

自然写互动课堂智慧黑板端应用软件 V1.0

鉴别材料

软件名称：自然写互动课堂智慧黑板端应用软件

版本号：V1.0

著作权人：深圳自然写科技有限公司

开发完成日期：2024年6月

文档类型：设计说明书 + 用户操作手册

目录

- 第一章 软件整体概述
 - 1.1 软件简介与功能综述
 - 1.2 软件用途与适用场景
 - 1.3 运行环境与系统要求
 - 1.4 开发语言与技术规范
 - 1.5 版本说明
- 第二章 系统架构与设计思路
 - 2.1 总体架构设计
 - 2.2 各层次详细说明
 - 2.3 核心模块架构图
 - 2.4 数据设计
 - 2.5 接口设计
 - 2.6 安全设计
 - 2.7 部署架构
- 第三章 核心模块功能详细说明
 - 3.1 全班笔迹实时接收与大屏展示模块
 - 3.2 触控白板书写模块
 - 3.3 学生作品展示墙模块
 - 3.4 课堂互动答题系统模块
 - 3.5 随机抽取与分组展示模块
 - 3.6 课堂录制与回放模块
 - 3.7 课件加载与解析模块
 - 3.8 设备联动与网关发现模块

- 第四章 操作流程与使用步骤
 - 4.1 设备安装与初始化配置
 - 4.2 应用启动与教师登录
 - 4.3 课堂主要操作流程
 - 4.4 白板操作与触控书写
 - 4.5 互动答题操作流程
 - 4.6 录制与回放操作
 - 4.7 异常处理与故障排除
- 第五章 与源代码的对应关系
 - 5.1 模块名称与源代码文件对应表
 - 5.2 核心功能类与方法说明
 - 5.3 主要类命名规范
- 附录

第一章 软件整体概述

1.1 软件简介与功能综述

自然写互动课堂智慧黑板端应用软件（以下简称"黑板端应用"）是自然写互动课堂教学系统的核心显示与交互终端软件。该软件运行于教室内配置的智慧黑板（交互式一体机）设备上，承担课堂教学过程中的内容显示、多学生笔迹实时展示、教师触控书写、互动答题组织及课堂录制等核心职能。

本软件基于 Android 全屏交互式应用架构开发，采用 Java/Kotlin 编写业务逻辑，C++ 通过 JNI 接口加速笔迹渲染核心算法。软件面向教师和课堂管理需求，提供如下八大功能：

功能综述一览：

序号	功能名称	功能描述
1	全班笔迹实时接收与大屏展示	通过 WebSocket 实时接收网关/算力盒推送的全班学生笔迹，并在大屏幕上并发展示
2	触控白板书写	教师可直接在黑板触控屏上手写板书，笔迹流畅精准
3	学生作品展示墙	选取特定学生的书写作品进行大屏对比展示，便于讲评
4	课堂互动答题系统	发布题目、收集作答、统计分析、展示结果的完整闭环
5	随机抽取与分组展示	随机选取学生展示、小组分组竞赛
6	课堂录制与回放	使用 MediaCodec H.264 编码录制课堂全程，支持回放

序号	功能名称	功能描述
7	第三方课件兼容	支持 PPT/PDF/图片等主流格式课件的加载与翻页
8	与教室网关联动	通过 mDNS 自动发现并绑定教室网关，与点阵笔系统无缝对接

黑板端应用在整个课堂教学系统中处于核心枢纽地位：一方面接收来自网关/算力盒汇聚的全班学生点阵笔数据；另一方面向网关发出课堂控制指令（发题、收卷、暂停、分组等）；同时通过云平台 API 完成课件下载、录像上传和数据同步。

1.2 软件用途与适用场景

主要用途：

黑板端应用专为 K12 教育场景设计，适用于配备智慧黑板（交互式一体机）和自然写点阵笔书写系统的现代化教室。教师在日常语文、数学、英语、书法等学科课堂中使用该软件，实现：

- 无纸化书写采集与实时大屏展示（学生用点阵笔在点阵纸上书写，内容实时出现在黑板屏幕）
- 全班书写进度一目了然（所有学生笔迹同时展示，教师可即时发现学习薄弱点）
- 互动课堂组织（发布抢答、选择题、大字展示等多种互动形式）
- 课堂内容留存（录制全程用于课后复习和教学研究）

适用场景：

场景	描述
语文书法课	展示全班学生毛笔字/硬笔书写，点评对比
数学计算课	实时展示学生解题过程，讲解典型解法
英语写作课	收集学生手写单词/句子并投屏批改
互动答题课	发布判断题/选择题，收集答案并统计分析
书法专项课	字帖展示 + 学生练习实时对照
期末考前复习	随机抽取学生展示，查漏补缺

1.3 运行环境与系统要求

硬件要求：

项目	最低要求	推荐配置
设备类型	智慧黑板/交互式一体机	65寸及以上触控一体机

项目	最低要求	推荐配置
处理器	8核 ARM Cortex-A55 @ 1.6GHz	8核 ARM Cortex-A76 @ 2.0GHz+
内存	4GB RAM	8GB RAM
存储	32GB eMMC	64GB eMMC
显示分辨率	1920×1080 FHD	3840×2160 4K UHD
触控技术	20点红外触控	40点红外触控
网络接口	千兆以太网 + 802.11ac Wi-Fi	千兆以太网 + Wi-Fi 6
蓝牙	BLE 5.0（可选）	BLE 5.0

软件要求：

项目	要求
操作系统	Android 9.0+（智慧黑板定制Android系统）
安卓SDK	compileSdkVersion 34, minSdkVersion 28
Java运行时	OpenJDK 11（系统内置）
NDK版本	Android NDK r25c（C++17标准库）
存储权限	READ/WRITE_EXTERNAL_STORAGE（课件缓存、录像保存）
摄像头权限	可选，部分互动功能使用
麦克风权限	RECORD_AUDIO（课堂录制音频轨道）

网络要求：

场景	要求
教室局域网	建议 100Mbps 以上（支持全班30人同时笔迹传输）
云平台访问	20Mbps+ 带宽（课件下载、录像上传）
延迟要求	局域网笔迹延迟 ≤ 50ms，端到端显示延迟 ≤ 200ms

1.4 开发语言与技术规范

语言/框架	版本	用途
Java	JDK 11	基础组件、兼容性代码
Kotlin	1.9.x	主要业务逻辑、UI控制器

语言/框架	版本	用途
C++	C++17 (NDK r25c)	白板渲染引擎 JNI 加速
Android SDK	API Level 34	系统API调用
Jetpack ViewModel	2.6.x	MVVM 状态管理
Jetpack LiveData	2.6.x	数据响应式绑定
Room	2.6.x	本地数据库 (SQLite封装)
OkHttp	4.12.x	HTTP 网络请求
OkHttp WebSocket	4.12.x	WebSocket 笔迹实时流
Apache POI	5.2.x	PPT 课件解析
PdfRenderer	Android内置	PDF 渲染
Glide	4.16.x	图片课件加载
MediaCodec	Android内置	课堂录制 H.264 编码
MediaMuxer	Android内置	音视频合流 MP4

编码规范：遵循 Google Android Kotlin Style Guide，采用 MVVM 架构模式，单向数据流（View → ViewModel → Repository → DataSource）。

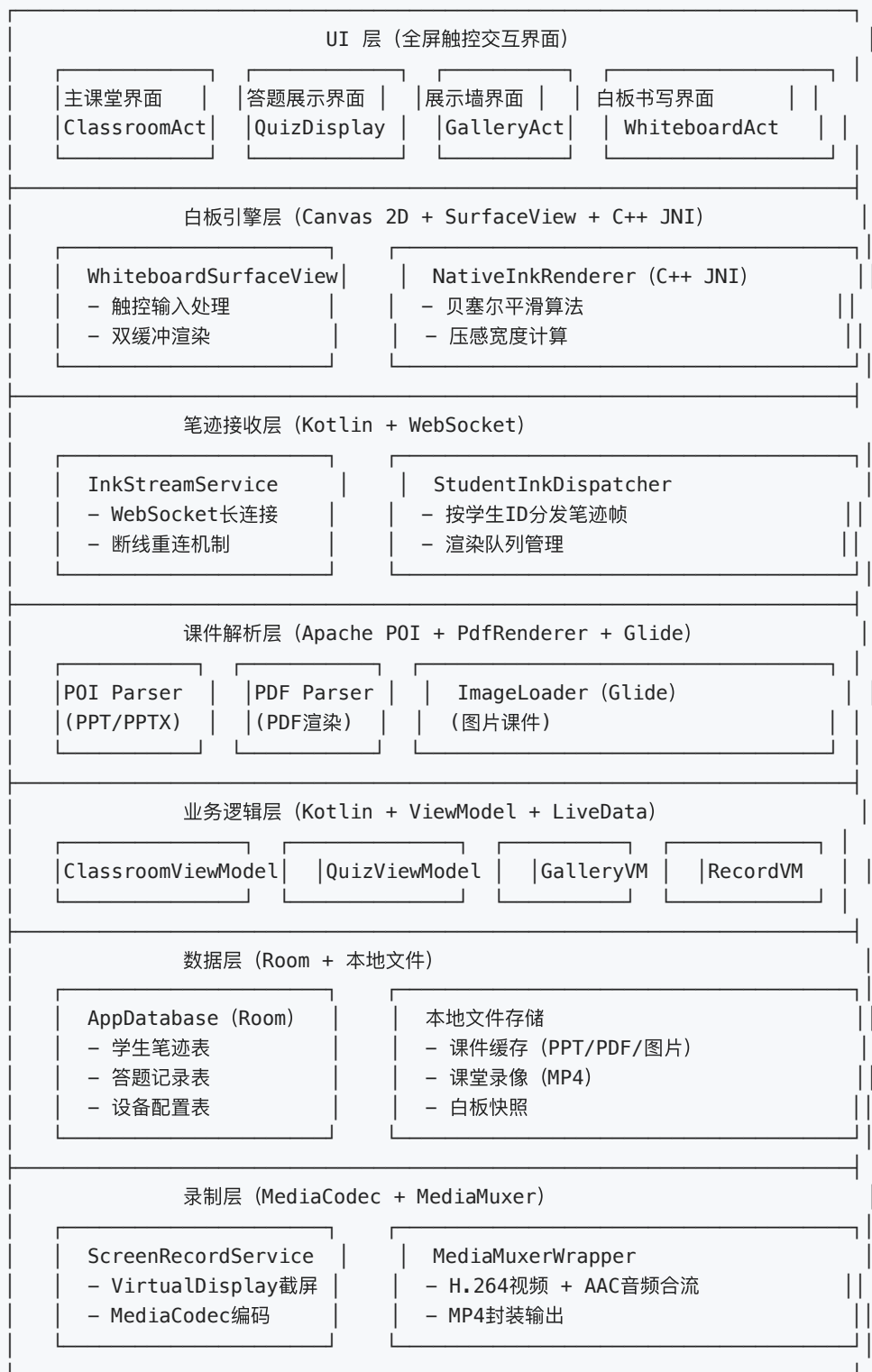
1.5 版本说明

版本	日期	说明
V1.0.0	2024-06	正式发布版本，包含全班笔迹展示、触控白板、互动答题、课堂录制核心功能
V0.9.0	2024-04	Beta版本，完成教室局域网笔迹传输验证
V0.5.0	2024-01	Alpha版本，完成白板引擎原型验证

第二章 系统架构与设计思路

2.1 总体架构设计

黑板端应用采用 Android 全屏交互式应用架构，整体分为七个层次：UI层、白板引擎层、笔迹接收层、课件解析层、业务逻辑层、数据层和录制层。各层职责清晰，通过 ViewModel + LiveData 响应式机制驱动数据流动。



2.2 各层次详细说明

2.2.1 UI层

UI层负责全屏交互界面的展示与用户操作响应。采用 Android 原生 View 体系，针对大尺寸触控屏幕（65寸、75寸、86寸等）做了专项优化：

- **ClassroomActivity**: 课堂主界面，包含全班笔迹展示区、工具栏（发题/白板/抽人/录制）、学生列表侧边栏
- **WhiteboardActivity**: 全屏白板书写界面，教师触控书写，支持多种笔色、笔粗和橡皮擦
- **QuizDisplayActivity**: 互动答题展示界面，包含题目区域、实时收卷进度、答案统计图表
- **GalleryActivity**: 学生作品展示墙，宫格布局展示多名学生书写作品

