记账本系统设计与建模实验报告

一、软件的主要功能

记账本系统由网页版前端和一个 proxy 后端构成。其主要功能包括：

- 用户注册 / 登录；
- 添加、查询、排序、删除、清空交易记录；
- 实现前后端通信的网络模块
- 页面动态背景与点击特效（由单独脚本实现）；
- 简单的日志面板与请求/响应查看方便调试。

技术栈

- 前端交互逻辑： `resource/scripts/app.js` ；
- 前端动画/特效： `resource/scripts/dynamic-effects.js` ；
- 前端样式： `resource/styles/main.css` 与 `resource/styles/dynamic-bg.css` ；
- 前端静态页面： `resource/index.html`
- 后端代理： `proxy` 目录（把“静态前端资源服务”和“HTTP→TCP 转发桥”统一暴露在一个对外端口上，方便 ngrok 免费版单端口场景）。
- 后端（服务器）： `src` 目录（根据前端请求对数据库进行操作并响应）

二、按模块说明（对应文件与职责）

1. 前端业务逻辑模块（2110行） — `resource/scripts/app.js` （简要说明）

- 用 Part1..Part10 做语义性的模块划分：
  - Part1 全局状态与常量（ `state` 、缓存与按钮池等）；
  - Part2 DOM 缓存与基础工具（元素缓存、按钮复用、表格回收、平滑删除 `smoothRemoveRow` ）；
  - Part3 网络端点与请求控制（构建 endpoint、AbortController、并发锁 `runWithRequestLock` 、 `sendPayload` ）；
  - Part4 应用生命周期与初始化（DOMContentLoaded、initializeApplication、UI 启动）；
  - Part5 标签页与表单交互（表单防抖/排队、提交流程、按钮忙态）；
  - Part6 输入校验与请求构造（金额/日期校验、 `composePayload` ）；
  - Part7 响应解析与执行操作（ `parseResponse` 、 `handlePostAction` ）；
  - Part8 查询与记录界面（ `renderRecords` 、 `buildRecordRow` 、注册删除事件等）；
  - Part9 数据格式化与排序（ `formatAmount` 、 `sortEntries` 、统计计算）；
  - Part10 日志与消息（ `logMessage` 、各类操作结果日志）。
- 主要实现细节：
  - 渲染采用 DocumentFragment 批量构建行，再一次性 append 到 tbody 以减少重排；
  - 删除采用动画（添加 `.removing` 类并监听 `transitionend` ）或超时回退，最终调用 `removeChild` 从 DOM 中移除元素；
  - 按钮复用使用 `BUTTON_POOL` 存放已重置的按钮以减少频繁创建 DOM 节点开销；
  - 前端为事件驱动更新：表单提交 → sendPayload → parseResponse → handlePostAction → render / log。

2. 前端动态视觉模块（719行） — `resource/scripts/dynamic-effects.js` （简要说明）

- 职责：canvas 粒子/光效、指针交互（move/click/leave）、渲染循环（requestAnimationFrame）、生命周期管理（init/start/stop/cleanup）。
- 与主业务逻辑解耦，仅影响视觉层

3. 前端样式模块（1126行） — `resource/styles/main.css` 、 `resource/styles/dynamic-bg.css` （简要说明）

- main.css：页面整体布局、组件样式、状态类（ `.is-busy` 、 `.removing` 、 `.active` 、 `.placeholder` 等）。
- dynamic-bg.css：背景容器、粒子/光效样式、transition 与 transform 规则，配合 `dynamic-effects.js` 使用。

