Cedar Point System · 本次对话总结与下一步计划

日期：2026-01-23（America/Toronto）

一、对话目标与本次进展

本次对话聚焦于：把“学生积分维护系统（Part 1）”的前端 UI 与 Google Sheets 数据源真正连通，实现“下载（读）+ 上传（写）”的闭环；并完成 GitHub 仓库连接。

- 确认同一个 Apps Script Web App URL 可以同时提供 GET（读）与 POST（写）接口；用 action 参数做路由分发。
- 修复“只能下载不能上传”的问题：通过调整请求方式与请求头，避免浏览器 CORS 预检导致的 Failed to fetch。
- 完善前端提交流程：事件在本地状态（events/students）中先即时生效，再写回表；失败时重新同步回到“表内真相”。
- 完成本地项目与 GitHub 远程仓库的链接与推送流程（git init / remote / push）。

二、核心概念澄清（用工程语言描述你已经做出来的东西）

1）系统三层（Web 常见架构视角）

你的实现可以用标准术语概括为三层：Storage layer（Google Sheets 持久化数据）、API layer（Apps Script Web App 暴露 HTTP 接口）、UI layer（main.html 在浏览器内渲染与交互）。

2）与你笔记中的三层命名的对应关系

你笔记中的分层	常见工程术语	在本项目中的具体对应
data layer	Storage layer	Google Sheets：students/events 两个表（持久化数据）
render layer	UI / Presentation layer	main.html：DOM 渲染、交互、提示、筛选/搜索等
structure layer	API contract + Application logic（协议/并层）	action + 字段契约 + 前端本地状态与业务组织（MVP 阶段常

3）关键机制：Apps Script 与 async/await

Apps Script Web App 没有 main()；当浏览器访问部署后的 /exec URL 时，运行环境会自动调用 doGet(e)（GET 请求）或 doPost(e)（POST 请求）。doGet/doPost 再按 action 分发到具体读写函数。前端使用 async/await 是为了在等待网络返回（fetch Promise）时不阻塞 UI；代码写起来像同步，运行时仍保持页面可响应。

4）CORS 预检与 Content-Type 的工程取舍

当跨域 POST 使用 application/json 等“非简单请求”配置时，浏览器可能先发 OPTIONS 预检。Apps Script 对预检支持有限，容易出现 Failed to fetch。因此在 MVP 阶段选择用 Content-Type: text/plain;charset=utf-8 发送 JSON 字符串，在服务端再 JSON.parse(e.postData.contents)，以绕开预检、提升稳定性。

三、你已经“做出来”的工程点（可写进 developing log 的词汇）

- Layered architecture：UI / API / Storage 分层，降低耦合。
- Action-based routing：单一 endpoint（/exec）+ action 参数分发不同业务。
- Contract（schema）thinking：前后端约定 event/student 字段，减少隐式假设。
- Local state cache：前端维护 students/events 数组作为运行态缓存。
- Optimistic UI update + resync on failure：先让 UI 生效，再写回表；失败则重新同步回到真相源。
- Event log / traceability：events 作为流水记录，points 为可重算的派生状态（便于审计与纠错）。
- CORS preflight avoidance：为稳定性选择 text/plain 作为折中实现。

四、下一步计划（以 deadline 为导向：先完成，再优化）

目标：在不提前“过度工程化”的前提下，按同样的思路完成“物资管理系统（Part 2）”，并最终实现双系统联动（积分　　物资）。建议继续沿用“先 UI → 再数据管理 → 最后联动”的节奏。

阶段 1：物资管理系统 UI（先跑通工作流）

- 定义 MVP 工作流：物资列表 → 物资入库/出库 → 学生兑换/发放 → 查询与筛选。
- UI 页面组件：物资表格（Item list）、物资详情/编辑、交易流水（transactions）、快速操作按钮（+入库/-出库/兑换）。
- 确定最小字段：item_id、name、category、unit、stock（显示值）、note；transaction_id、time、item_id、qty_delta、reason、operator、student_id（可选）。
- 本地状态设计：items[]、transactions[]；优先保证渲染与交互完整。

阶段 2：物资系统数据管理（复用你已掌握的接口模式）

- Storage：Google Sheets 新增 tabs，例如 items / transactions。
- API（Apps Script）：复用同一套 doGet/doPost + action 路由；GET items/transactions；POST appendTransaction / updateItem（或仅用 appendTransaction 并由 transactions 推导库存）。
- 一致性策略（MVP）：transactions 作为真相源；stock 为派生显示（可重算/可校验）。
- 最小测试用例：新增一条入库/出库记录后，刷新页面仍能正确显示库存与流水。

阶段 3：双系统联动（积分系统　　物资系统）

- 联动触发点：当发生“兑换/发放”时，同时产生两条记录：积分 events（扣分）+ 物资 transactions（出库）。
- 接口层实现方式：前端一次操作执行两次 POST；如果其中一个失败，记录 pending 状态并提示重试，最后以重新同步校验一致性。

- ID 对齐：student_id 在两系统通用；item_id 为物资系统主键；在兑换事件中把 item_id 写入 description 或专用字段，便于追溯。
- 验收标准：一次兑换操作能在两张流水表中同时出现，且刷新后仍一致；异常时能检测并恢复。

附：建议的“先完成再优化”优先级

- 先完成功能闭环与数据可追溯（能用、可查、可重算）。
- 再做易用性优化（搜索/筛选/快捷按钮/默认值/输入校验）。
- 最后再考虑结构优化（抽模块、减少重复、提升性能、并发处理）。