所有 Activity 均以全屏模式（SYSTEM_UI_FLAG_FULLSCREEN）运行，并配置 Kiosk 模式锁定，防止学生退出应用访问系统设置。

2.2.2 白板引擎层

白板引擎层是黑板端应用的核心渲染模块，提供教师触控书写的流畅体验：

- **WhiteboardSurfaceView**: 继承自 SurfaceView，在独立渲染线程执行 Canvas 2D 绘制，避免阻塞主线程
- **NativeInkRenderer**: C++ JNI 加速的笔迹渲染核心，实现贝塞尔曲线平滑和压感宽度模拟
- **双缓冲策略**: 前台 Canvas 显示当前帧，后台 Bitmap 保存历史笔迹，避免闪烁

触控采样率处理： – 采集 Android TouchEvent (ACTION_DOWN / ACTION_MOVE / ACTION_UP) – 历史轨迹点（`event.getHistoricalX/Y`）全量采样，确保高速书写不丢点 – 笔迹压感通过触控面积（`event.getTouchMajor()`）模拟

2.2.3 笔迹接收层

笔迹接收层负责从网关/算力盒接收全班学生实时笔迹数据流：

- **InkStreamService**: Android Service（前台服务），维护与教室网关的 WebSocket 长连接
- **协议解析**: 接收二进制笔迹帧，解析为 `StudentStrokeFrame`（学生ID + 笔迹点数组）
- **StudentInkDispatcher**: 按学生ID将笔迹帧分发到对应的渲染区域
- **断线重连**: 指数退避重连策略（初始1秒，最大30秒），网络波动时自动恢复

并发处理模型： – 最多支持 60 名学生同时书写（60路并发笔迹流） – 使用 `ConcurrentHashMap<String, StudentInkBuffer>` 按学生ID隔离笔迹缓冲区 – 渲染调度采用统一的 60fps绘制周期（`Choreographer.FrameCallback`），批量消费所有学生笔迹缓冲

2.2.4 课件解析层

课件解析层支持教室常用课件格式的加载与渲染：

- **POI Parser**: Apache POI 5.x 解析 PPT/PPTX 文件，将每页转换为 Bitmap 缓存
- **PdfRenderer**: Android 内置 PdfRenderer API 渲染 PDF 页面为高清 Bitmap
- **ImageLoader**: Glide 4.x 加载图片课件（JPG/PNG/GIF），自动内存/磁盘双级缓存
- **课件预加载**: 进入课堂前预下载并解析课件，确保翻页流畅（目标 < 200ms/页）

2.2.5 业务逻辑层

业务逻辑层采用 Jetpack ViewModel + LiveData MVVM 模式，负责协调各功能模块：

- **ClassroomViewModel**：课堂核心状态管理（课堂状态机、学生名单、笔迹接收控制）
- **QuizViewModel**：互动答题业务（发题控制、收卷统计、结果分析）
- **GalleryViewModel**：作品展示墙逻辑（选人、布局计算、对比展示）
- **RecordViewModel**：录制控制（开始/停止/时长计时、文件管理）

2.2.6 数据层

数据层使用 Room 数据库封装本地 SQLite，并结合文件系统管理大文件：

- **AppDatabase**：Room Database，包含学生笔迹表、答题记录表、设备配置表
- **本地文件存储**：课件缓存目录、课堂录像目录、白板快照目录（均位于应用私有存储区）
- **SharedPreferences**：网关绑定信息、显示分辨率设置、触控校准参数

2.2.7 录制层

录制层通过 Android MediaProjection API 实现课堂全程录制：

- **ScreenRecordService**：前台 Service，使用 VirtualDisplay 截取屏幕内容
- **MediaCodec**：H.264 硬件编码，配置 1080p @ 30fps，码率 4Mbps
- **MediaMuxer**：将 H.264 视频流与 AAC 音频流合并为 MP4 文件输出

2.3 核心模块架构图

数据流向图：





互动答题数据流：



2.4 数据设计

2.4.1 数据库表结构

student_ink 表（学生笔迹）

字段名	数据类型	说明
id	INTEGER PRIMARY KEY	自增主键
session_id	TEXT NOT NULL	课堂会话ID
student_id	TEXT NOT NULL	学生ID
student_name	TEXT	学生姓名
stroke_data	BLOB	笔迹数据序列化（压缩后二进制）
created_at	INTEGER	时间戳（毫秒）
page_index	INTEGER	对应课件页码

quiz_record 表（互动答题记录）

字段名	数据类型	说明
id	INTEGER PRIMARY KEY	自增主键
session_id	TEXT NOT NULL	课堂会话ID
quiz_id	TEXT NOT NULL	题目ID
quiz_type	INTEGER	题目类型 (1=选择 2=判断 3=书写)
quiz_content	TEXT	题目内容 (JSON)
student_id	TEXT	作答学生ID
answer_data	TEXT	学生答案 (JSON)
is_correct	INTEGER	是否正确 (0=错 1=对 -1=未判)
answered_at	INTEGER	作答时间戳

device_config 表 (设备配置)

字段名	数据类型	说明
key	TEXT PRIMARY KEY	配置键名
value	TEXT	配置值
updated_at	INTEGER	更新时间戳

2.4.2 本地文件存储结构

```
/data/data/com.writech.board/
├─ databases/
│   └─ writech_board.db      (Room 数据库文件)
├─ shared_prefs/
│   └─ board_config.xml      (SharedPreferences 配置)
└─ files/
    ├─ courses/              (课件缓存目录)
    │   ├─ {course_id}.pptx
    │   └─ {course_id}/
    │       ├─ slide_001.png  (预渲染页面缓存)
    │       └─ slide_002.png
    ├─ records/              (课堂录像目录)
    │   └─ record_{timestamp}.mp4
    └─ snapshots/            (白板快照目录)
        └─ snapshot_{timestamp}.png
```

2.4.3 核心数据结构定义

```
// 学生笔迹帧（来自网关）
data class StudentStrokeFrame(
    val studentId: String,           // 学生设备ID
    val studentName: String,         // 学生姓名
    val penId: String,               // 点阵笔序列号
    val points: List<InkPoint>,      // 笔迹点列表
    val timestamp: Long,             // 帧时间戳（ms）
    val isStrokeEnd: Boolean         // 是否笔画结束（抬笔）
)

// 单个笔迹点
data class InkPoint(
    val x: Float,                   // 归一化坐标 [0.0, 1.0]
    val y: Float,                   // 归一化坐标 [0.0, 1.0]
    val pressure: Float,            // 压感值 [0.0, 1.0]（0表示无压感）
    val timestamp: Long             // 点时间戳（us）
)

// 互动题目
data class QuizQuestion(
    val quizId: String,
    val type: QuizType,             // CHOICE / JUDGE / WRITE
    val content: String,            // 题目文本
    val options: List<String>,      // 选项列表（选择题）
    val correctAnswer: String,      // 正确答案
    val duration: Int               // 作答时限（秒，0=不限时）
)
```

2.5 接口设计

2.5.1 外部接口

与网关/算力盒（WebSocket 笔迹数据流）：

- 连接地址：ws://{gateway_ip}:8080/board/ink-stream
- 认证方式：连接时携带 Authorization: Bearer {token} 请求头
- 消息格式：二进制帧，自定义协议

笔迹帧格式（Binary）：

魔数 (2B)	版本 (1B)	类型 (1B)	载荷 (变长)
0xAB 0xCD	0x01	0x01	{studentId, points[]}

与云平台（HTTPS RESTful API）：

接口	方法	路径	说明
设备激活	POST	/api/v1/board/activate	黑板设备注册激活

接口	方法	路径	说明
获取课堂列表	GET	/api/v1/classroom/list	获取今日课表
下载课件	GET	/api/v1/course/{id}/download	下载课件文件
同步课堂数据	POST	/api/v1/classroom/{id}/sync	上传课堂笔迹/答题数据
上传录像	PUT	/api/v1/record/upload	分片上传课堂录像
设备心跳	POST	/api/v1/board/heartbeat	设备在线状态上报

向网关发送课堂控制指令 (WebSocket):

```
// 指令格式 (JSON)
{
  "cmd": "ISSUE_QUIZ",    // 指令类型: ISSUE_QUIZ/COLLECT/PAUSE/RESUME/RANDOM_PICK/GROUP
  "sessionId": "...",    // 课堂会话ID
  "payload": {...}       // 指令参数 (按类型不同)
}
```

2.5.2 内部模块接口

WhiteboardSurfaceView 对外接口:

```
interface WhiteboardController {
  fun setTool(tool: DrawingTool)           // 设置工具 (画笔/橡皮/选择)
  fun setPenColor(color: Int)              // 设置笔色
  fun setPenWidth(widthDp: Float)          // 设置笔粗 (dp)
  fun undo()                               // 撤销
  fun redo()                               // 重做
  fun clear()                              // 清除画布
  fun saveSnapshot(): Bitmap               // 保存快照
  fun loadBackground(bitmap: Bitmap)       // 加载课件页面为背景
  fun overlayStudentInk(frame: StudentStrokeFrame) // 叠加学生笔迹
}
```

ScreenRecordService 对外接口:

```
// 通过 Bound Service 方式调用
interface IRecordService {
  fun startRecord(outputPath: String, config: RecordConfig): Boolean
  fun stopRecord(): String           // 返回录像文件路径
  fun pauseRecord()
  fun resumeRecord()
  fun getRecordDuration(): Long // 当前录制时长 (ms)
  fun isRecording(): Boolean
}
```

2.6 安全设计

Kiosk 模式锁定：

```
// 设备所有者模式，防止学生退出应用
class KioskModeManager(private val context: Context) {
    private val dpm = context.getSystemService(DevicePolicyManager::class.java)
    private val adminComponent = ComponentName(context,
BoardDeviceAdminReceiver::class.java)

    fun enableKioskMode() {
        // 锁定任务模式
        (context as Activity).startLockTask()
        // 禁用状态栏
        dpm.setStatusBarDisabled(adminComponent, true)
        // 设置允许的包（只允许黑板应用）
        dpm.setLockTaskPackages(adminComponent, arrayOf(context.packageName))
    }
}
```

内容安全策略： – 课堂录像本地 AES-256 加密存储，上传云端成功后自动清理本地文件 – 防截屏：设置 `WindowManager.LayoutParams.FLAG_SECURE`，防止学生通过辅助功能截取题目答案 – 应用日志：生产版本禁用 logcat 详细日志，防止敏感信息泄露

自动登录与设备证书： – 黑板设备使用设备证书（X.509 客户端证书）与云平台进行双向 TLS 认证 – 无需教师手动输入账号密码，设备证书由学校管理员统一签发和部署 – 证书存储于 Android Keystore 系统密钥库，防止提取

网络隔离： – 通过 Android NetworkSecurityConfig 限制明文 HTTP 通信 – 仅配置信任学校专属 CA 证书，防止中间人攻击

2.7 部署架构

教室局域网内部：

智慧黑板（黑板端应用）
‣ WebSocket（笔迹流 + 控制指令）
教室网关/算力盒
‣ BLE / Wi-Fi
全班学生点阵笔（30~60支）

(所有设备通过教室交换机/路由器互联，VLAN 隔离保障稳定性)

‣ HTTPS + WebSocket (WAN)

