# 移动互联网技术及应用

# 大作业报告

# 题目： 北邮 70 周年校庆智能体助手系统的设计与实现

类型：应用系统设计实现

| 姓名 | 班级 | 学号 |
| --- | --- | --- |
| 张晨阳 | **2022211305** | **2022211683** |

**2025.5**

# 目录

# 1. 相关技术

## 1.1. 开发平台与语言选择

本系统的开发平台选择为 Android Studio Meerkat | 2024.3.1 版本，开发语言为 Kotlin，并采用 Empty Activity + Jetpack Compose 模板作为项目起始结构。Android Studio 作为官方推荐的 Android 应用开发环境，提供了完善的调试工具、模拟器支持和便捷的 Gradle 项目管理功能，适用于本次移动互联网系统开发。

Kotlin 是 Google 推出的 Android 首选编程语言，具有简洁、安全、现代化等特性，能显著减少样板代码（boilerplate）并提升开发效率。结合 Jetpack Compose，Kotlin 能实现响应式 UI 构建机制，使界面逻辑更为清晰灵活，特别适合页面切换较多、状态变化频繁的应用系统。

在整个开发过程中，Kotlin 提供了良好的空值安全（Null Safety）、数据类（data class）与协程（Coroutines）机制，为异步网络请求、动画控制、状态管理等功能提供了稳定的基础。

因此，选择 Kotlin 语言与 Android Studio 平台，不仅能够快速构建现代化 Android 应用，还能提升开发与调试的效率，为整个系统的实现打下了坚实的技术基础。

## 1.2. Jetpack Compose UI

本系统采用 Jetpack Compose 作为主要的界面开发框架。Jetpack Compose 是 Google 推出的现代声明式 UI 工具包，已逐步成为 Android 应用开发的主流方式。它通过 Kotlin 编写 UI 组件，抛弃了传统基于 XML 的布局机制，使界面代码更加紧凑、灵活、响应式。

与传统 View 系统相比，Jetpack Compose 具有以下优势：

- 声明式界面构建：开发者只需描述 UI 应该“看起来如何”，无需关心界面如何随状态变化而更新，大大简化了状态驱动界面的实现。
- 状态管理便捷：结合 remember 和 mutableStateOf，可以轻松实现界面状态

（如导航栏选项、弹幕内容等）与 UI 的联动。

- 组件复用性强：Compose 中的每一个 @Composable 函数都可被独立调用和组合，方便模块化开发和测试。
- 更少的模板代码：通过 Kotlin 的语法特性，Compose 避免了大量 XML 文件与 findViewById() 操作，减少了错误发生的概率。

在本系统中，Jetpack Compose 被应用于以下模块的 UI 构建：

- **首页活动列表：**基于 LazyColumn 快速渲染活动信息，并结合 Card、Text 等控件进行样式美化。
- **弹幕留言墙：**通过 Box 和 Animatable 动画组件实现弹幕的随机轨迹展示。
- **地图展示页：**使用 Image 搭配缩放手势响应，实现地图的查看与缩放控制。
- **底部导航栏：**使用 NavigationBar 和 NavigationBarItem 构建切换式页面布局。

总之，Jetpack Compose 为本项目提供了更现代、更简洁的 UI 构建能力，大幅提升了页面开发效率和界面动态响应能力，适配了多模块、高交互场景下的需求，是本系统不可或缺的核心技术之一。

## 1.3. 网络通信技术

为了实现智能助手模块中与大模型 API 的对接，本系统采用了 HTTP 网络通信机制，并使用 OkHttp 库作为底层通信框架来完成请求发送与响应处理。

OkHttp 是广泛用于 Android 开发的网络库，具备如下优势：

- 支持异步/同步请求；
- 支持连接池和缓存机制；
- 提供灵活的 Request 和 Response 构造方式；
- 与 Kotlin 协程兼容性良好。

在本项目中，OkHttp 被用于构建与 Kimi API 的对话请求，包括设置请求头、构建 JSON 请求体以及处理模型返回的数据结构。

### 请求构建与发送过程

系统通过以下步骤完成与远程模型的通信：

1. 使用 JSONObject 构造消息体内容，包括模型名称、温度参数、用户输入内容等；
2. 设置请求头；
3. 通过 Request.Builder() 创建 POST 请求；
4. 调用 OkHttpClient.newCall(request).enqueue(...) 进行异步网络请求；
5. 在 onResponse() 中解析响应 JSON 数据，提取模型返回的聊天内容。

该过程全程运行于子线程，符合 Android 对网络请求的线程限制要求，避免了主线程阻塞导致 UI 卡顿问题。

### 错误处理机制

系统在请求过程中添加了如下基本错误处理：

- 请求失败（如无网络）时，通过 onFailure() 返回 null，并提示用户“出错了，请稍后再试”；
- 若服务器响应内容异常，使用 try-catch 结构捕获 JSON 解析异常，避免程序崩溃；
- 增加日志打印（Logcat）用于开发阶段调试网络状态与返回内容。

通过上述网络通信机制的设计与实现，系统成功实现了智能助手模块与大模型服务之间的低延迟交互，为校庆场景下的自然语言问答提供了可靠支撑。

## 1.4. Kimi API

在本系统的智能问答模块中，我们接入了由 Moonshot 推出的 Kimi 大语言模型 API（接口地址为 https://api.moonshot.cn/v1/chat/completions），实现了一个基于自然语言处理的聊天助手，帮助用户在校庆活动中快速获得问询解答。

Kimi 是一款通用型大语言模型，具备良好的中文语义理解与生成能力，尤其在中文领域表现稳定，支持高并发、长文本、多轮对话等应用场景，适合部署于校园活动信息问答类场合。

