Web 程序设计 2025 实践课程报告

轻量级 Web 应用程序 – 在线记账本

姓名 陈铜

学号 102201418

授课教师 陈捷

2025 年 6 月 19 日

目录

1.项目基本信息

- **项目名称：**嘿嘿记账本
- **选题方向：**在线记账本
- **作者信息：**102201418　陈铜
- **项目访问方式：** https://github.com/icter99/onlineAccount.git

  克隆仓库后，本地运行下面指令安装依赖，并启动开发服务器

```
npm install
npm run dev
```

- **项目简介：**

本项目是一个基于 Vue 3 开发的轻量级在线记账应用，提供个人收支记录管理、数据可视化分析等功能。主要特点包括：

- 支持收入和支出的记录管理，包括增删查改等基本功能
- 提供多维度数据筛选和统计，查找数据便捷
- 实现数据可视化展示，收支情况直观
- 支持数据导入导出功能

- **技术选型：**

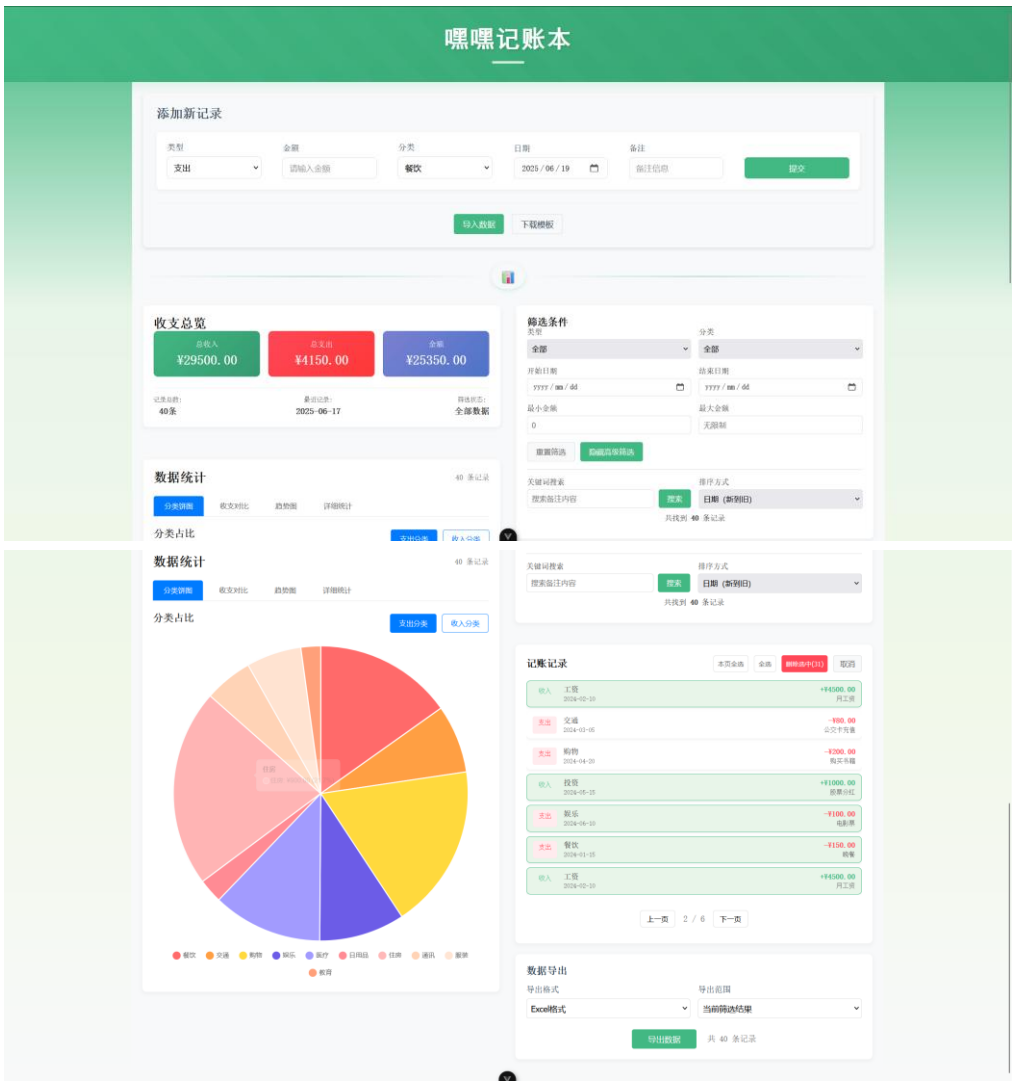
1. 前端框架：Vue 3
   - 选择理由：组件化开发、响应式数据管理、更好的代码组织
   - 使用 Composition API 提升代码复用性和可维护性
2. 数据持久化：LocalStorage
   - 选择理由：无需后端支持，适合轻量级应用