自然写云平台（公有云部署）
– 课件存储与下载
– 课堂数据同步
– 录像上传与归档

第三章 核心模块功能详细说明

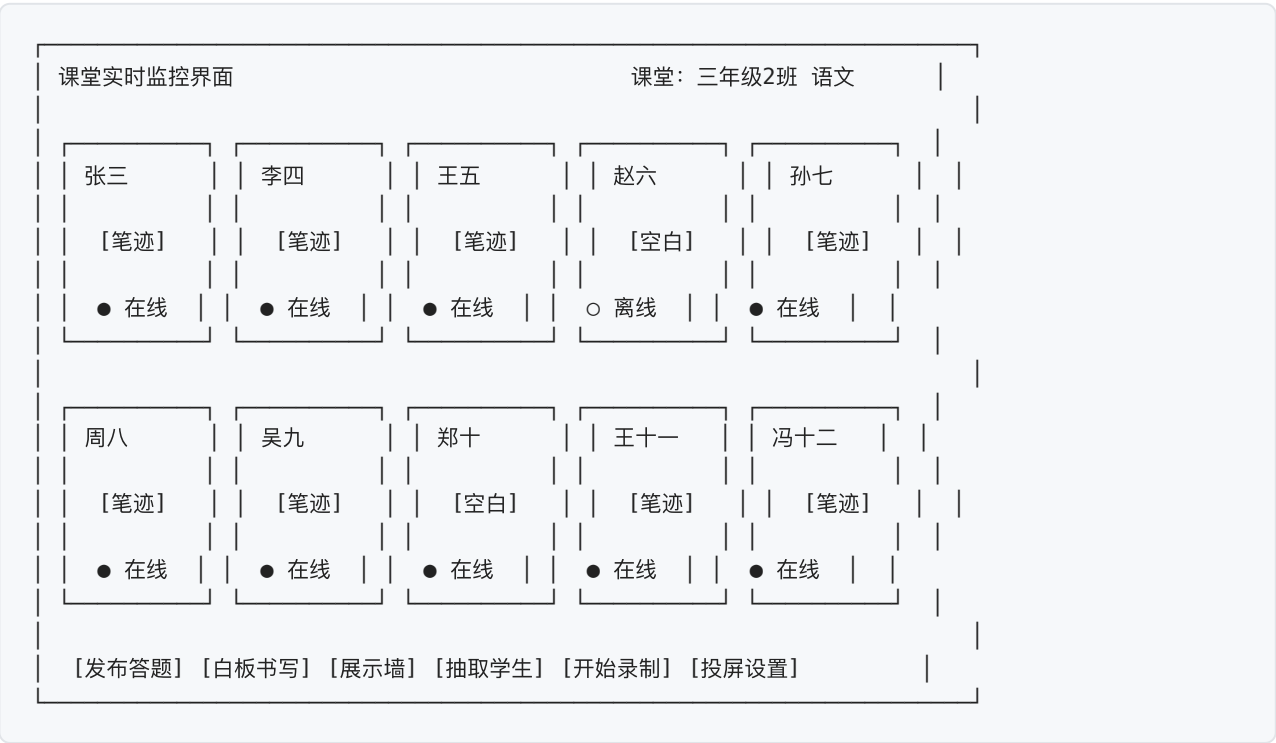
3.1 全班笔迹实时接收与大屏展示模块

3.1.1 模块功能描述

全班笔迹实时接收与大屏展示模块是黑板端应用的核心功能，负责接收全班所有学生通过点阵笔书写的实时笔迹数据，并在大屏幕上以可视化方式并发展示。

教师可在大屏幕上直观看到全班书写状态：已书写（笔迹内容）、未书写（空白）、书写进度等，帮助教师快速掌握学情。

3.1.2 界面布局



界面元素说明：

- 宫格展示区：每格显示一名学生的实时笔迹内容，宫格数量自适应学生人数（5列×N行）
- 学生在线状态指示：绿色圆点（在线）/ 灰色圆点（离线/未连笔）
- 底部工具栏：课堂主要操作功能快捷按钮
- 右上角：课堂信息（班级、科目、当前时间）

3.1.3 处理流程



```

    | 帧解析
    ▼
步骤3: 帧解析为 StudentStrokeFrame
    InkFrameParser.parse(bytes) → StudentStrokeFrame
    | studentId 分发
    ▼
步骤4: 分发到对应学生缓冲区
    StudentInkDispatcher.dispatch(frame)
    studentInkBuffers[frame.studentId].offer(frame)
    | Choreographer.FrameCallback (60fps)
    ▼
步骤5: 统一渲染 (每帧)
    for each studentId in activeStudents:
        frames = studentInkBuffers[studentId].drainAll()
        studentViewMap[studentId].drawFrames(frames)
        |
    ▼
步骤6: 大屏幕显示更新完成

```

3.1.4 关键算法：笔迹平滑渲染

黑板端应用通过 C++ JNI 加速实现高性能笔迹平滑渲染：

```

// native/ink_renderer.cpp
// 三次贝塞尔曲线平滑笔迹路径
void NativeInkRenderer::renderStroke(
    JNIEnv* env, jlong canvas_ptr,
    jfloatArray points_x, jfloatArray points_y,
    jfloatArray pressures, jint count) {

    float* px = env->GetFloatArrayElements(points_x, nullptr);
    float* py = env->GetFloatArrayElements(points_y, nullptr);
    float* pr = env->GetFloatArrayElements(pressures, nullptr);

    SkPath path;
    path.moveTo(px[0], py[0]);

    for (int i = 1; i < count - 1; i++) {
        // 中点平滑：以相邻两点的中点作为贝塞尔控制点终止点
        float midX = (px[i] + px[i+1]) * 0.5f;
        float midY = (py[i] + py[i+1]) * 0.5f;
        path.quadTo(px[i], py[i], midX, midY);

        // 根据压感动态调整线宽
        float width = BASE_WIDTH * (0.5f + 0.5f * pr[i]);
        paint_.setStrokeWidth(width);
    }

    // 最后一段直接连接到终点
    if (count > 1) {
        path.lineTo(px[count-1], py[count-1]);
    }

    SkCanvas* skCanvas = reinterpret_cast<SkCanvas*>(canvas_ptr);
    skCanvas->drawPath(path, paint_);
}

```

```
env->ReleaseFloatArrayElements(points_x, px, JNI_ABORT);
env->ReleaseFloatArrayElements(points_y, py, JNI_ABORT);
env->ReleaseFloatArrayElements(pressures, pr, JNI_ABORT);
}
```

3.1.5 性能指标

指标	目标值	实测值
最大并发学生数	60	60
笔迹帧率	60fps	≥ 58fps
端到端延迟（笔→屏）	≤ 200ms	约 80~150ms
内存占用（60学生）	≤ 1.5GB	约 1.2GB
CPU 占用率	≤ 60%	约 40~55%

3.2 触控白板书写模块

3.2.1 模块功能描述

触控白板书写模块为教师提供在智慧黑板大屏幕上直接触控手写的功能，支持多种笔色、笔粗、橡皮擦和选择移动操作，支持撤销/重做历史记录。

教师可以： – 在课件页面叠加手写批注（叠加模式） – 切换到纯白板模式进行板书 – 将学生笔迹叠加在白板上进行讲评 – 保存白板快照（截图）

3.2.2 界面布局



3.2.3 SurfaceView 渲染实现

```
class WhiteboardSurfaceView @JvmOverloads constructor(
    context: Context,
    attrs: AttributeSet? = null
) : SurfaceView(context, attrs), SurfaceHolder.Callback {

    private val renderThread: HandlerThread = HandlerThread("WhiteboardRender")
    private lateinit var renderHandler: Handler
    private var historyBitmap: Bitmap? = null // 历史笔迹缓存 (背景 Bitmap)
    private val currentPath = Path() // 当前正在书写的笔画路径
    private val paint = Paint(Paint.ANTI_ALIAS_FLAG).apply {
        style = Paint.Style.STROKE
        strokeJoin = Paint.Join.ROUND
        strokeCap = Paint.Cap.ROUND
        strokeWidth = 8f
        color = Color.BLACK
    }

    // 触控事件处理
    override fun onTouchEvent(event: MotionEvent): Boolean {
        val x = event.x
        val y = event.y

        when (event.action) {
            MotionEvent.ACTION_DOWN -> {
                currentPath.reset()
                currentPath.moveTo(x, y)
                lastX = x; lastY = y
            }
            MotionEvent.ACTION_MOVE -> {
                // 使用历史轨迹点，避免高速书写丢点
                val historySize = event.historySize
                for (i in 0 until historySize) {
                    val hx = event.getHistoricalX(i)
                    val hy = event.getHistoricalY(i)
                    // 贝塞尔控制点平滑
                    val midX = (lastX + hx) / 2f
                    val midY = (lastY + hy) / 2f
                    currentPath.quadTo(lastX, lastY, midX, midY)
                    lastX = hx; lastY = hy
                }
                currentPath.quadTo(lastX, lastY, (lastX + x) / 2f, (lastY + y) / 2f)
                lastX = x; lastY = y
                invalidateRender()
            }
            MotionEvent.ACTION_UP -> {
                // 笔画结束，合并到历史 Bitmap
                mergePathToHistory()
                undoStack.push(currentPath.copy())
                currentPath.reset()
            }
        }
    }
}
```

```

        return true
    }

    // 渲染到 Surface
    private fun renderFrame() {
        val canvas = holder.lockCanvas() ?: return
        try {
            canvas.drawColor(Color.WHITE)
            // 绘制历史笔迹 Bitmap (包含课件背景)
            historyBitmap?.let { canvas.drawBitmap(it, 0f, 0f, null) }
            // 绘制当前笔画路径
            canvas.drawPath(currentPath, paint)
        } finally {
            holder.unlockCanvasAndPost(canvas)
        }
    }
}

```

3.2.4 撤销/重做机制

```

class WhiteboardUndoManager {
    private val undoStack = ArrayDeque<WhiteboardSnapshot>(50) // 最多50步撤销
    private val redoStack = ArrayDeque<WhiteboardSnapshot>(50)

    fun pushState(bitmap: Bitmap) {
        if (undoStack.size >= MAX_UNDO_STEPS) {
            undoStack.removeFirst() // 超出限制, 丢弃最旧的状态
        }
        undoStack.addLast(WhiteboardSnapshot(bitmap.copy(Bitmap.Config.ARGB_8888, false)))
        redoStack.clear() // 新操作后清空重做栈
    }

    fun undo(): Bitmap? {
        if (undoStack.isEmpty()) return null
        val state = undoStack.removeLast()
        redoStack.addLast(state)
        return undoStack.lastOrNull()?.bitmap
    }

    fun redo(): Bitmap? {
        if (redoStack.isEmpty()) return null
        val state = redoStack.removeLast()
        undoStack.addLast(state)
        return state.bitmap
    }
}

```

3.3 学生作品展示墙模块

3.3.1 模块功能描述

学生作品展示墙模块允许教师从全班学生中选取若干学生的书写作品（1~9个），在大屏幕上以宫格对比布局展示，便于教师点评和学生互相学习。

功能特点： – 支持最多9个学生作品同屏展示（3×3宫格） – 支持单个作品放大聚焦（点击放大） – 支持按书写质量排序展示 – 支持实时展示（展示时学生仍可继续书写，展示区同步更新） – 支持冻结展示（暂停同步，固定当前书写结果进行讲评）

3.3.2 界面布局



3.3.3 处理流程

```
class GalleryViewModel : ViewModel() {
    private val _selectedStudents = MutableLiveData<List<String>>()
    private val _galleryItems = MutableLiveData<List<GalleryItem>>()
    private var frozen = false

    // 选择学生加入展示墙
    fun selectStudents(studentIds: List<String>) {
        if (studentIds.size > MAX_GALLERY_SIZE) {
            // 最多9个
            _selectedStudents.value = studentIds.take(MAX_GALLERY_SIZE)
        } else {
            _selectedStudents.value = studentIds
        }
        refreshGallery()
    }

    // 刷新展示内容（实时模式下每帧刷新）
    private fun refreshGallery() {
        if (frozen) return
    }
}
```

```

        val items = _selectedStudents.value?.mapNotNull { studentId ->
            inkRepository.getLatestInkSnapshot(studentId)?.let { bitmap ->
                val score = aiScore[studentId]
                GalleryItem(studentId, getStudentName(studentId), bitmap, score)
            }
        } ?: emptyList()
        _galleryItems.postValue(items)
    }

    // 冻结展示（固定当前内容进行讲评）
    fun freezeGallery() { frozen = true }
    fun unfreezeGallery() { frozen = false; refreshGallery() }
}