本系统使用的模型版本为 moonshot-v1-32k，支持最大 32k token 长度的上下文输入，确保即便用户连续提问也能维持上下文一致性。

## 请求结构

系统构造的 JSON 请求体主要包含以下字段：

```
{
  "model": "moonshot-v1-32k",
  "messages": [
    {
      "role": "user",
      "content": "校庆什么时候开始？"
    }
  ],
  "temperature": 0.3
}
```

- "model"：指定调用的模型版本；
- "messages"：多轮对话上下文，用户提问内容放在 "content" 字段中；
- "temperature"：控制生成结果的随机性（值越高回答越灵活，越低越稳重）。

## 响应解析

API 的返回结构中包含模型生成的 "choices" 数组。系统在 onResponse() 回调中使用 JSONObject 提取该内容：

```
val choices = JSONObject(responseBody).getJSONArray("choices")
val reply = choices.getJSONObject(0).getJSONObject("message").getString("content")
```

返回结果被回传至 Compose UI 中用于展示，同时结合本地状态管理进行会话更新。

虽然 Kimi 模型具备强大的问答能力，但为了适配校园校庆场景，系统还通过“提示工程”进行定制化。

通过集成 Kimi API，系统显著提升了人机交互能力，为校庆活动参与者提供了高效、自然、智能的服务体验。

# 2. 系统功能需求

## 2.1. 系统背景与设计目标

### 背景介绍

2025 年是我校建校的 70 周年纪念年份，为了更好地组织和展示校庆期间的系列活动，提升校友与师生的参与感和归属感，亟需一个集信息展示、互动留言与智能问答于一体的校园节日活动辅助工具。

传统的线下海报、纸质日程等方式信息更新不及时、互动性弱，而基于移动互联网的应用程序具备即时性、便携性与可扩展性，能够更好地满足现代校园活动管理与服务的需求。

### 设计目标

本系统旨在设计并实现一个基于 Android 平台的“校园节日活动智能体助手”应用，主要目标如下：

- **活动信息展示**：提供清晰、统一的校庆活动日程、地点与简介，方便师生与校友了解整体安排；
- **智能问答服务**：接入大语言模型 API，支持用户通过自然语言提问，快速获得关于活动、校史、地点等相关信息；
- **互动留言墙**：用户可发送祝福语，并通过弹幕动画形式实时展示，增强节日氛围；
- **校园地图浏览**：提供校区地图图像，支持手势缩放，便于校友定位校内活动场所；
- **离线数据存储**：不依赖后端服务器，使用本地文件进行数据保存，保证系统简单、部署方便；
- **UI 体验友好**：采用 Jetpack Compose 构建界面，提升视觉一致性与响应速度，适应移动设备交互需求。

通过上述功能设计与技术手段，本系统不仅为校庆活动提供了便利的数字化辅助工具，也展示了现代移动互联网技术在校园场景中的实际应用潜力。

## 2.2. 主要用户角色与使用场景

为了确保系统设计贴近真实需求，在开发过程中我们对目标用户进行了分类，并基于典型场景进行了交互流程设计。以下是本系统面向的主要用户角色及其典型使用场景。

1. **在校学生**
   - 希望通过移动设备快速获取校庆活动的安排与更新；
   - 希望向母校表达祝福、观看弹幕留言氛围；
   - 在校内可能因地理不熟或信息缺失，希望通过智能问答或地图快速定位活动地点。
2. **校友返校人员**
   - 关注校庆重要活动时间、地点；
   - 希望了解往届校友聚会安排、活动内容；
   - 希望通过留言墙向师长、母校表达情感。
3. **教职员工**
   - 配合组织方发布活动信息，提醒相关人员出席；
   - 了解整体活动安排，方便调配资源；
   - 通过系统解答参会人员的常见问题。
4. **活动组织者/志愿者**
   - 使用本系统进行日程确认、临时通知；
   - 利用问答助手及时回复现场参会者的提问；
   - 通过留言功能收集校友对活动的反馈。

| 场景 | 使用场景说明 |
| --- | --- |
| S1 | 校友抵达校园，打开 APP 查询“校友见面会”时间和地点。 |
| S2 | 学生在宿舍中输入“文艺汇演几点开始”并通过智能问答获取答案。 |
| S3 | 教师通过留言墙查看学生们发送的祝福，感受校庆氛围。 |
| S4 | 志愿者协助家长使用地图模块缩放查看校区结构，寻找礼堂位置。 |
| S5 | 组织者添加活动更新，学生点击首页后立即看到新安排。 |

## 2.3. 功能需求分析

基于前文提出的系统目标与用户场景分析，本系统需满足如下功能性与非功能性需求：

### 功能性需求

**1) 活动信息展示**

- 系统在“首页”模块展示校庆各项活动的详细信息，包括标题、时间、地点与简介；
- 所有活动信息从本地 JSON 文件读取，支持离线查看。

**2) 智能问答助手**

- 用户可进入“助手”模块，通过文本输入自然语言问题；
- 系统调用 Moonshot 的 Kimi 大语言模型 API 实现问答功能；
- 模型返回内容后在对话区域中呈现，模拟即时交互。

**3) 留言弹幕墙**

- 用户在“留言”模块可输入祝福语；
- 所有留言通过本地 JSON 文件持久化存储；
- 系统实时从留言数据中选取内容，以弹幕动画形式在屏幕中横向滚动显示，形成动态展示氛围；
- 支持留言循环播放，保持页面持续热度。

**4) 校园地图展示**

- 在“地图”模块展示一张校园地图图像；
- 支持手势缩放（双指放大/缩小），便于用户查看不同区域；
- 地图位置固定，防止偏移影响视觉体验。

**5) 本地数据存储**

- 系统不依赖网络数据库，所有数据（活动信息、留言内容）均通过 filesDir 下的 JSON 文件进行读写；
- 数据初始化时注入默认祝福语和活动数据，保证首次打开系统时已有展示内容。

