MiniChat 聊天应用产品报告

1. 产品概述

1.1 产品概述

MiniChat是一款基于Android平台的即时通讯应用，使用Jetpack Compose开发用户界面，实现了基本的群聊功能。目标是提供轻量级、高性能的群聊功能。主要面向小型团队内部交流使用场景。

MiniChat使用使用本地数据库进行用户管理，具有基础的登录和注册功能，并实现了基于WebSocket的实时消息同步，也可以可自定义WebSocket服务器地址和端口，以及显示用户加入/退出的系统通知和消息发送时间。

1.2 产品特色

MiniChat作为轻量级群里应用，具有以下特点：

* 简洁的用户界面：使用MaterialDesign组件，实现了清晰的消息气泡布局；
* 可靠的通信机制：基于WebSocket实现双向通信，配备了自动重连机制处理连接中断；
* 完整的用户系统：使用Room数据库存储用户信息，支持登录、注册与输入验证功能；
* 灵活的配置选项：使用DataStore进行配置管理，使服务器连接参数可配置

1. 程序概要设计

2.1 整体架构

MiniChat采用MVVM(Model-View-ViewModel)架构模式，结合Jetpack组件和Kotlin协程实现响应式编程。整体架构分为以下几层：

1. UI层（View）

包含登录、注册、聊天、设置等界面，使用Jetpack Compose构建UI界面，通过ViewModel观察数据变化并更新UI；

1. 视图模型层（ViewModel）

负责处理UI相关的业务逻辑，管理界面状态和数据；

1. 数据层（Model）

Repository模式管理数据访问，Room数据库处理本地存储，WebSocket服务处理实时通信；

1. 依赖注入层

使用Hilt框架管理依赖，提供数据库、网络等组件实例。

分层架构图如图2.1所示。

图形用户界面, 网站

描述已自动生成

图1 MiniChatApp分层架构图

按照以上分层，项目代码可以分为data包(包括本地存储、网络通信和数据库实现代码)，di包(提供依赖注入Module)、domain.model包(定义核心数据类型，User和ChatMessage等实体类)、ui包(components可复用UI组件、screens和theme)。

项目代码按功能模块可以划分为用户登录与注册模块、WebSocket消息同步模块、系统配置模块和数据库模块，软件模块结构图如图2所示。

图示

描述已自动生成

图2 软件模块结构图

2.2 用户登录与注册模块

用户登录与注册模块采用MVVM架构模式实现，通过AuthViewModel连接视图层和数据层。在视图层，LoginScreen和RegisterScreen使用Jetpack Compose构建界面，实现了响应式的表单交互。登录界面包含用户名和密码输入框，同时提供了注册入口；注册界面则额外增加了密码确认输入框，并实现了基本的输入验证。

数据层由UserRepository负责统一管理用户认证相关的业务逻辑。UserRepository通过Room数据库实现本地用户信息的存储，其中User实体类包含用户名、密码和创建时间等基本信息。用户注册时，UserRepository会先检查用户名是否已存在，若不存在则创建新用户记录；登录时则验证用户名和密码的匹配情况。同时密码输入框实现了明文/密文切换功能，增强了用户体验。注册时的输入验证确保用户名长度不少于3个字符，密码长度不少于6个字符，并要求两次输入的密码保持一致。

认证结果通过Kotlin Flow机制传递给ViewModel，再由视图层观察状态变化并作出相应的界面更新。登录成功后，用户会被导航至聊天主界面；注册成功后则返回登录界面。整个过程采用协程处理异步操作，确保界面响应流畅。

屏幕的手机截图

描述已自动生成屏幕上有个手机

描述已自动生成

图3 登录与注册界面示意

2.3 WebSocket消息同步模块

WebSocket消息同步模块是MiniChat应用的核心通信组件，采用客户端-服务器架构实现实时消息传输。客户端采用ChatService类管理WebSocket连接，使用OkHttp实现WebSocket客户端功能。ChatService通过协程和Flow机制实现消息的异步处理和状态管理，确保消息收发的可靠性。

ChatViewModel作为消息管理的核心组件，负责处理消息的发送和接收逻辑。当用户发送消息时，ChatViewModel将消息通过ChatService发送至服务器；接收到新消息时，则通过状态流通知UI层更新显示。消息列表采用协程的StateFlow机制管理，确保UI层能实时反映最新的消息状态。为了提高连接可靠性，ChatService实现了自动重连机制。当检测到连接断开时，系统会根据配置的重连间隔自动尝试重新建立连接。连接状态的变化会通过系统消息通知用户，使用户能够及时了解当前的连接情况。消息展示界面通过不同的消息气泡样式，清晰区分系统消息、自己发送的消息和他人发送的消息。