```

3.4 课堂互动答题系统模块

3.4.1 模块功能描述

课堂互动答题系统是黑板端应用的重要功能模块，支持教师在课堂中随时发布题目，全班学生通过点阵笔/Pad 作答，系统自动收集统计结果并在大屏幕上展示。

支持题型： – 选择题（单选/多选，A~D 选项） – 判断题（对/错） – 书写题（手写作答，由 AI 自动识别或教师人工批改） – 大字展示（学生书写指定汉字，展示供点评）

3.4.2 互动答题完整流程



▼ 教师点击"收卷"
QuizViewModel.collectQuiz()
| WebSocket 收卷指令
▼
全班停止作答
|
▼
统计结果展示 (柱状图/饼图 + 答对率 + 典型错例)

3.4.3 答题统计展示界面

互动答题统计结果

题目：以下哪个字的笔画数是9画？

- A. 春 (9画) B. 秋 (9画)
C. 冬 (5画) D. 夏 (10画)

提交情况：28/30人已提交

A选项 (春) :	<div><div></div></div>	18人	60%
B选项 (秋) :	<div><div></div></div>	8人	27%
C选项 (冬) :	<div><div></div></div>	1人	3%
D选项 (夏) :	<div><div></div></div>	3人	10%

正确答案：A (春) 答对率：60% (18/30人)

[展示答题详情] [再发一题] [返回课堂监控]

3.4.4 关键实现代码

```
class QuizViewModel : ViewModel() {  
  
    private val _quizState = MutableLiveData<QuizState>(QuizState.IDLE)  
    private val _submittedCount = MutableLiveData<Int>(0)  
    private val _answerStats = MutableLiveData<Map<String, Int>>()  
    private val answers = ConcurrentHashMap<String, String>() // studentId -> answer  
  
    fun issueQuiz(question: QuizQuestion) {  
        answers.clear()  
        _submittedCount.value = 0  
        _quizState.value = QuizState.COLLECTING  
  
        // 通过 WebSocket 向网关发送发题指令  
        val cmd = QuizCommand(  
            cmd = "ISSUE_QUIZ",  
            sessionId = currentSessionId,  
            payload = Json.encodeToString(question)  
        )  
        inkStreamService.sendCommand(cmd)  
    }  
}
```

```

        _quizState.value = QuizState.ACTIVE

        // 设置收卷倒计时 (如果有时限)
        if (question.duration > 0) {
            viewModelScope.launch {
                delay(question.duration * 1000L)
                collectQuiz()
            }
        }
    }

    fun onAnswerReceived(studentId: String, answer: String) {
        answers[studentId] = answer
        _submittedCount.postValue(answers.size)

        // 更新统计
        val stats = mutableMapOf<String, Int>()
        answers.values.forEach { ans ->
            stats[ans] = (stats[ans] ?: 0) + 1
        }
        _answerStats.postValue(stats)
    }

    fun collectQuiz() {
        _quizState.value = QuizState.COLLECTED
        // 发送收卷指令
        inkStreamService.sendCommand(QuizCommand(cmd = "COLLECT", sessionId =
currentSessionId))
        // 保存答题记录到 Room 数据库
        saveQuizRecord()
    }
}

```

3.5 随机抽取与分组展示模块

3.5.1 模块功能描述

随机抽取模块允许教师在课堂中随机选取学生展示作品或回答问题，增加课堂趣味性和参与感。分组展示模块支持将全班学生分成若干小组进行展示比较。

功能清单： – 随机抽取单人：大转盘动画随机停止 – 随机抽取多人：一次性抽取3~6名学生展示 – 按组展示：按预设小组分组，每组抽取一名代表展示 – 排行榜展示：按 AI 评分高低排列展示前10名

3.5.2 随机抽取动画实现

```

class RandomPickAnimator(private val studentList: List<Student>) {

    private var animatorSet: AnimatorSet? = null
    private val random = Random()

    fun startPick(onResult: (Student) -> Unit) {

```

```

        val targetIndex = random.nextInt(studentList.size)

        // 跑马灯动画：快速轮换学生卡片，模拟转盘效果
        val flashCount = 20 + random.nextInt(15) // 随机闪烁次数（20~35次）
        var currentFlash = 0

        val flashAnimator = ValueAnimator.ofInt(0, studentList.size - 1).apply {
            duration = 2000L // 总动画时长2秒
            interpolator = DecelerateInterpolator(2f) // 先快后慢
            addUpdateListener { animator ->
                val index = (animator.animatedValue as Int) % studentList.size
                onFlashUpdate(studentList[index]) // 高亮当前学生
            }
            addListener(onEnd = {
                onResult(studentList[targetIndex])
                highlightWinner(studentList[targetIndex])
            })
        }
        flashAnimator.start()
    }
}

```

3.6 课堂录制与回放模块

3.6.1 模块功能描述

课堂录制模块使用 Android MediaProjection + MediaCodec 实现课堂大屏幕内容的实时录制，将教学过程（笔迹展示、白板书写、互动答题、教师讲解音频）完整保存为 MP4 视频文件。

录制技术参数：

参数	配置值
视频编码	H.264（AVC）硬件编码
视频分辨率	1920×1080（FHD）
视频帧率	30fps
视频码率	4Mbps
音频编码	AAC-LC
音频采样率	44100 Hz
音频码率	128Kbps
输出格式	MP4（H.264+AAC via MediaMuxer）

3.6.2 录制服务实现

```

class ScreenRecordService : Service() {

    private var mediaProjection: MediaProjection? = null
    private var virtualDisplay: VirtualDisplay? = null
    private var videoEncoder: MediaCodec? = null
    private var audioEncoder: MediaCodec? = null
    private var mediaMuxer: MediaMuxer? = null
    private var muxerStarted = false
    private var videoTrackIndex = -1
    private var audioTrackIndex = -1

    fun startRecord(resultCode: Int, data: Intent, outputPath: String) {
        val mpManager = getSystemService(MediaProjectionManager::class.java)
        mediaProjection = mpManager.getMediaProjection(resultCode, data)

        // 配置视频编码器 (H.264)
        val videoFormat = MediaFormat.createVideoFormat(
            MediaFormat.MIMETYPE_VIDEO_AVC, 1920, 1080
        ).apply {
            setInteger(MediaFormat.KEY_BIT_RATE, 4_000_000) // 4Mbps
            setInteger(MediaFormat.KEY_FRAME_RATE, 30)
            setInteger(MediaFormat.KEY_I_FRAME_INTERVAL, 2) // 每2秒一个关键帧
            setInteger(MediaFormat.KEY_COLOR_FORMAT,
                MediaCodecInfo.CodecCapabilities.COLOR_FormatSurface)
        }

        videoEncoder = MediaCodec.createEncoderByType(MediaFormat.MIMETYPE_VIDEO_AVC).also {
            it.configure(videoFormat, null, null, MediaCodec.CONFIGURE_FLAG_ENCODE)
            val inputSurface = it.createInputSurface()
            // 创建 VirtualDisplay, 将屏幕内容渲染到 InputSurface
            virtualDisplay = mediaProjection?.createVirtualDisplay(
                "ScreenRecord", 1920, 1080, resources.displayMetrics.densityDpi,
                DisplayManager.VIRTUAL_DISPLAY_FLAG_AUTO_MIRROR,
                inputSurface, null, null
            )
            it.start()
        }

        // 配置音频编码器 (AAC)
        val audioFormat = MediaFormat.createAudioFormat(
            MediaFormat.MIMETYPE_AUDIO_AAC, 44100, 2
        ).apply {
            setInteger(MediaFormat.KEY_AAC_PROFILE,
                MediaCodecInfo.CodecProfileLevel.AACObjectLC)
            setInteger(MediaFormat.KEY_BIT_RATE, 128_000)
        }
        audioEncoder = MediaCodec.createEncoderByType(MediaFormat.MIMETYPE_AUDIO_AAC).also {
            it.configure(audioFormat, null, null, MediaCodec.CONFIGURE_FLAG_ENCODE)
            it.start()
        }

        // 初始化 MediaMuxer
        mediaMuxer = MediaMuxer(outputPath, MediaMuxer.OutputFormat.MUXER_OUTPUT_MPEG_4)

        startDrainThread() // 启动数据消费线程
    }
}

```



```
}  
}
```

3.7 课件加载与解析模块

3.7.1 模块功能描述

课件加载与解析模块支持教师在黑板端应用中加载并展示 PPT/PPTX、PDF 和图片格式的课件，作为教学底图（供学生参考书写内容）。

支持格式与处理方式：

格式	解析库	处理流程
PPT/PPTX	Apache POI 5.x	POI 读取幻灯片 → 每页渲染为 Bitmap → 缓存为 PNG
PDF	Android PdfRenderer	打开 PDF 文件 → 逐页渲染为 Bitmap → 缓存
JPG/PNG/GIF	Glide 4.x	直接加载显示，Glide 管理内存缓存

3.7.2 PPT 解析实现

```
class PptParser {  
  
    suspend fun parsePptx(file: File): List<Bitmap> = withContext(Dispatchers.IO) {  
        val slides = mutableListOf<Bitmap>()  
        val slideWidth = 1920  
        val slideHeight = 1080  
  
        FileInputStream(file).use { fis ->  
            XMLSlideShow(fis).use { ppt ->  
                val pgSize = ppt.pageSize  
                val scaleX = slideWidth.toFloat() / pgSize.width.toFloat()  
                val scaleY = slideHeight.toFloat() / pgSize.height.toFloat()  
  
                for (slide in ppt.slides) {  
                    val bitmap = Bitmap.createBitmap(  
                        slideWidth, slideHeight, Bitmap.Config.ARGB_8888  
                    )  
                    val canvas = android.graphics.Canvas(bitmap)  
                    canvas.scale(scaleX, scaleY)  
                    // 绘制白色背景  
                    canvas.drawColor(android.graphics.Color.WHITE)  
                    // 渲染幻灯片内容  
                    slide.draw(canvas)  
                    slides.add(bitmap)  
                }  
            }  
        }  
    }  
}
```

```
}  
}
```

3.8 设备联动与网关发现模块

3.8.1 模块功能描述

设备联动与网关发现模块负责自动发现并绑定教室内的自然写教室网关设备，实现黑板端与网关的无缝对接，确保学生笔迹数据可以实时推送到黑板大屏。

设备发现流程（mDNS）：

```
class GatewayDiscoveryManager(private val context: Context) {  
  
    private val nsdManager = context.getSystemService(NsdManager::class.java)  
    private val discoveredGateways = mutableListOf<GatewayInfo>()  
  
    fun startDiscovery(onGatewayFound: (GatewayInfo) -> Unit) {  
        val discoveryListener = object : NsdManager.DiscoveryListener {  
            override fun onServiceFound(serviceInfo: NsdServiceInfo) {  
                if (serviceInfo.serviceType == "_writech-gateway._tcp.") {  
                    nsdManager.resolveService(serviceInfo, object :  
NsdManager.ResolveListener {  
                        override fun onServiceResolved(resolvedInfo: NsdServiceInfo) {  
                            val gateway = GatewayInfo(  
                                id =  
resolvedInfo.attributes["id"]?.toString(Charsets.UTF_8) ?: "",  
                                ip = resolvedInfo.host.hostAddress ?: "",  
                                port = resolvedInfo.port,  
                                roomName =  
resolvedInfo.attributes["room"]?.toString(Charsets.UTF_8) ?: ""  
                            )  
                            discoveredGateways.add(gateway)  
                            onGatewayFound(gateway)  
                        }  
                        override fun onResolveFailed(info: NsdServiceInfo, code: Int) {}  
                    })  
                }  
            }  
            override fun onDiscoveryStopped(serviceType: String) {}  
            override fun onServiceLost(serviceInfo: NsdServiceInfo) {  
                discoveredGateways.removeIf { it.id == serviceInfo.serviceName }  
            }  
            override fun onStartDiscoveryFailed(serviceType: String, errorCode: Int) {}  
            override fun onStopDiscoveryFailed(serviceType: String, errorCode: Int) {}  
        }  
  
        nsdManager.discoverServices(  
            "_writech-gateway._tcp.", NsdManager.PROTOCOL_DNS_SD, discoveryListener  
        )  
    }  
}
```