4. 后端代理（285行） — `proxy` 目录（反向代理与路由、http→tcp桥）

- 前后端通信的桥梁。（把“静态前端资源服务”和“HTTP→TCP 转发桥”统一暴露在一个对外端口上，方便 ngrok 免费版单端口场景）。

5. 后端（服务器685行）— `src/main/java` （主要说明）

- 技术栈：Java 17（Maven 构建），内置数据库使用 H2 (`com.h2database:h2:2.2.224`)。
- 进程与端口：
  - Java TCP 服务： `communication.ReceiveService` 监听 TCP `8080` ，按“逐行文本协议”处理请求并回写一行响应；
  - HTTP→TCP 桥： `proxy/http_to_tcp_proxy.js` 默认监听 `8081` ，将 HTTP POST 的请求体转为一行文本转发到 TCP `8080` ；
  - 静态资源服务： `scripts/serve_static.ps1` 启动的静态站点默认 `8082` ；
  - 反向代理： `proxy/reverse_proxy.js` 将静态与 API 汇聚到一个端口（脚本示例中为 `8083` ，便于通过 ngrok 暴露一个端口）。
- 包与职责划分：
  - `acounting_system.Main` ： 入口。初始化数据库（ `sqloperation.initialize()` ），创建 `ReceiveService(8080, handler)` 并启动；
  - `communication.ReceiveService` ： 轻量 TCP 行协议服务器。为每个连接用固定线程池处理：逐行读取 → 调用业务 `handler(socket, line)` → 将返回字符串写回并在对端关闭时结束；
  - `RequestManagement.parser` ： 协议解析与路由。将一行请求 `username,action,...` 解析后分派到具体处理： `add / register / login / search / list / clear / delete` ；
  - `RequestManagement.sqloperation` ： 数据库访问层（H2 文件库，路径为工作目录下 `accounting_db` ）。提供用户与账目 CRUD（预编译语句、基础索引）；
  - `RequestManagement.model` ： 请求 DTO（ `addrequest/loginrequest/...` ）；
  - `ResultManagement.ParseResult` ： 后端统一响应对象，序列化为 `action~success~message~entries` 的一行字符串；
  - `sharedmodel.Entry` ： 账目记录实体， `toString()` 序列化为 `id,username,amount,type,date,subject,note` 。
- 文本协议（与前端/代理的约定）：
  - 请求格式：单行 UTF-8 文本，以“英文逗号”分隔： `username,action,[更多参数...]` ，末尾以 `\n` 结束；
  - 支持操作与参数：
    - `register` ： `username,register,password`
    - `login` ： `username,login,password`
    - `add` ： `username,add,amount,date[,type,subject,note]` （缺省 `type=expense` ，其余可空）
    - `search` ： `username,search,startDate,endDate,type,minAmount,maxAmount` （任意可空，后端做动态 SQL 过滤）
    - `list` ： `username,list`
    - `delete` ： `username,delete,entryId`
    - `clear` ： `username,clear`
  - 响应格式： `action~success~message~entries` （单行）。其中 `success` 为 `1/0/null` ； `message` 可能为 `null` ； `entries` 为若干条 `Entry.toString()` 结果用 `|` 连接，若无则 `null` 。为避免分隔冲突， `message` 中的 `~` 会被转义为 `\~` 。
- 数据存储与表结构（H2）：
  - `users(username PRIMARY KEY, password NOT NULL)` ；
  - `entries(id BIGINT AUTO_INCREMENT PRIMARY KEY, username, amount DOUBLE NOT NULL, type VARCHAR(32), date, subject, note)` ；
  - 初始化阶段创建必要索引： `entries(username/date/type)` ；缺失列按需 `ALTER TABLE ... ADD COLUMN IF NOT EXISTS` 以兼容旧数据；
  - 连接管理： `ThreadLocal<Connection>` 做每线程复用，操作结束后关闭并清理；
  - 查询实现： `search` 构建动态 SQL（条件存在才追加 WHERE 子句），所有写操作使用 `PreparedStatement` 防注入； `type` 统一归一化为 `income|expense` ，默认 `expense` 。
- 并发与健壮性：

    - TCP 层：每连接设置 `SoTimeout=30s`；使用固定大小线程池（CPU 核心数的上限），避免创建过多线程；异常以连接粒度捕获并保障关闭；
    - SQL 层：每个操作获取/关闭一次线程绑定连接，避免连接泄漏；
    - 失败场景：`parser` 捕获 `SQLException/Exception`，返回 `success=0` 与相应错误信息；`login` 对“用户不存在/密码错误”分别返回明确提示。
- 与代理协同：
    - 浏览器→反向代理(`8083`)：`/api/*` 转发到 HTTP→TCP 桥(`8081`)，其余请求转静态站点(`8082`)，并对 `/static/*` 做路径改写；
    - HTTP→TCP 桥：接收 POST 文本，自动追加换行并转发到 TCP 后端(`8080`)，收集后端关闭连接前的输出作为 HTTP 响应返回；对超时、上游错误返回 5xx JSON（含 CORS 头）。

6. 运行脚本以及命令行工具（略，详见项目根目录下的README_proxy_setup.md）

三、根据 UML 图实现代码 （本项目代码为5000行,UML图庞大，不便赘述，请参照第二章节）

四、使用大模型辅助开发的心得

在项目开发过程中，大模型帮助我开拓和梳理思路。借助大模型，我了解网络代理的相关知识，实现了网页版的记账本系统。大模型设置了前端样式的具体参数（颜色种类等），省去了繁琐枯燥的参数调试环节。