**6) 多页面导航与一致性界面**

- 底部导航栏包含四个页面：首页、助手、留言、地图；
- 所有页面使用 Jetpack Compose 构建，统一风格，响应式设计；
- 页面切换流畅，支持状态记忆与 UI 适配。

## 非功能性需求

- 易用性：界面简洁，交互直观，适配不同年龄段用户；
- 响应性：页面切换流畅，问答响应及时；
- 可维护性：模块结构清晰，逻辑与视图分离，便于后期扩展；
- 适配性：适配主流 Android 手机屏幕尺寸；
- 安全性：敏感信息（如 API Key）不硬编码于版本控制中，避免泄露风险（本地开发阶段可留作 placeholder）；
- 离线可用性：除问答模块需联网，其余页面支持无网络访问，保障校内场景适用性。

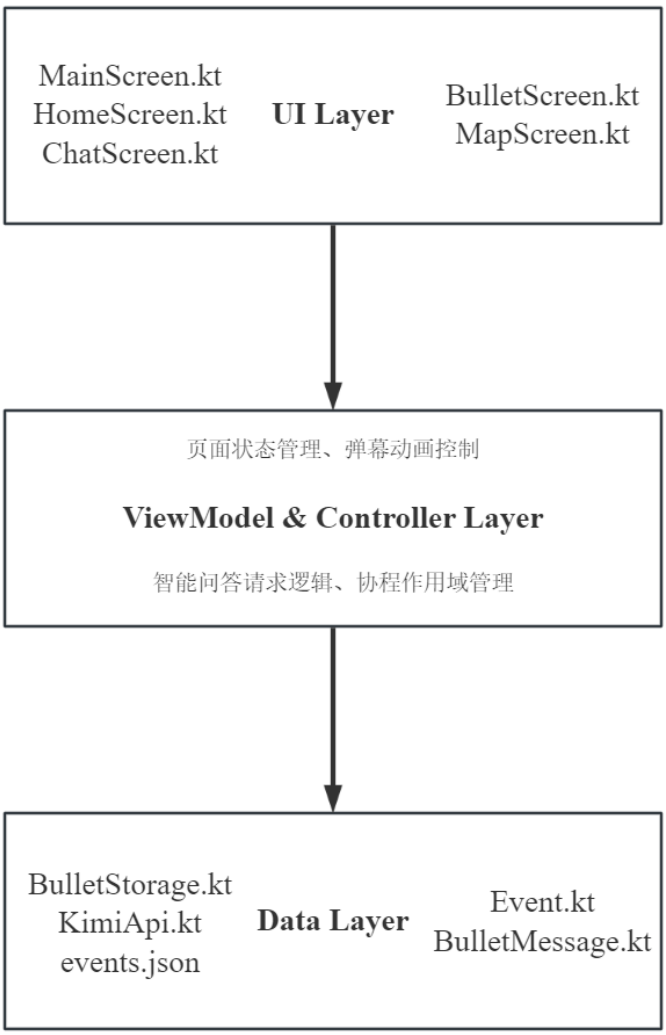
# 3. 系统设计与实现

## 3.1. 总体架构设计

本系统采用模块化设计思想，基于 Kotlin 编程语言与 Jetpack Compose UI 框架构建，整体结构清晰，逻辑分层明确，便于功能的开发与维护。

### 3.1.1.系统架构分层

整个系统分为以下三个主要层：



- **界面层（UI Layer）**
    - 使用 Jetpack Compose 构建所有页面（如首页、助手、留言、地图）；
    - 页面通过 MainScreen 中的 BottomNavigationBar 进行统一导航管理；
    - 提供用户与系统交互的直观入口，并负责状态展示与响应。
- **逻辑层（ViewModel & Controller Layer）**
    - 各页面的用户交互与状态管理通过 @Composable 状态管理机制与协程控制；
    - 简单的逻辑处理如弹幕发送、切换状态等在 Composable 内部完成；
    - 外部 API 调用与异步处理通过封装的工具类和作用域进行解耦管理。
- **数据层（Data Layer）**
    - 活动信息（Event）、留言（BulletMessage）等结构体定义在 model 包中；
    - 数据的读取与存储通过 storage 包中对应的 JSON 文件读写方法实现；
    - 智能问答请求则由 network 包中的 KimiApi.kt 管理，统一封装 HTTP 请求。

### 3.1.2.页面导航与模块协作

系统通过 MainActivity.kt 调用 MainScreen() 启动应用主页面，底部导航栏控制四个主要功能模块页面的切换，每个模块内部自成体系，各自处理逻辑与 UI，保持松耦合关系。模块间的数据通过状态变量传递或读取本地文件实现，避免全局变量依赖。

### 3.1.3.本地数据设计理念

考虑到系统部署与运行环境的简单性需求，本系统未使用网络数据库或远程后端服务，而是采用 Android 提供的 filesDir 本地文件目录进行 JSON 文件存取。这种设计具有如下优点：

- 简单易部署，脱离网络环境也可正常使用；

- JSON 文件格式清晰，便于调试与维护；
- 数据持久化控制在本地，降低数据丢失风险。

### 3.1.4.第三方服务调用

为实现更具交互性的“智能问答”功能，系统集成了 Moonshot 提供的 Kimi 大模型 API。请求过程封装在 KimiApi.kt 中，用户输入消息后通过 OkHttp 异步发送 HTTP POST 请求，并将模型响应展示于界面中，模拟一个智能体助手的体验。