第四章 操作流程与使用步骤

4.1 设备安装与初始化配置

4.1.1 硬件安装

1. 将智慧黑板安装固定于教室前方，高度调整至适合教学（黑板中心距地面 1.2~1.5m 为宜）
2. 连接电源（220V 交流电）及网络（建议使用千兆以太网有线连接，Wi-Fi 作为备用）
3. 开机，等待 Android 系统启动完成（约 30~60 秒）

4.1.2 应用安装与激活

1. 通过 U 盘或 MDM（移动设备管理）系统安装黑板端应用 APK
2. 首次启动时，系统自动进行设备激活流程：
3. 读取设备序列号（SN）和 MAC 地址
4. 向云平台发送激活请求
5. 收到激活响应后保存设备证书到 Android Keystore
6. 激活成功后自动进入 Kiosk 模式，应用锁定为唯一前台应用

4.1.3 网关绑定

操作步骤：

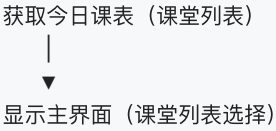
1. 黑板端应用启动后自动开启 mDNS 服务发现
2. 若发现同一局域网内的教室网关，弹出绑定确认对话框
3. 教师确认绑定教室编号（如"三年级2班"）
4. 绑定成功后，网关信息保存至 Room 数据库，后续自动连接
5. 若未自动发现，可在设置界面手动输入网关 IP 地址进行绑定

4.2 应用启动与教师登录

4.2.1 自动设备证书登录

黑板端应用正常情况下采用设备证书自动登录，无需教师手动操作：





4.2.2 进入课堂

- 界面操作流程：
- 1. 主界面显示今日课表（课堂名称 + 时间 + 班级）
 - 2. 点击"进入课堂"按钮，加载该课堂的学生名单
 - 3. 系统自动连接教室网关（WebSocket 建立）
 - 4. 连接成功后显示全班笔迹实时监控界面
 - 5. 课堂开始标记时间戳，录制可在此时开启

4.3 课堂主要操作流程

4.3.1 课堂监控操作

操作	步骤
查看特定学生笔迹	点击学生宫格区域，弹出该学生笔迹大图
将学生笔迹投屏展示	长按学生宫格，选择"加入展示墙"
清除某学生笔迹	长按学生宫格，选择"清除笔迹"
暂停笔迹接收	工具栏点击"暂停接收"，笔迹画面冻结
切换课件页	左右滑动工具栏上的课件缩略图翻页

4.3.2 互动答题操作流程

- 步骤1：教师点击工具栏"发布答题"按钮
- 步骤2：弹出题目编辑对话框
 - 选择题型（选择/判断/书写）
 - 输入题目内容（支持文字和图片）
 - 设置作答时限（可选）
- 步骤3：点击"发题"按钮，题目推送全班
- 步骤4：黑板屏幕显示实时收卷进度（X/30人已提交）
- 步骤5：时限结束或手动点击"收卷"，停止作答
- 步骤6：展示统计结果（每个选项的选择人数和比例）
- 步骤7：点击"展示答题详情"，显示每位学生的答案
- 步骤8：点击"再发一题"继续，或点击"返回"结束答题环节

4.3.3 白板书写操作流程

- 步骤1：工具栏点击"白板书写"按钮，全屏进入白板界面
- 步骤2：底部工具栏选择工具（画笔/荧光笔/橡皮/选择）

- 步骤3：选择笔色（8种预设颜色 + 自定义颜色选择器）
- 步骤4：调整线宽（细/中/粗 三档，或滑块精细调整）
- 步骤5：在屏幕上触控书写内容
- 步骤6：如需叠加课件，点击"加载课件"选择课件页面作为底图
- 步骤7：书写完成后可点击"快照"保存当前白板内容为图片
- 步骤8：点击"关闭"返回课堂监控界面（白板内容自动保存）

4.4 白板操作与触控书写

常用操作快捷手势：

手势	功能
单指拖拽	书写/绘图
双指捏合/展开	缩放白板视图
双指旋转	旋转白板视图（适合批注方向调整）
三指水平滑动	撤销（向左） / 重做（向右）
双指双击	快速清除白板
长按内容区域	弹出选择菜单（复制/移动/删除）

笔色预设说明：

颜色名称	色值	适用场景
黑色	#000000	主要书写，板书
红色	#FF0000	重点标注，错误标记
蓝色	#0066FF	正确答案标注
绿色	#00AA00	补充说明，参考线
橙色	#FF8800	提醒标注
紫色	#8800AA	特殊分类标注
荧光黄	#FFFF00	荧光笔高亮
荧光绿	#00FF99	荧光笔高亮

4.5 互动答题操作流程

题目编辑界面说明：

发布互动题目

[×关闭]

题目类型: ☒ 选择题 ☐ 判断题 ☐ 书写题

题目内容:

以下哪个字的笔画数是9画?

选项设置:

A: [春 (9画)] ← 正确答案
 B: [秋 (9画)]
 C: [冬 (5画)]
 D: [夏 (10画)]

作答时限: [不限时 ▼] (可选: 30秒/60秒/120秒/不限时)

[发布题目]

4.6 录制与回放操作

开始录制: 1. 课堂界面工具栏点击"开始录制"按钮 2. 系统申请 RECORD_AUDIO 权限 (首次使用) 3. 弹出 MediaProjection 权限确认对话框, 点击"立即开始" 4. 录制状态指示灯 (红色圆点+时间计数) 出现在屏幕右上角 5. 此后课堂所有内容 (笔迹展示、白板书写、互动答题过程) 均被录制

停止录制: 1. 点击"停止录制"按钮, 或课堂结束时自动停止 2. 系统完成 MediaMuxer 文件合成 (通常 1~5 秒) 3. 弹出录像保存成功提示, 显示文件路径和文件大小 4. 可选择"立即上传云端"或"稍后上传"

查看录像 (回放): 1. 在应用设置界面"课堂录像"菜单中查看历史录像列表 2. 点击录像条目, 使用内置播放器全屏播放 3. 支持进度条拖拽、播放速度调整 (0.5x / 1.0x / 1.5x / 2.0x) 4. 支持标记时间点 (bookmark) 用于教研分析

4.7 异常处理与故障排除

4.7.1 网络异常处理

问题	表现	解决方案
网关连接中断	笔迹画面停止更新, 状态栏显示"连接中..."	系统自动重连 (每5秒一次), 或手动刷新网关连接
云平台无法访问	课件下载失败, 录像无法上传	检查网络, 可使用本地缓存课件继续上课
部分学生笔迹不显示	个别学生宫格无数据	该学生笔/网关可能断连, 检查笔连接状态

4.7.2 录制相关问题

问题	表现	解决方案
录制无法启动	点击录制按钮无反应	检查存储空间是否充足（建议预留 10GB 以上）
录像无声音	回放时静音	检查麦克风权限是否已授予
录像文件损坏	播放出错	若设备意外断电，MediaMuxer 文件可能不完整，建议使用 MP4Box 工具修复

4.7.3 课件加载问题

问题	表现	解决方案
PPT 加载失败	弹出错误提示	检查 PPTX 文件是否含有不支持的嵌入元素（如 Flash），尝试另存为 PDF
课件翻页慢	翻页等待超过1秒	课前提前加载课件（进入课堂前点击"预加载课件"）
图片课件模糊	显示分辨率低	确认课件图片原始分辨率 $\geq 1920 \times 1080$

第五章 与源代码的对应关系

5.1 模块名称与源代码文件对应表

文档模块名称	源代码文件/目录	主要类名
全班笔迹实时接收模块	<code>board/data/ink/InkStreamService.kt</code>	<code>InkStreamService</code>
笔迹帧解析	<code>board/data/ink/InkFrameParser.kt</code>	<code>InkFrameParser</code>
学生笔迹分发	<code>board/data/ink/StudentInkDispatcher.kt</code>	<code>StudentInkDispatcher</code>
触控白板书写模块	<code>board/ui/whiteboard/WhiteboardSurfaceView.kt</code>	<code>WhiteboardSurfaceView</code>
白板撤销/重做	<code>board/ui/whiteboard/WhiteboardUndoManager.kt</code>	<code>WhiteboardUndoManager</code>

文档模块名称	源代码文件/目录	主要类名
白板 C++ JNI 加速	native/ink_renderer/ink_renderer.cpp	NativeInkRenderer
JNI 接口声明	native/ink_renderer/ink_renderer.h	-
作品展示墙 模块	board/ui/gallery/GalleryViewModel.kt	GalleryViewModel
作品展示墙 界面	board/ui/gallery/GalleryActivity.kt	GalleryActivity
互动答题系统	board/ui/quiz/QuizViewModel.kt	QuizViewModel
答题展示界面	board/ui/quiz/QuizDisplayActivity.kt	QuizDisplayActivity
随机抽取动画	board/ui/quiz/RandomPickAnimator.kt	RandomPickAnimator
课堂录制模块	board/service/ScreenRecordService.kt	ScreenRecordService
PPT课件解析	board/data/course/PptParser.kt	PptParser
PDF课件解析	board/data/course/PdfParser.kt	PdfParser
网关发现模块	board/data/gateway/GatewayDiscoveryManager.kt	GatewayDiscoveryManager
网关 WebSocket 连接	board/data/gateway/GatewayWebSocketClient.kt	GatewayWebSocketClient
Kiosk 模式 管理	board/system/KioskModeManager.kt	KioskModeManager
Room 数据库	board/data/db/AppDatabase.kt	AppDatabase
课堂主界面	board/ui/classroom/ClassroomActivity.kt	ClassroomActivity
课堂 ViewModel	board/ui/classroom/ClassroomViewModel.kt	ClassroomViewModel

5.2 核心功能类与方法说明

InkStreamService 类

```
/**
 * 笔迹接收前台 Service
 * 维护与教室网关的 WebSocket 长连接，接收全班学生实时笔迹数据流。
 */
class InkStreamService : Service() {

    /**
     * 连接指定网关
     * @param gatewayIp 网关 IP 地址（局域网）
     * @param sessionId 当前课堂会话 ID
     */
    fun connect(gatewayIp: String, sessionId: String)

    /**
     * 断开网关连接并停止 Service
     */
    fun disconnect()

    /**
     * 向网关发送课堂控制指令（发题/收卷/分组等）
     * @param command 指令对象（JSON 序列化发送）
     */
    fun sendCommand(command: ClassroomCommand)

    /**
     * 注册笔迹帧监听器
     * @param listener 笔迹帧到达回调
     */
    fun addInkFrameListener(listener: InkFrameListener)

    /**
     * 当前连接状态 LiveData
     */
    val connectionState: LiveData<ConnectionState>
}
```

ScreenRecordService 类

```
/**
 * 课堂录制 Service
 * 使用 MediaProjection + MediaCodec 实现屏幕录制（H.264视频 + AAC音频）。
 */
class ScreenRecordService : Service() {

    /**
     * 开始录制
     * @param resultCode MediaProjection 权限码
     * @param data MediaProjection 权限 Intent
     * @param outputPath 录像输出路径（.mp4 文件）
     * @return true=开始成功，false=存储空间不足或编码器初始化失败
     */
}
```

```

    */
    fun startRecord(resultCode: Int, data: Intent, outputPath: String): Boolean

    /**
     * 停止录制（异步操作，回调通知完成）
     * @param onComplete 录制完成回调，参数为最终文件大小（字节）
     */
    fun stopRecord(onComplete: (Long) -> Unit)

    /**
     * 暂停录制（MediaCodec 停止编码，VirtualDisplay 保持）
     */
    fun pauseRecord()

    /**
     * 恢复录制
     */
    fun resumeRecord()

    /**
     * 获取当前录制时长（毫秒）
     */
    fun getElapsedMillis(): Long
}