图形用户界面, 文本, 应用程序

描述已自动生成

图4 聊天室示意图

服务器端基于Spring Boot框架开发，使用Spring WebSocket实现消息处理。服务器维护一个ConcurrentHashMap来管理所有活动的WebSocket会话，支持多客户端并发连接。当用户通过URL参数提供用户名并成功建立WebSocket连接后，服务器会向所有在线用户广播该用户加入的系统消息。

在消息传输层面，客户端和服务器之间采用JSON格式进行数据交换，使用ChatMessage数据类封装消息内容。每条消息包含发送者ID、消息内容、时间戳等属性。系统消息使用特定的发送者ID"System"进行标识，用于显示用户加入、退出等状态变更通知。

2.4 系统配置模块

系统配置模块基于Android的DataStore机制实现配置信息的持久化存储。AppSettings类作为配置管理的核心组件，使用Kotlin的Preference DataStore存储服务器地址、端口号和重连间隔等关键参数。相比传统的SharedPreferences，DataStore提供了基于Flow的类型安全的数据访问接口，并支持异步操作。配置界面通过SettingsScreen和SettingsViewModel实现。用户可以在设置界面修改WebSocket服务器的连接参数，包括服务器地址和端口号，以及自动重连的时间间隔。SettingsViewModel通过StateFlow机制管理配置状态，当配置发生变更时，通过AppSettings将新的配置值异步写入DataStore。

为了确保配置参数的有效性，系统在保存配置时进行基本的参数验证。端口号必须在1到65535的有效范围内，重连间隔不得小于1秒。系统默认配置了适用于Android模拟器测试环境的参数，服务器地址默认为"10.0.2.2"，端口号默认为8080，重连间隔默认为5秒。

屏幕上有个手机

描述已自动生成

图5 系统配置界面示意图

2.5 数据库设计

数据库模块基于Room持久化库实现，为应用提供稳定的本地数据存储能力。AppDatabase作为数据库的核心类，继承自RoomDatabase数据实体采用User类设计，使用@Entity注解标记数据表结构。用户表以username作为主键，同时包含password和createdAt字段，分别用于存储用户密码和账号创建时间。数据访问接口通过UserDao定义，提供了insertUser、getUserByUsername和getUserByCredentials等基本的数据操作方法。为防止用户名重复，insertUser方法使用ABORT冲突策略，在发生冲突时抛出异常。

数据库操作全部采用挂起函数实现，配合Kotlin协程使用，避免阻塞主线程。用户注册时，系统首先通过getUserByUsername检查用户名是否已存在，若不存在则调用insertUser写入新用户记录。登录验证则使用getUserByCredentials方法，同时匹配用户名和密码字段。DatabaseModule在数据库首次创建时通过回调机制初始化默认用户数据，便于开发测试。数据库的创建和访问通过Room的Builder模式配置，确保数据库操作的安全性和可靠性。整个数据库模块通过Repository模式与上层业务逻辑解耦，提供清晰的数据访问接口。

1. 项目成果与优化方向

3.1 当前实现的功能

MiniChat目前实现了即时通讯应用的基础功能框架。在用户认证方面，应用支持本地账号的注册和登录，包含了基本的输入验证机制。登录界面预设了测试账号，方便开发调试。针对用户体验，密码输入框实现了明文切换功能，注册界面会对用户输入进行实时检查。

即时通讯功能基于WebSocket实现了可靠的消息同步机制。系统能够显示在线用户的加入和退出状态，消息展示界面采用不同样式区分系统消息和用户消息。消息气泡中包含发送者信息和发送时间，为用户提供清晰的对话上下文。应用实现了自动重连功能，在网络波动时自动恢复连接。

系统设置方面，应用支持服务器连接参数的自定义配置。用户可以根据实际部署环境修改服务器地址和端口，并可调整自动重连的时间间隔。这些配置通过DataStore持久化存储，确保应用重启后配置仍然有效。

3.2 后续优化方向

1. 增加图片库以及图片发送功能；
2. 增加联系人功能；
3. 将用户信息转移到服务器存储；