## 3.2. 系统模块划分与实现

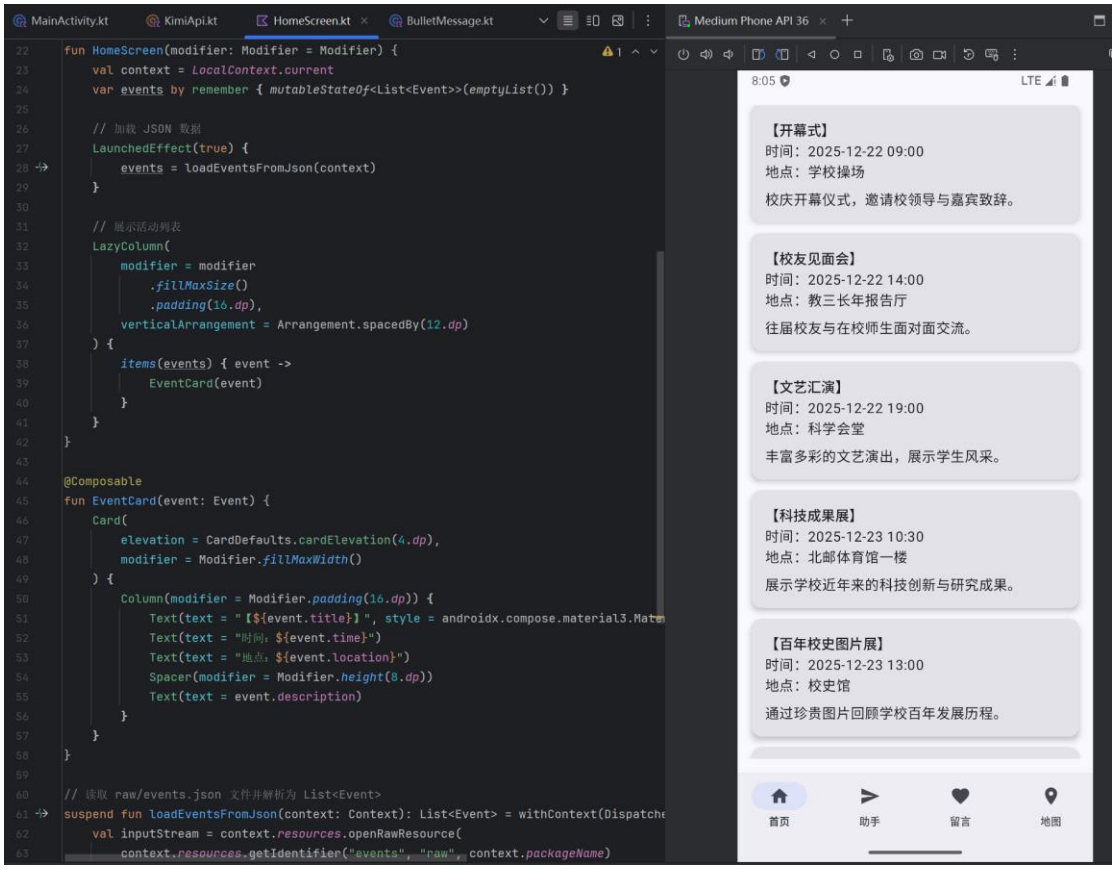
### 3.2.1.首页模块

首页模块是用户进入应用后首先看到的界面，其主要功能是展示节日活动信息列表，包括每个活动的标题、时间、地点与简介，便于用户快速了解校庆的安排与亮点。

**（1）功能实现概述**

首页模块采用 Jetpack Compose 构建 UI，使用 LazyColumn 组件实现活动列表的高效展示。每一项活动使用 Card 卡片方式展示，增强视觉层次感与可读性。活动数据来源于本地 res/raw/events.json 文件，通过 Context.resources.openRawResource() 加载并解析为模型类 Event。

**（2）界面效果示意**




**（3）主要组件与逻辑结构**

- Event 数据类定义如下，用于描述活动的核心信息：

```
data class Event(
```

```
    val title: String,
    val time: String,
    val location: String,
    val description: String
)
```

- 首页使用 HomeScreen.kt 文件实现核心逻辑：

```
@Composable
fun HomeScreen(modifier: Modifier = Modifier) {
    val context = LocalContext.current
    var events by remember { mutableStateOf<List<Event>>(emptyList()) }

    // 加载 JSON 数据
    LaunchedEffect(true) {
        events = loadEventsFromJson(context)
    }

    // 展示活动列表
    LazyColumn(
        modifier = modifier
            .fillMaxSize()
            .padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(12.dp)
    ) {
        items(events) { event ->
            EventCard(event)
        }
    }
}
```

**（4）数据加载方法**

- 事件数据通过以下方式从本地 JSON 文件中加载：

```
// 读取 raw/events.json 文件并解析为 List<Event>
suspend fun loadEventsFromJson(context: Context): List<Event> =
withContext(Dispatchers.IO) {
    val inputStream = context.resources.openRawResource(
        context.resources.getIdentifier("events", "raw",
context.packageName)
    )
    val reader = BufferedReader(InputStreamReader(inputStream))
    val jsonString = reader.readText()
    reader.close()

    val jsonArray = JSONArray(jsonString)
```

```
    val result = mutableListOf<Event>()
    for (i in 0 until jsonArray.length()) {
        val obj = jsonArray.getJSONObject(i)
        result.add(
            Event(
                title = obj.getString("title"),
                time = obj.getString("time"),
                location = obj.getString("location"),
                description = obj.getString("description")
            )
        )
    }
    result
}
```

### 3.2.2.智能助手模块

智能助手模块通过接入大模型接口，为用户提供智能问答服务。该模块的设计目标是：让用户能够就校庆相关问题与 AI 助手进行自然语言交互，获取实时、个性化的回复。

**（1）功能概述**

- 支持用户输入文本消息并发送；
- 后端调用 Moonshot 提供的 Kimi 大模型 API ；
- 将助手回复显示在对话界面中；
- 聊天记录按时序排列展示；
- 网络请求异步执行，保持界面响应。