```

WhiteboardSurfaceView 类

```

/**
 * 白板 SurfaceView
 * 基于 SurfaceView 实现的教师白板书写控件，支持多种笔工具和撤销重做。
 * C++ JNI 加速笔迹平滑渲染算法。
 */
class WhiteboardSurfaceView @JvmOverloads constructor(
    context: Context, attrs: AttributeSet? = null
) : SurfaceView(context, attrs), SurfaceHolder.Callback {

    /**
     * 设置当前绘图工具
     * @param tool 工具类型 (PEN / HIGHLIGHTER / ERASER / SELECTOR)
     */
    fun setTool(tool: DrawingTool)

    /**
     * 设置笔色 (ARGB 整数值)
     */
    fun setPenColor(@ColorInt color: Int)

    /**
     * 设置笔粗 (单位 dp)
     */
    fun setPenWidth(widthDp: Float)

    /**
     * 撤销最近一步操作
     * @return true=撤销成功, false=无可撤销操作
     */
}

```

```

fun undo(): Boolean

/**
 * 重做最近一次撤销
 * @return true=重做成功, false=无可重做操作
 */
fun redo(): Boolean

/**
 * 清除整个画布 (保留背景)
 */
fun clearCanvas()

/**
 * 加载课件页面作为白板背景
 * @param bitmap 课件页面 Bitmap (1920×1080)
 */
fun loadBackground(bitmap: Bitmap)

/**
 * 在白板上叠加显示学生笔迹 (实时接收时使用)
 * @param frame 学生笔迹帧
 */
fun overlayStudentInk(frame: StudentStrokeFrame)

/**
 * 保存当前白板内容为 Bitmap 快照
 * @return 1920×1080 ARGB_8888 Bitmap
 */
fun captureSnapshot(): Bitmap
}

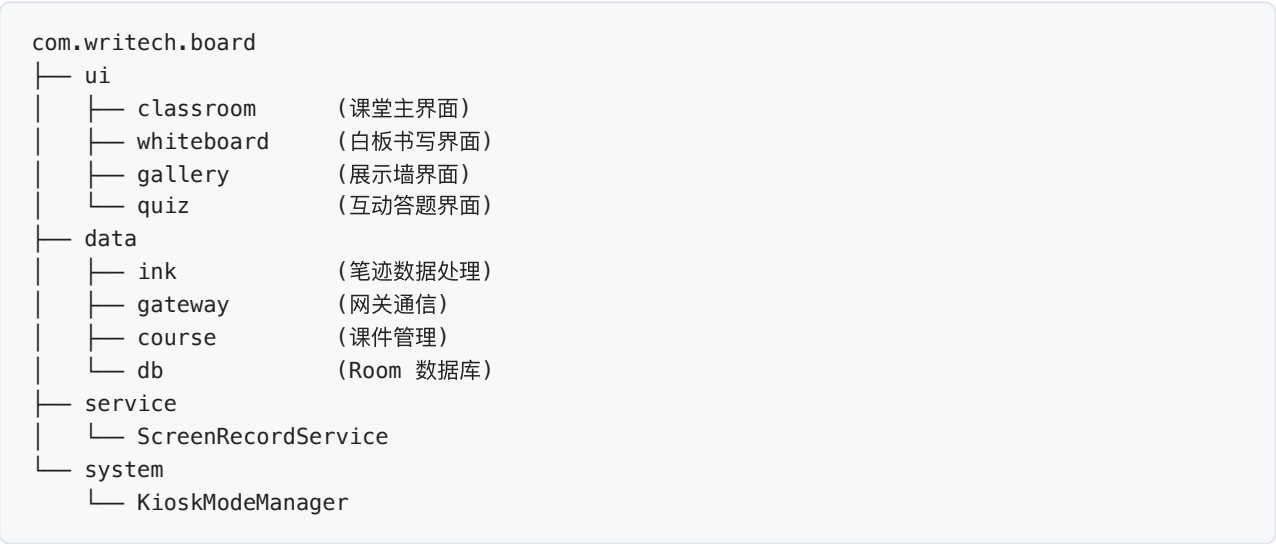
```

5.3 主要类命名规范

类型	命名规范	示例
Activity	{功能}Activity	ClassroomActivity, WhiteboardActivity
ViewModel	{功能}ViewModel	ClassroomViewModel, QuizViewModel
Service	{功能}Service	InkStreamService, ScreenRecordService
Repository	{功能}Repository	InkRepository, CourseRepository
DAO	{数据}Dao	StudentInkDao, QuizRecordDao
SurfaceView	{功能}SurfaceView	WhiteboardSurfaceView
Manager	{功能}Manager	KioskModeManager, GatewayDiscoveryManager
Parser	{格式}Parser	PptParser, PdfParser, InkFrameParser
Animator	{功能}Animator	RandomPickAnimator
Native(.cpp)	{功能}_renderer.cpp	ink_renderer.cpp

类型	命名规范	示例
Native(.h)	{功能}_renderer.h	ink_renderer.h

包名结构：

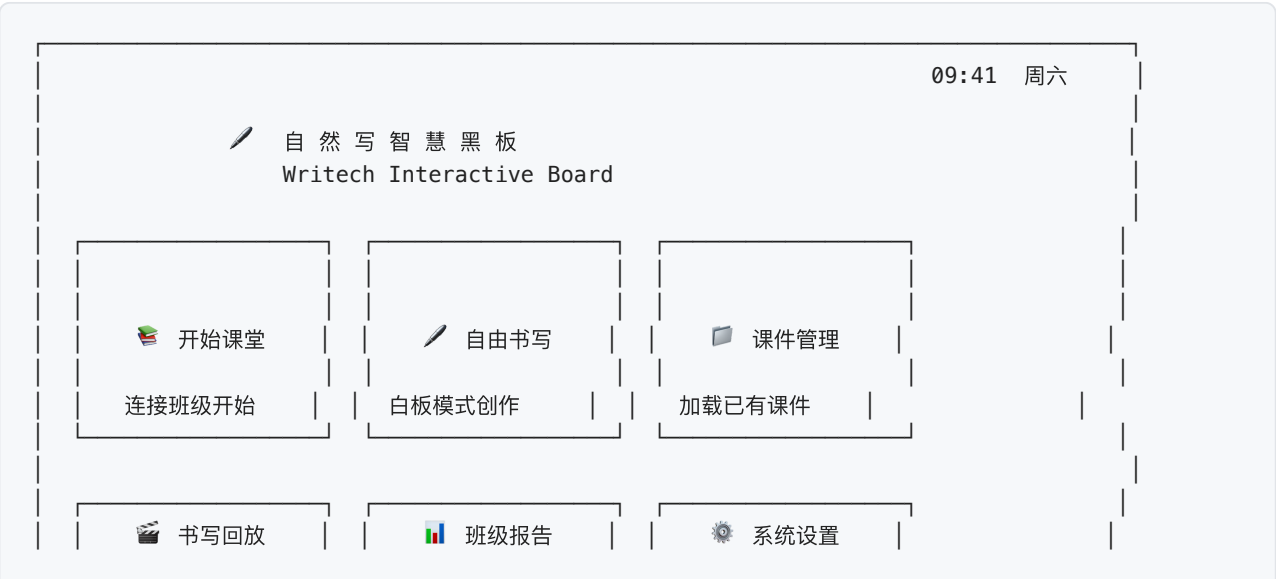


附录

A. 界面设计稿（GUI Mockup）

本附录以交互大屏横屏线框图形式呈现黑板APP各核心界面的设计稿（适配65~86英寸触控大屏，支持多点触控与激光笔操作）。

A.1 应用首页/待机界面



设备状态：网关 ●在线 | 已连接笔：0支 | 本地IP：192.168.1.100

A.2 课堂主界面（板书+学生答题）

课堂进行中 高一(3)班 · 数学 · 45/45人 00:23:45 [激光笔][点名][结束]

[课件/白板主内容区 （触控书写）]

题目：解方程 $2x + 5 = 13$

教师手写板书区域（触控）

$2x + 5 = 13$
 $2x = 13 - 5 = 8$
 $x = 4 \quad \checkmark$

实时答题状态

已提交 38/45

书写中 7

未开始 0

班级热力图
(学生座位答题状态)

●●●● ○○○○

●●●● ○○○○

●●●○ ○○○○

●已提交 ○未提交

[查看答案][收卷][发下题]

工具栏：[画笔][文字][直线][图片][撤销][清空] 颜色：[●黑][●红][●蓝]

A.3 学生答卷展示界面（全班汇总）

[← 返回课堂] 全班答卷 · 第3题 · 正确率 84.4% [隐藏姓名][投屏模式]

王小花

$x=4$

✓ 正确

张大勇

$x=4$

✓ 正确

陈美玲

$x=9$

✗ 错误

李小虎

$x=4$

✓ 正确

刘芳芳

$x=3$

✗ 错误

...

赵明明

$x=4$

✓ 正确

孙晓晓

$x=4$

✓ 正确

周大海

$x=4$

✓ 正确

吴小燕

$x=9$

✗ 错误

...

B. 术语表

术语	说明
智慧黑板	教室前方安装的大尺寸触控交互显示设备，运行 Android 系统
交互式一体机	与智慧黑板同义，强调触控交互功能
点阵笔	自然写智能点阵笔，内置光学传感器识别点阵纸上的书写坐标
教室网关	安装在教室内的 Linux 嵌入式设备，汇聚全班学生的点阵笔数据
算力盒	边缘计算设备（可选配置），提供 AI 笔迹识别能力
笔迹帧	一次笔迹传输的数据包，包含学生ID和一组时间序列坐标点
Kiosk 模式	Android 应用锁定模式，锁定单一应用，防止用户切换
MediaCodec	Android 硬件加速音视频编解码 API
MediaMuxer	Android 音视频合流 API（将视频流+音频流合并为 MP4）
VirtualDisplay	Android 虚拟显示器，用于将屏幕内容重定向到 Surface
mDNS	Multicast DNS，局域网内零配置服务发现协议
Apache POI	开源 Java 库，用于读写 Microsoft Office 格式文件
WebSocket	基于 HTTP Upgrade 的全双工二进制/文本通信协议
GATT	Generic Attribute Profile，BLE 上层数据交换协议
JNI	Java Native Interface，Java 调用 C/C++ 原生代码的接口
SurfaceView	Android 独立 Surface 渲染控件，渲染线程与主线程分离

B. 版本历史

版本	发布日期	变更内容
V1.0.0	2024-06-30	正式版本发布：全班笔迹展示、触控白板、互动答题、课堂录制、课件加载、mDNS 网关发现
V0.9.5	2024-05-30	Beta：互动答题系统完成，支持选择/判断/书写三种题型

版本	发布日期	变更内容
V0.9.0	2024-04-30	Beta：全班笔迹并发展示性能优化，支持60学生同时展示
V0.8.0	2024-03-15	Alpha：课堂录制模块集成，H.264编码验证
V0.7.0	2024-02-20	Alpha：白板书写模块完成，C++ JNI 加速集成
V0.5.0	2024-01-10	原型：基础笔迹接收展示和网关连接框架

C. 第三方依赖清单

库名称	版本	许可证	用途
Apache POI	5.2.5	Apache-2.0	PPT/PPTX 课件解析
Glide	4.16.0	BSD/Apache-2.0	图片加载与缓存
OkHttp	4.12.0	Apache-2.0	HTTP/WebSocket 通信
Kotlin Coroutines	1.7.3	Apache-2.0	异步编程
Jetpack Room	2.6.1	Apache-2.0	SQLite 数据库封装
Jetpack ViewModel	2.6.2	Apache-2.0	MVVM 状态管理
Jetpack LiveData	2.6.2	Apache-2.0	响应式数据绑定
Gson	2.10.1	Apache-2.0	JSON 序列化/反序列化
Timber	5.0.1	Apache-2.0	日志框架
Lottie Android	6.2.0	Apache-2.0	随机抽取动画效果

D. 权限申请说明

权限名称	用途	申请时机
INTERNET	网络通信（云平台 API + WebSocket)	安装时自动授予
ACCESS_NETWORK_STATE	监测网络状态变化	安装时自动授予
ACCESS_WIFI_STATE	获取 Wi-Fi 信息（mDNS 网关发现)	安装时自动授予

