数据适配器的高效显示

（1）技术亮点：

高效的列表显示：应用使用 RecyclerView 和 Adapter 来高效地显示聊天消息和用户列表。RecyclerView 提供了高效的视图复用和滚动优化，适合大数据量场景。

（2）实现原理：

RecyclerView 使用了视图复用机制（通过 ViewHolder），当列表项滑出视图时，相应的视图会被回收，避免了频繁的视图创建和销毁，提高性能。

适配器模式使得界面数据和视图显示解耦，方便管理数据源与视图的绑定。

（3）关键点：

分页加载：对于消息列表较长的聊天记录，RecyclerView 可以与分页技术结合，通过分页加载提高性能。

平滑滚动：结合 smoothScrollToPosition 方法，保证消息列表的顺畅体验，避免因为大量数据加载导致界面卡顿。

4.7 跨平台支持与灵活扩展

（1）技术亮点：

Firebase 跨平台能力：通过 Firebase 提供的 SDK，可以将同一个后端服务应用到 Android、iOS 以及 Web 平台上，极大地减少开发和维护的成本。

（2）实现原理：

Firebase 提供了跨平台的支持，使得即便你的应用需要跨多个平台（如 Android 和 iOS）运行，后台服务依然是统一的。

前端可以共享同样的 Firestore 数据库，用户在一个平台上发送的消息，能够实时同步到其他平台。

（3）关键点：

统一的认证和数据管理：无论平台如何变化，Firebase 提供的认证和数据库服务始终保持一致，减少了多平台支持的复杂度。

实时数据共享：确保所有平台的用户都能即时获取最新数据，提高了系统的可扩展性。