**（2）界面效果示意**

### （3）数据模型设计

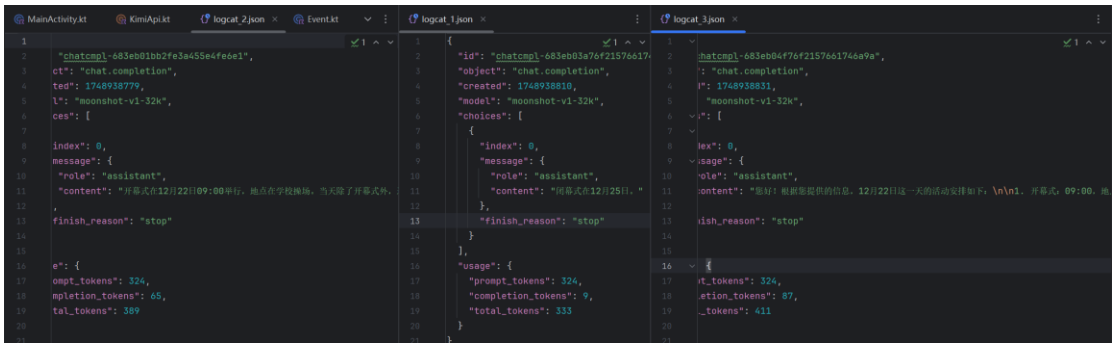
模块中使用 ChatMessage.kt 定义单条消息的数据结构：

```
// 表示一条聊天消息，区分是用户说的还是 AI 回复
data class ChatMessage(
    val sender: Sender,
    val content: String
)

enum class Sender {
    USER, BOT
}
```

该结构便于前端 UI 区分“用户输入”与“AI 回复”的消息样式。

此处展示出上图问答，kimi 返回的内容结构：




### （4）网络请求封装

智能助手模块中网络请求通过 KimiApi.kt 封装，使用 OkHttp 发起 POST 请求。

### （5）界面实现

模块核心界面在 ChatScreen.kt 中实现。

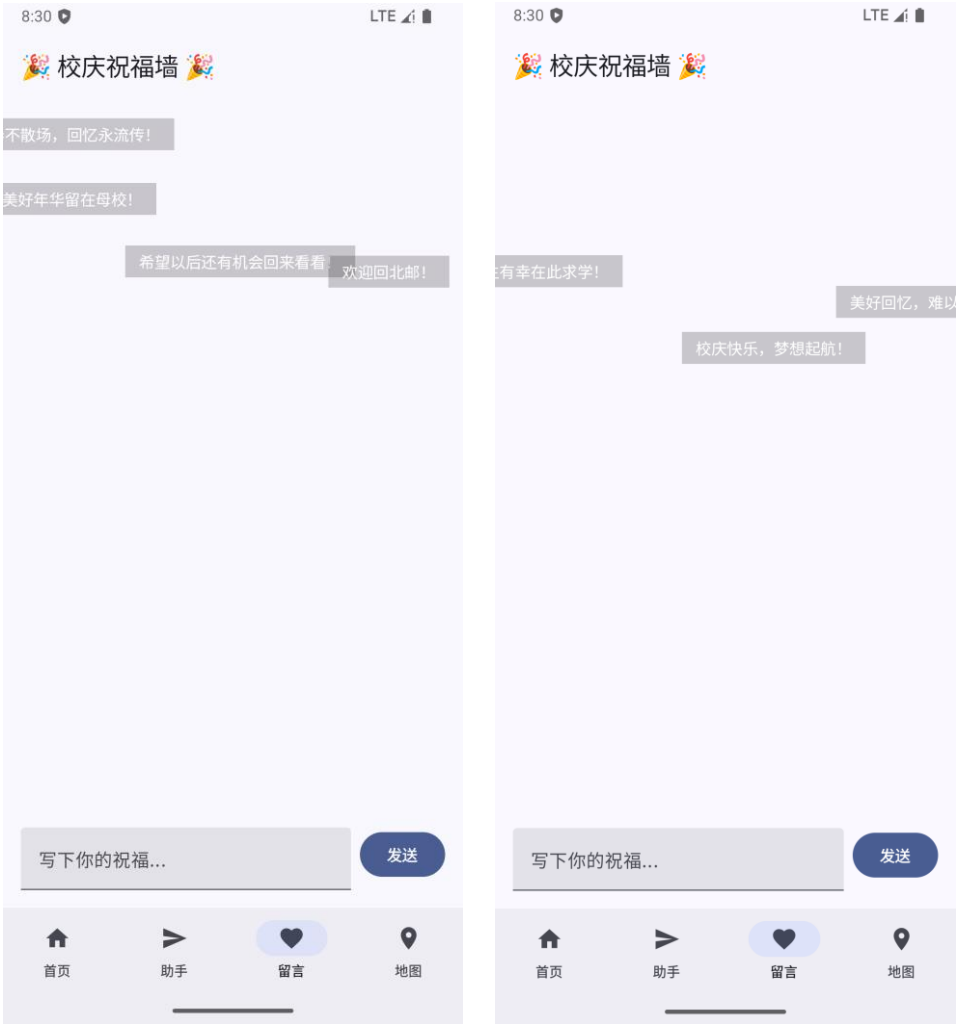
### 3.2.3.留言模块

留言模块（又称“祝福墙”或“弹幕墙”）用于展示校友、在校师生对母校的祝福。用户可通过输入框提交祝福语，这些祝福将以“弹幕”的形式在屏幕上动态循环播放，增强校庆的氛围与互动性。

**（1）功能描述**

- 用户可自由输入祝福内容并点击发送；
- 所有祝福将保存在本地 JSON 文件中；
- 页面自动加载历史祝福并循环播放；
- 弹幕内容以动画方式横向漂浮，增加趣味性；
- 支持多条弹幕同时展示，随机高度与速度；
- 初次运行时自动注入一批预设祝福语。