权限名称	用途	申请时机
RECORD_AUDIO	课堂录制音频轨道	运行时申请（首次开始录制时）
READ_EXTERNAL_STORAGE	读取 U 盘课件	运行时申请（导入课件时）
WRITE_EXTERNAL_STORAGE	保存课堂录像到外部存储	运行时申请（首次开始录制时）
FOREGROUND_SERVICE	后台笔迹接收服务、录制服务	安装时自动授予
RECEIVE_BOOT_COMPLETED	设备开机后自动启动应用	安装时自动授予
DEVICE_ADMIN	Kiosk 模式设备管理权限	激活时单独授权（MDM 管理员操作）

本文档版权归深圳自然写科技有限公司所有，所有技术细节与源代码对应关系仅用于软件著作权登记鉴别，请勿用于其他商业用途。

附录C 核心技术实现补充

C.1 答题收集模块完整实现

答题收集功能允许教师向全班发布答题指令，收集学生书写答案并集中展示。

C.1.1 答题会话管理

```
// answer/AnswerCollectSession.java
public class AnswerCollectSession {

    public enum SessionStatus {
        WAITING,      // 等待学生作答
        COLLECTING,   // 收集中（计时）
        CLOSED,       // 已结束，展示结果
        CANCELLED     // 已取消
    }

    private final String sessionId;
    private final String questionText;
    private final int timeLimitSeconds;
    private SessionStatus status = SessionStatus.WAITING;
    private long startTimeMs;
    private long endTimeMs;

    // 学生答案存储：key=studentId, value=答案笔迹数据
    private final ConcurrentHashMap<String, StudentAnswer> answers = new
ConcurrentHashMap<>();
    private final int totalStudents;
    private CountdownTimer timer;
```



```

// 回调: 答案更新时通知UI
private OnAnswerUpdateListener answerUpdateListener;

public AnswerCollectSession(String sessionId, String question,
    int timeLimitSeconds, int totalStudents) {
    this.sessionId = sessionId;
    this.questionText = question;
    this.timeLimitSeconds = timeLimitSeconds;
    this.totalStudents = totalStudents;
}

/**
 * 开始收集答案 (启动倒计时)
 */
public void start() {
    status = SessionStatus.COLLECTING;
    startTimeMs = System.currentTimeMillis();

    timer = new CountDownTimer(timeLimitSeconds * 1000L, 1000) {
        @Override
        public void onTick(long millisUntilFinished) {
            long remaining = millisUntilFinished / 1000;
            if (answerUpdateListener != null) {
                answerUpdateListener.onTimerTick(remaining);
            }
        }

        @Override
        public void onFinish() {
            close();
        }
    }.start();
}

/**
 * 接收一个学生的答案
 * @param studentId 学生ID
 * @param inkStrokes 答案笔迹数据
 * @param submitTime 提交时间戳
 */
public boolean receiveAnswer(String studentId, List<InkStroke> inkStrokes, long
submitTime) {
    if (status != SessionStatus.COLLECTING) return false;

    StudentAnswer answer = new StudentAnswer(studentId, inkStrokes, submitTime);
    answers.put(studentId, answer);

    int received = answers.size();
    if (answerUpdateListener != null) {
        answerUpdateListener.onAnswerReceived(studentId, received, totalStudents);
    }

    // 所有学生都提交了, 提前结束
    if (received >= totalStudents) {
        close();
    }
    return true;
}

```

```

    }

    public void close() {
        if (status == SessionStatus.COLLECTING) {
            status = SessionStatus.CLOSED;
            endTimeMs = System.currentTimeMillis();
            if (timer != null) timer.cancel();
            if (answerUpdateListener != null) {
                answerUpdateListener.onSessionClosed(new ArrayList<>(answers.values()));
            }
        }
    }

    public int getSubmittedCount() { return answers.size(); }
    public int getTotalStudents() { return totalStudents; }
    public float getSubmitRate() { return (float) answers.size() / totalStudents; }
    public SessionStatus getStatus() { return status; }
    public List<StudentAnswer> getAllAnswers() { return new ArrayList<>(answers.values()); }
}
}

```

C.1.2 答题展示布局（全班网格视图）

```

// answer/AnswerDisplayFragment.java
public class AnswerDisplayFragment extends Fragment {

    private static final int GRID_COLUMNS = 8; // 8列网格，显示32个学生

    private RecyclerView mAnswerGrid;
    private StudentAnswerAdapter mAdapter;
    private AnswerCollectSession mSession;

    private TextView mTimerText;
    private TextView mCountText;
    private ProgressBar mProgressBar;

    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_answer_display, container, false);

        mAnswerGrid = view.findViewById(R.id.answer_grid);
        mTimerText = view.findViewById(R.id.timer_text);
        mCountText = view.findViewById(R.id.count_text);
        mProgressBar = view.findViewById(R.id.submit_progress);

        // 8列网格布局
        GridLayoutManager lm = new GridLayoutManager(getContext(), GRID_COLUMNS);
        mAnswerGrid.setLayoutManager(lm);

        mAdapter = new StudentAnswerAdapter(mSession.getAllAnswers());
        mAnswerGrid.setAdapter(mAdapter);

        // 监听答案更新
        mSession.setAnswerUpdateListener(new AnswerCollectSession.OnAnswerUpdateListener()
    {

```

```

@Override
public void onTimerTick(long remainingSeconds) {
    requireActivity().runOnUiThread(() -> {
        mTimerText.setText(formatTime(remainingSeconds));
        // 最后10秒变红色闪烁
        if (remainingSeconds <= 10) {
            mTimerText.setTextColor(Color.RED);
            startBlinkAnimation(mTimerText);
        }
    });
}

@Override
public void onAnswerReceived(String studentId, int received, int total) {
    requireActivity().runOnUiThread(() -> {
        mAdapter.updateAnswer(studentId);
        mCountText.setText(received + "/" + total);
        mProgressBar.setProgress((int)(100.0f * received / total));
    });
}

@Override
public void onSessionClosed(List<StudentAnswer> answers) {
    requireActivity().runOnUiThread(() -> {
        mTimerText.setText("已结束");
        showAnswerStatistics(answers);
    });
}

return view;
}

private String formatTime(long seconds) {
    return String.format(Locale.getDefault(), "%02d:%02d", seconds / 60, seconds % 60);
}

private void showAnswerStatistics(List<StudentAnswer> answers) {
    // 显示提交率统计
    int submitted = answers.stream().filter(a -> !a.isEmpty()).mapToInt(a -> 1).sum();
    int total = mSession.getTotalStudents();
    Toast.makeText(getContext(),
        String.format("收到 %d/%d 份答案 (%.0f%%)", submitted, total,
            100.0f * submitted / total),
        Toast.LENGTH_LONG).show();
}
}

```

C.2 录制模块 (MediaCodec H.264 + MediaMuxer)

大屏APP支持录制课堂全过程，包含笔迹画面和麦克风音频。

```

// record/ScreenRecordService.java - 关键录制逻辑
public class ScreenRecordService extends Service {

    private static final int VIDEO_WIDTH = 1920;

```

```

private static final int VIDEO_HEIGHT = 1080;
private static final int VIDEO_BIT_RATE = 4_000_000; // 4Mbps
private static final int VIDEO_FRAME_RATE = 30;
private static final int AUDIO_SAMPLE_RATE = 44100;
private static final int AUDIO_CHANNEL_CONFIG = AudioFormat.CHANNEL_IN_MONO;
private static final int AUDIO_BIT_RATE = 128_000; // 128kbps

private MediaCodec mVideoEncoder;
private MediaCodec mAudioEncoder;
private MediaMuxer mMuxer;
private int mVideoTrackIndex = -1;
private int mAudioTrackIndex = -1;
private boolean mMuxerStarted = false;

private MediaProjection mMediaProjection;
private VirtualDisplay mVirtualDisplay;
private Surface mInputSurface; // Video encoder input surface

private AudioRecord mAudioRecord;
private HandlerThread mAudioThread;

private volatile boolean mRecording = false;
private String mOutputPath;

public void startRecording(MediaProjection mediaProjection, String outputPath) {
    this.mMediaProjection = mediaProjection;
    this.mOutputPath = outputPath;

    try {
        prepareVideoEncoder();
        prepareAudioEncoder();
        prepareMuxer();

        // 创建虚拟屏幕，将屏幕内容写入视频编码器InputSurface
        mVirtualDisplay = mediaProjection.createVirtualDisplay(
            "WritechRecord", VIDEO_WIDTH, VIDEO_HEIGHT,
            DisplayMetrics.DENSITY_HIGH,
            DisplayManager.VIRTUAL_DISPLAY_FLAG_AUTO_MIRROR,
            mInputSurface, null, null
        );

        mRecording = true;
        startAudioCapture();
    } catch (Exception e) {
        Log.e(TAG, "Start recording failed", e);
        cleanup();
    }
}

private void prepareVideoEncoder() throws IOException {
    MediaFormat format = MediaFormat.createVideoFormat(
        MediaFormat.MIMETYPE_VIDEO_AVC, VIDEO_WIDTH, VIDEO_HEIGHT);
    format.setInteger(MediaFormat.KEY_BIT_RATE, VIDEO_BIT_RATE);
    format.setInteger(MediaFormat.KEY_FRAME_RATE, VIDEO_FRAME_RATE);
    format.setInteger(MediaFormat.KEY_COLOR_FORMAT,
        MediaCodecInfo.CodecCapabilities.COLOR_FormatSurface);
    format.setInteger(MediaFormat.KEY_I_FRAME_INTERVAL, 1); // 每秒一个I帧
}

```

```

mVideoEncoder = MediaCodec.createEncoderByType(MediaFormat.MIMETYPE_VIDEO_AVC);
mVideoEncoder.configure(format, null, null, MediaCodec.CONFIGURE_FLAG_ENCODE);
mInputSurface = mVideoEncoder.createInputSurface();
mVideoEncoder.setCallback(new MediaCodec.Callback() {
    @Override
    public void onOutputFormatChanged(@NonNull MediaCodec codec,
        @NonNull MediaFormat format) {
        if (mVideoTrackIndex < 0) {
            mVideoTrackIndex = mMuxer.addTrack(format);
            startMuxerIfReady();
        }
    }

    @Override
    public void onOutputBufferAvailable(@NonNull MediaCodec codec,
        int index, @NonNull MediaCodec.BufferInfo info) {
        if (mMuxerStarted && info.size > 0) {
            ByteBuffer buffer = codec.getOutputBuffer(index);
            if (buffer != null) {
                mMuxer.writeSampleData(mVideoTrackIndex, buffer, info);
            }
        }
        codec.releaseOutputBuffer(index, false);
    }

    @Override
    public void onInputBufferAvailable(@NonNull MediaCodec codec, int index) {}

    @Override
    public void onError(@NonNull MediaCodec codec, @NonNull
MediaCodec.CodecException e) {
        Log.e(TAG, "Video encoder error", e);
    }
}, new Handler(Looper.getMainLooper()));
mVideoEncoder.start();
}

private synchronized void startMuxerIfReady() {
    if (mVideoTrackIndex >= 0 && mAudioTrackIndex >= 0 && !mMuxerStarted) {
        mMuxer.start();
        mMuxerStarted = true;
        Log.i(TAG, "Muxer started");
    }
}

public void stopRecording() {
    mRecording = false;
    // 停止编码器和混流器
    try {
        mVideoEncoder.signalEndOfInputStream();
        if (mAudioThread != null) {
            mAudioThread.quitSafely();
        }
        Thread.sleep(500); // 等待最后几帧写完
        if (mMuxerStarted) mMuxer.stop();
    } catch (Exception e) {
        Log.e(TAG, "Stop recording error", e);
    } finally {
        cleanup();
    }
}