**（2）界面效果示意**



页面上半部分为弹幕播放区域，下半部分为输入框与发送按钮。

**（3）数据模型设计**

留言模块的数据结构定义在 BulletMessage.kt：

```
data class BulletMessage(
    val content: String,
    val timestamp: Long
)
```

**4）数据存储与加载**

采用文件存储方式，在内部存储路径中创建 bullet_messages.json 文件，相关操作封装于 BulletStorage.kt。

**（5）弹幕动画实现**

在 BulletScreen.kt 中使用 LaunchedEffect + Animatable 实现弹幕循环动画。

弹幕不断从右向左漂移，漂移结束自动移除，并通过 LaunchedEffect 不断注入新弹幕，形成循环播放效果。

## 3.2.4.地图模块

地图模块用于展示学校校园平面图，便于校友或访客快速定位校庆相关活动的举办场所，如开幕式地点、文艺汇演场地等。本模块通过加载本地图片资源并结合 Compose 的图像缩放手势支持，实现简单直观的互动式地图浏览功能。

**（1）功能描述**

- 展示一张校园地图图片；
- 支持双指缩放与手势操作；
- 地图始终居中显示；
- 图片资源打包在 res/drawable 文件夹中，无需联网加载，确保离线可用。

**（2）界面效果示意**


地图占据页面主体位置，支持手势放大缩小。

**（3）图片资源说明**

本地图图片保存在路径：

```
res/drawable/campus_map.jpg
```

**（4）主要实现代码**

地图模块逻辑定义在 MapScreen.kt 文件中，使用 rememberTransformableState 与 Modifier.graphicsLayer 实现缩放功能。

## 3.3. 主要数据模型设计

本系统的数据模型主要围绕两类核心内容构建：节日活动信息与用户留言信息。数据模型使用 Kotlin 的 data class 构建，具备简洁、可序列化、易于维护等优点，同时与 Jetpack Compose UI 与本地文件 I/O 实现之间配合紧密，为系统的各模块提供了结构清晰的数据支持。

### 3.3.1.活动数据 Event.kt

该数据类用于描述首页中展示的每一条活动内容。包括活动标题、举办时间、地点以及简要介绍。其结构定义如下：

```
data class Event(
    val title: String,
    val time: String,
    val location: String,
    val description: String
)
```

活动数据通常以 JSON 文件（如 res/raw/events.json）的形式进行配置，程序启动时统一加载，避免硬编码内容，使得系统具备更好的可维护性与扩展性。

示例 JSON 数据如下：

```
[
  {
    "title": "开幕式",
    "time": "2025-12-22 09:00",
    "location": "学校操场",
    "description": "校庆开幕仪式，邀请校领导与嘉宾致辞。"
  },
  {
    "title": "校友见面会",
    "time": "2025-12-22 14:00",
    "location": "教三长年报告厅",
    "description": "往届校友与在校师生面对面交流。"
  },
  {
    "title": "文艺汇演",
    "time": "2025-12-22 19:00",
    "location": "科学会堂",
    "description": "丰富多彩的文艺演出，展示学生风采。"
  },
  {
    "title": "科技成果展",
    "time": "2025-12-23 10:30",
    "location": "北邮体育馆一楼",
    "description": "展示学校近年来的科技创新与研究成果。"
  },
  {
    "title": "百年校史图片展",
```

数据通过 JSONArray + JSONObject 解析后转为 List<Event>，最终在 Compose UI 中通过 LazyColumn 列表动态渲染展示。

### 3.3.2.留言数据 BulletMessage.kt

该数据类用于描述留言墙模块中每一条用户祝福语。字段设计简洁，便于存取与播放动画。

```
data class BulletMessage(
    val content: String,
    val timestamp: Long
)
```

留言数据通过本地文件 bullet_messages.json 存储在 app 的私有目录中（context.filesDir），应用首次启动时会写入初始批量留言，后续用户发送的新留言也会追加保存，实现数据持久化。

留言数据会不断地被系统读取，并以弹幕形式随机循环播放，活跃祝福氛围，增强节日参与感。

## 3.4. 核心逻辑实现

### 3.4.1.活动信息加载逻辑

在系统的首页模块中，活动信息的展示依赖于从本地 events.json 文件中读取并动态加载数据。该文件存储在 Android 工程的 res/raw 目录中，以 JSON 格式描述各项活动的标题、时间、地点与简介等内容。

#### 数据读取流程

系统在 HomeScreen.kt 中使用 LaunchedEffect + remember 的组合，在首次进入首页时异步加载数据，并更新 Compose 界面的状态。

以下是读取流程的关键步骤：

1. 通过 context.resources.openRawResource(...) 获取 events.json 输入流；
2. 使用 BufferedReader 读取内容并转换为字符串；
3. 将字符串解析为 JSONArray，逐项转化为 Event 数据类对象；
4. 使用 mutableStateOf 与 Compose 状态管理，驱动 LazyColumn 展示更新。

#### 核心代码片段如下：

```
LaunchedEffect(true) {
    events = loadEventsFromJson(context)
}

suspend fun loadEventsFromJson(context: Context): List<Event> =
withContext(Dispatchers.IO) {
    val inputStream = context.resources.openRawResource(
        context.resources.getIdentifier("events", "raw",
context.packageName)
    )
    ...
    val jsonArray = JSONArray(jsonString)
}
```

#### UI 展示逻辑

在 Compose 界面中，我们使用 LazyColumn 和 Card 组件对每一条活动数据进行展示。每条活动以卡片形式呈现，内含活动标题、时间、地点与描述信息。排版整洁、交互友好。

### 3.4.2.智能问答通信逻辑

在本系统的“智能助手”模块中，我们通过接入 Moonshot 提供的类 ChatGPT 模型服务（即 Kimi API），实现了用户提问、模型理解与智能回复的问答交互体验。