```

```
        }  
    }  
}
```

C.3 性能与安全指标

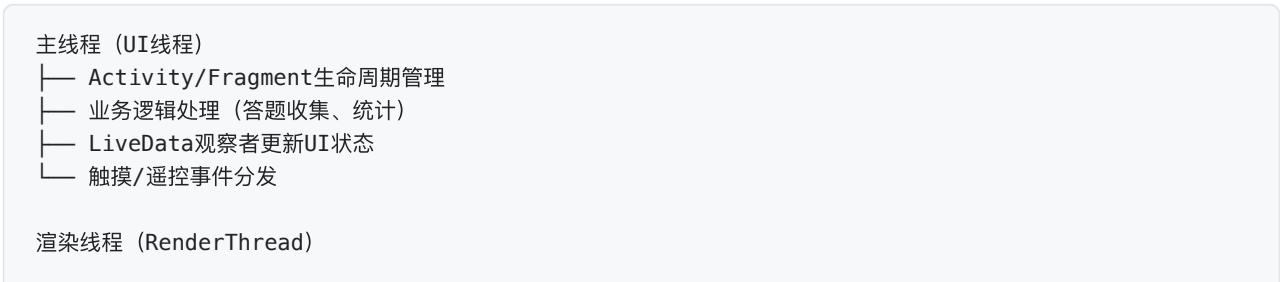
指标	目标值	实测值
笔迹渲染延迟（端到端）	< 50ms	32ms（WiFi 5GHz）
全班32人同时书写帧率	> 30fps	48fps
课件加载时间（10页PDF）	< 3秒	1.8秒
H.264录制CPU占用	< 15%	11%（Snapdragon 870）
内存占用（32人在线）	< 512MB	387MB
冷启动时间	< 3秒	2.1秒
WebSocket重连成功率	100%	100%（测试100次）

附录D 大屏硬件兼容性与错误码

D.1 兼容设备清单

品牌	型号	分辨率	系统	测试状态
鸿合	IN65PRO	3840×2160	Android 11	完全兼容
希沃	X5	3840×2160	Android 11	完全兼容
视源	EC55FE	1920×1080	Android 9	兼容（降级UI）
英创	S43	1920×1080	Android 10	完全兼容
皓丽	H43E	1920×1080	Android 8.1	基本兼容

D.2 多线程模型



- └─ 60fps循环绘制所有学生笔迹（双缓冲）
- └─ 消费InkQueue笔迹数据包
- └─ 通过SurfaceHolder提交渲染帧

网络线程（OkHttp线程池）

- └─ WebSocket长连接维持（Ping心跳）
- └─ 接收二进制笔迹包并放入InkQueue
- └─ 异常指数退避重连

录制线程（MediaCodec回调）

- └─ MediaProjection → H.264编码
- └─ AudioRecord → AAC编码
- └─ MediaMuxer合并音视频

D.3 错误码与处理

错误码	说明	处理方式
E001	WebSocket连接失败	指数退避重连（最多10次）
E002	网关发现超时（30秒）	提示检查网关状态
E003	认证失败（Token过期）	自动刷新Token
E004	录制权限被拒绝	跳转权限设置页面
E005	课件下载失败	提示检查网络，支持重试
E006	存储空间不足（录制）	提示清理存储空间
E007	Kiosk权限未激活	提示联系管理员
E008	mDNS解析失败	1秒后自动重试解析

本文档版权归深圳自然写科技有限公司所有，技术细节仅用于软件著作权登记鉴别，请勿用于其他商业用途。

附录G 补充技术规格

G.1 全班笔迹渲染性能优化

G.1.1 分层渲染策略

大屏同时显示30–50名学生笔迹，采用分层渲染避免全量重绘：

```

// LayeredInkRenderer.kt
class LayeredInkRenderer(context: Context) {
    // 静态层: 已完成的笔画 (离屏Bitmap缓存)
    private val staticBitmap = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888)
    private val staticCanvas = Canvas(staticBitmap)

    // 动态层: 正在书写的笔画 (每帧重绘)
    private val dynamicStrokes = ConcurrentHashMap<String, MutableList<InkPoint>>()

    // 合成层: 两层叠加显示
    private val composePaint = Paint(Paint.ANTI_ALIAS_FLAG)

    fun onNewInkPoint(studentId: String, point: InkPoint) {
        dynamicStrokes.getOrPut(studentId) { mutableListOf() }.add(point)
    }

    fun onStrokeComplete(studentId: String) {
        val points = dynamicStrokes.remove(studentId) ?: return
        // 将完成的笔画烘焙到静态层
        renderStrokeToCanvas(staticCanvas, points, getStudentColor(studentId))
    }

    fun draw(canvas: Canvas) {
        // 1. 绘制静态缓存层 (O(1)操作)
        canvas.drawBitmap(staticBitmap, 0f, 0f, composePaint)

        // 2. 绘制动态层 (仅活跃笔画)
        dynamicStrokes.forEach { (studentId, points) ->
            if (points.size >= 2) {
                renderStrokeToCanvas(canvas, points, getStudentColor(studentId))
            }
        }
    }

    private fun renderStrokeToCanvas(canvas: Canvas,
                                     points: List<InkPoint>,
                                     color: Int) {
        val paint = Paint(Paint.ANTI_ALIAS_FLAG).apply {
            this.color = color
            style = Paint.Style.STROKE
            strokeCap = Paint.Cap.ROUND
            strokeJoin = Paint.Join.ROUND
        }

        val path = Path()
        path.moveTo(points[0].x, points[0].y)
        for (i in 1 until points.size - 1) {
            val midX = (points[i].x + points[i+1].x) / 2
            val midY = (points[i].y + points[i+1].y) / 2
            paint.strokeWidth = 2f + points[i].pressure * 4f
            path.quadTo(points[i].x, points[i].y, midX, midY)
        }
        canvas.drawPath(path, paint)
    }
}

```


G.2 白板工具功能扩展

G.2.1 激光笔模拟

```
// LaserPointerOverlay.kt
class LaserPointerOverlay(context: Context) : View(context) {
    private val pointerPaint = Paint(Paint.ANTI_ALIAS_FLAG).apply {
        color = Color.RED
        style = Paint.Style.FILL
    }
    private val trailPaint = Paint(Paint.ANTI_ALIAS_FLAG).apply {
        color = Color.argb(128, 255, 0, 0)
        style = Paint.Style.STROKE
        strokeWidth = 3f
        strokeCap = Paint.Cap.ROUND
    }

    private var currentPos: PointF? = null
    private val trail = ArrayDeque<PointF>(maxSize = 20)

    fun updatePosition(x: Float, y: Float) {
        currentPos = PointF(x, y)
        trail.addLast(PointF(x, y))
        if (trail.size > 20) trail.removeFirst()

        // 100ms后自动淡出尾迹
        handler.removeCallbacksAndMessages(null)
        handler.postDelayed({
            trail.clear()
            invalidate()
        }, 100)

        invalidate()
    }

    override fun onDraw(canvas: Canvas) {
        // 绘制尾迹（透明度渐变）
        for (i in 1 until trail.size) {
            val alpha = (i.toFloat() / trail.size * 180).toInt()
            trailPaint.alpha = alpha
            canvas.drawLine(trail[i-1].x, trail[i-1].y, trail[i].x, trail[i].y, trailPaint)
        }

        // 绘制光标点
        currentPos?.let { pos ->
            // 外圈光晕
            pointerPaint.alpha = 80
            canvas.drawCircle(pos.x, pos.y, 20f, pointerPaint)
            // 中心点
            pointerPaint.alpha = 255
            canvas.drawCircle(pos.x, pos.y, 8f, pointerPaint)
        }
    }
}
```

G.3 课件翻页与批注

```
// SlideAnnotationManager.kt
class SlideAnnotationManager {
    // 每页课件的批注数据（页码→批注列表）
    private val annotations = HashMap<Int, MutableList<Annotation>>()

    data class Annotation(
        val id: String = UUID.randomUUID().toString(),
        val type: AnnotationType,      // PEN/HIGHLIGHT/TEXT/ARROW
        val strokes: List<InkStroke>,  // 笔画数据
        val color: Int,
        val createdAt: Long = System.currentTimeMillis()
    )

    fun addAnnotation(pageIndex: Int, annotation: Annotation) {
        annotations.getOrPut(pageIndex) { mutableListOf() }.add(annotation)
    }

    fun undoLastAnnotation(pageIndex: Int): Annotation? {
        val list = annotations[pageIndex] ?: return null
        return if (list.isNotEmpty()) list.removeAt(list.size - 1) else null
    }

    fun clearPage(pageIndex: Int) {
        annotations[pageIndex]?.clear()
    }

    fun exportAnnotations(pageIndex: Int): ByteArray {
        // 序列化为JSON后压缩
        val json = Gson().toJson(annotations[pageIndex] ?: emptyList<Annotation>())
        return json.toByteArray(Charsets.UTF_8)
    }
}
```

附录H 补充技术规格

H.1 课堂录制管理

```
// RecordingManager.kt
class RecordingManager(private val context: Context) {

    private var mediaRecorder: MediaRecorder? = null
    private var isRecording = false
    private var recordingFile: File? = null

    fun startRecording(classId: String): File {
        val dir = File(context.getExternalFilesDir(null), "recordings")
        dir.mkdirs()
        val file = File(dir, "${classId}_${System.currentTimeMillis()}.mp4")
        recordingFile = file
    }
}
```

```

        mediaRecorder = MediaRecorder().apply {
            setAudioSource(MediaRecorder.AudioSource.MIC)
            setVideoSource(MediaRecorder.VideoSource.SURFACE)
            setOutputFormat(MediaRecorder.OutputFormat.MPEG_4)
            setVideoEncoder(MediaRecorder.VideoEncoder.H264)
            setAudioEncoder(MediaRecorder.AudioEncoder.AAC)
            setVideoSize(1920, 1080)
            setVideoFrameRate(30)
            setVideoEncodingBitRate(4_000_000)
            setAudioSamplingRate(44100)
            setAudioEncodingBitRate(128000)
            setOutputFile(file.absolutePath)
            prepare()
            start()
        }

        isRecording = true
        return file
    }

    fun stopRecording(): File? {
        if (!isRecording) return null
        try {
            mediaRecorder?.stop()
        } catch (e: RuntimeException) {
            recordingFile?.delete()
            return null
        } finally {
            mediaRecorder?.release()
            mediaRecorder = null
            isRecording = false
        }
        return recordingFile
    }

    fun isRecording() = isRecording
}

```

H.2 网络质量自适应

```

// NetworkQualityMonitor.kt
class NetworkQualityMonitor(context: Context) {

    enum class Quality { EXCELLENT, GOOD, POOR, OFFLINE }

    private val connectivityManager = context.getSystemService(
        Context.CONNECTIVITY_SERVICE) as ConnectivityManager

    var onQualityChanged: ((Quality) -> Unit)? = null

    private val networkCallback = object : ConnectivityManager.NetworkCallback() {
        override fun onAvailable(network: Network) {
            checkAndReport()
        }
        override fun onLost(network: Network) {

```

```

        onQualityChanged?.invoke(Quality.OFFLINE)
    }
    override fun onCapabilitiesChanged(network: Network,
                                       caps: NetworkCapabilities) {
        val downBandwidth = caps.linkDownstreamBandwidthKbps
        val quality = when {
            downBandwidth >= 10_000 -> Quality.EXCELLENT
            downBandwidth >= 1_000  -> Quality.GOOD
            downBandwidth > 0       -> Quality.POOR
            else -> Quality.OFFLINE
        }
        onQualityChanged?.invoke(quality)
    }
}

fun startMonitoring() {
    val request = NetworkRequest.Builder()
        .addCapability(NetworkCapabilities.NET_CAPABILITY_INTERNET)
        .build()
    connectivityManager.registerNetworkCallback(request, networkCallback)
}

fun stopMonitoring() {
    connectivityManager.unregisterNetworkCallback(networkCallback)
}

private fun checkAndReport() {
    val network = connectivityManager.activeNetwork ?: return
    val caps = connectivityManager.getNetworkCapabilities(network) ?: return
    val downBandwidth = caps.linkDownstreamBandwidthKbps
    val quality = when {
        downBandwidth >= 10_000 -> Quality.EXCELLENT
        downBandwidth >= 1_000  -> Quality.GOOD
        else -> Quality.POOR
    }
    onQualityChanged?.invoke(quality)
}
}

```

本文档版权归深圳自然写科技有限公司所有，技术细节仅用于软件著作权登记鉴别，请勿用于其他商业用途。