#### 技术选型与实现方式

通信模块基于 OkHttp 实现异步 HTTP 请求，通过发送用户输入与系统提示词（System Prompt）到远程接口，接收结构化 JSON 响应后进行解析展示。

#### 系统提示词与上下文构建

为实现定制化的校庆问答体验，我们在每一次会话中，构造一个带有“系统提示词”的消息上下文。该提示词明确告知模型本助手的角色与可回答的内容范围，包括活动时间、地点与安排。例如：

```
{
  "role": "system",
  "content": "你是北京邮电大学 70 周年校庆活动的智能助手。以下是活动安排……"
}
```

随后，用户输入内容作为 "user" 消息追加，从而形成一段具备明确上下文的消息数组 messages。

#### 请求构造逻辑

请求数据采用 JSON 格式，包括模型类型、上下文消息与温度（随机性）等参数。使用 OkHttpClient 提交 POST 请求，关键代码如下：

```
val mediaType = "application/json".toMediaType()
val body = json.toString().toRequestBody(mediaType)

val request = Request.Builder()
    .url(ENDPOINT)
    .header("Authorization", "Bearer $API_KEY")
    .header("Content-Type", "application/json")
    .post(body)
   .build()
```

并通过异步 enqueue 方式发送，避免阻塞 UI 主线程：

```
client.newCall(request).enqueue(object : Callback {
    override fun onFailure(call: Call, e: IOException) { ... }
```

```
    override fun onResponse(call: Call, response: Response) { ... }
})
```

## 响应处理与错误日志

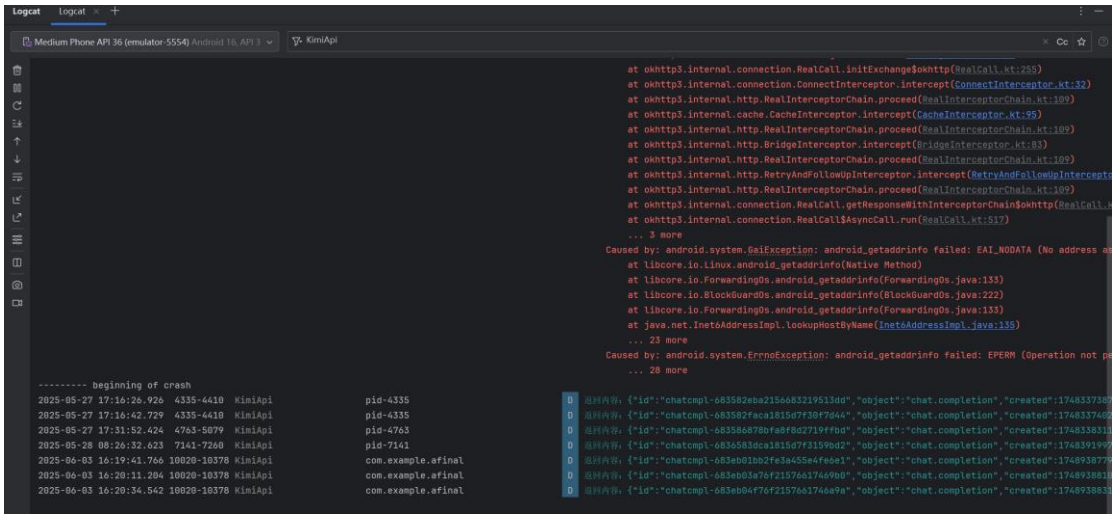
服务端返回结果包含 choices 数组，解析后提取 content 字段作为回复内容：

```
val choices = JSONObject(responseBody).getJSONArray("choices")
val reply =
choices.getJSONObject(0).getJSONObject("message").getString("content")
```

为提升调试体验，系统加入了详细的错误日志记录功能：

```
Log.e("KimiApi", "请求失败: ${e.message}", e)
Log.d("KimiApi", "返回内容: $responseBody")
```

调试过程如下图所示：



### 3.4.3.留言的本地读写与弹幕播放机制

为营造热烈活泼的校庆氛围，系统设计了一个留言墙功能。用户可以在此页面书写祝福语，提交后将以“弹幕”形式循环滚动播放。其背后核心逻辑包括本地存储读取机制与循环动画播放机制两个方面。

**1. 留言的本地存储与读取**

所有留言数据通过 BulletMessage 数据类进行统一表示：

```
data class BulletMessage(
    val content: String,
    val timestamp: Long
)
```

留言的持久化操作由 BulletStorage 工具类封装，通过 Context.filesDir 目录下的 bullet_messages.json 文件完成 JSON 读写。以下代码在 BulletScreen 启动时调用，自动加载已有留言：

```
LaunchedEffect(true) {
    messages = BulletStorage.loadMessages(context)
}
```

而每次用户点击“发送”后，系统会将新留言追加保存：

```
scope.launch(Dispatchers.IO) {
    BulletStorage.saveMessages(context, updatedList)
}
```

该机制实现了离线保存与自动恢复历史留言，保证用户体验的连贯性与数据完整性。

**2. 弹幕循环播放逻辑**

弹幕展示使用 Jetpack Compose 的动画系统完成。逻辑上，每 1.5 秒随机从留言列表中选取一条加入播放队列 activeBullets：

```
LaunchedEffect(Unit) {
    while (true) {
        if (messages.isNotEmpty()) {
            val nextMessage = messages.random()
            activeBullets.add(nextMessage.content)
            delay(1500L)
        } else {
            delay(3000L)
        }
```

```
    }
}
```

然后，activeBullets 中的每条留言都会触发一个 AnimatedBullet 动画组件，通过 Animatable 实现水平方向从右向左的滑动：

```
val offsetX = remember { Animatable(screenWidth.toFloat()) }
offsetX.animateTo(
    targetValue = -screenWidth.toFloat(),
    animationSpec = tween(durationMillis = duration, easing =
LinearEasing)
)
```

一旦动画播放完成，该留言将被从 activeBullets 中移除：

```
onAnimationEnd()
```

此机制保证了弹幕播放的动态性与持续性，即使留言数量较少也能反复轮播，避免“播放完就没了”的尴尬。

## 3.4.4.地图的缩放控制

为方便校庆期间参会师生快速查阅校园布局与活动场所，本系统提供了地图展示页面，内置校园高清地图，支持双指缩放查看细节。整个交互界面使用 Jetpack Compose 构建，简洁高效，交互自然。

**1. 地图组件加载**

地图资源通过 res/drawable/campus_map.jpg 引入，使用 Image 组件进行展示。通过 painterResource 加载静态资源，同时设置 ContentScale.Fit 以适应容器尺寸：

```
Image(
    painter = painterResource(id = R.drawable.campus_map),
    contentDescription = "校园地图",
    contentScale = ContentScale.Fit,
    modifier = Modifier
        .graphicsLayer(
            scaleX = scale,
            scaleY = scale
        )
)
```

地图初始缩放比例为 1.0，通过动态变更 graphicsLayer 的 scaleX 和

scaleY 属性控制整体缩放效果。

**2. 缩放手势的实现**

为了实现直观的双指缩放操作，Compose 提供了强大的手势检测系统。我们使用 pointerInput 与 detectTransformGestures 来捕获用户手势中的缩放变化：

```
.pointerInput(Unit) {
    detectTransformGestures { _, _, zoom, _ ->
        scale = (scale * zoom).coerceIn(0.5f, 4f)
    }
}
```

在这里：

- zoom 表示当前手势的缩放比例；
- scale 是一个响应式状态变量（remember { mutableStateOf(1f) }），控制最终缩放因子；
- 使用 coerceIn(0.5f, 4f) 限制缩放范围，避免地图过大或过小。

地图默认居中显示，并在用户双指缩放时，保持位置不变，仅调整比例，增强了地图的稳定性与可读性。

# 4. 系统可能的扩展

本系统目前已实现校庆活动展示、智能问答、留言墙及校园地图等核心功能，基本满足用户对校庆信息获取与互动的需求。

为了提升系统的可用性、扩展性与智能化水平，未来可在以下几个方面进行功能拓展：

## 4.1. 实时消息推送功能

通过接入 Firebase Cloud Messaging（FCM）或国内推送平台（如个推、信鸽），实现对用户的实时活动变更通知推送。例如：

- 活动开始前的提醒；
- 活动地点临时调整的通知；
- 发布最新校庆动态或校友留言精选。

## 4.2. 留言点赞与评论机制

在现有留言墙基础上，引入互动机制，支持用户对留言进行点赞、评论、转发等操作，增强留言内容的参与感与传播性。

## 4.3. 校园导航与定位功能

结合 GPS、WiFi 或蓝牙定位技术，配合电子地图展示用户当前位置与目标活动地点，实现：

- 校园 AR 导航；
- 快速查找最近的楼宇或展区；
- 多场馆之间的路径规划。

## 4.4. 更智能的问答助手

目前的智能助手依赖系统提示词与固定内容构建语义上下文。未来可通过：

- 接入校庆活动的后台数据库实现实时问答；
- 引入知识图谱技术扩展问答准确性；
- 支持语音输入、语音朗读功能，提升交互便捷性。

## 4.5. 用户身份系统与校友认证

引入登录系统，支持：

- 在校师生使用校园统一认证登录；
- 校友通过校友认证机制绑定身份；
- 提供个性化内容推送、活动报名、签到打卡等功能。

## 4.6. 多平台适配与数据同步

当前系统为 Android 客户端版本，未来可拓展为：

- Web 端：便于在大屏幕、投影仪上展示；
- iOS 端：覆盖更多用户；
- 后台管理平台：支持活动信息发布、留言审核、数据分析等。

未来这些扩展功能的实现，将使系统不仅仅是一个信息展示平台，更成为服务校庆、连接校友、增强归属感的重要桥梁。

# 5. 总结体会

通过本次“校园节日活动智能助手”系统的设计与开发，我不仅深入理解了移动互联网应用的开发流程，也在实践中提升了对 Android 平台与 Jetpack Compose 编程模式的掌握。整个项目从需求分析、功能设计、界面开发，到逻辑实现与本地数据存储，覆盖了完整的应用开发生命周期。

本项目让我熟悉了以下核心技术：

- **Jetpack Compose** 的声明式 UI 构建方式，使界面开发更加高效与模块化；
- **OkHttp 与 JSON 解析** 的结合，实现了对外部 AI 服务的异步调用；
- **数据本地存储与加载** 技术，提升了用户交互的持久性与稳定性；
- **Canvas 与动画机制** 的基础应用，为留言墙弹幕提供了生动的展示效果；
- **多线程协程（Kotlin Coroutine）** 的使用，提高了异步逻辑处理的能力与响应速度。

项目实施过程中，我体会到系统架构设计的重要性。合理的分层结构（界面层、逻辑层、数据层）能大幅提升代码的可维护性与拓展性。此外，在处理如“智能助手应答逻辑”时，如何设计提示词并结合真实活动数据，成为了系统“智能性”的关键。

通过不断测试、调试与优化，我也深刻理解了用户体验对移动互联网产品成功的决定作用。比如：留言的轮播间隔、弹幕密度控制、地图的缩放手势等细节，都直接影响使用舒适度。

总的来说，这次项目开发不仅加深了我对移动互联网相关技术的理解，也锻炼了我独立思考、解决问题与项目管理的能力。相信这将为我未来继续从事相关领域的学习与工作打下坚实基础。