3.状态管理：Pinia

- 选择理由：集中式状态管理，与 localStorage 结合实现数据持久化，页面刷新后数据不会丢失，自动保存数据变更。所有记账数据统一存储在 Pinia store 中，避免组件间直接传递数据，方便数据共享和同步；响应式数据，数据变化自动触发视图更新，组件可以实时获取最新数据，无需手动处理数据同步。

4. 数据可视化：Chart.js
   - 选择理由：轻量级、易用性高、支持多种图表类型
   - 实现收支分类饼图、月度对比柱状图、趋势线图等
5. 其他技术：
   - XLSX.js：处理 Excel 文件导入导出
   - FileSaver.js：文件下载功能

2.页面展示



3.核心功能实现

3.1 数据管理

数据记录的 JSON 格式

```
const record = {
  id: String,          // 唯一标识
  type: String,        // 类型: '收入' | '支出'
  amount: Number,      // 金额
  category: String,    // 分类
  date: String,        // 日期 (YYYY-MM-DD)
  note: String         // 备注
}
```

使用 Pinia 实现集中式状态管理，实现记录的增删改查操作

```
import {
  defineStore
} from 'pinia';
import {
  ref,
  watch
} from 'vue';

export const useAccountStore = defineStore('account', () => {
  // 初始化数据（从 LocalStorage 读取或设为空数组）
  const loadRecords = () => {
    const saved = localStorage.getItem('accountRecords');
    return saved ? JSON.parse(saved) : [];
  };

  // 定义响应式数据
  const records = ref(loadRecords());

  // 监听数据变化并同步到 LocalStorage
  watch(
    records,
    (newVal) => {
      localStorage.setItem('accountRecords', JSON.stringify(newVal));
    }, {
      deep: true
    } // 深度监听对象/数组内部变化
  );

  // 添加记录的方法
  const addRecord = (record) => {
    // 确保每条记录都有唯一的 ID
    const newRecord = {
      id: Date.now() + Math.random().toString(36).substr(2, 9), // 使用时间戳加随机字符串生成唯一 ID
      type: record.type || '支出',
      amount: Number(record.amount) || 0,
      category: record.category || '其他',
      date: record.date || new Date().toISOString().split('T')[0], // 默认当天
      note: record.note || '',
    };
    records.value.push(newRecord);
  };

  // 更新记录的方法
  const updateRecord = (updatedRecord) => {
    const index = records.value.findIndex(record => record.id === updatedRecord.id);
    if (index !== -1) {
      records.value[index] = {
```

```
48.         ...updatedRecord,
49.         amount: Number(updatedRecord.amount) || 0,
50.       };
51.     }
52.   };
53.
54.   // 删除记录的方法
55.   const deleteRecord = (id) => {
56.     records.value = records.value.filter((item) => item.id !== id);
57.   };
58.
59.   return {
60.     records,
61.     addRecord,
62.     updateRecord,
63.     deleteRecord
64.   };
65. });
```

3.2 条件筛选

多条件筛选实现：筛选系统采用链式过滤的设计模式，支持以下筛选条件的任意组合：

- **类型筛选**：收入/支出/全部
- **分类筛选**：17 种预设分类选择
- **日期范围**：开始日期到结束日期
- **金额范围**：最小金额到最大金额
- **关键词搜索**：在备注和分类中模糊搜索
- **排序方式**：按日期或金额的升序/降序

条件筛选流程

组件初始化 → 设置默认筛选条件 → 用户交互修改条件：

├── 下拉选择（立即触发）

├── 日期/金额输入（立即触发）

└── 关键词搜索（防抖 500ms）

→ 多条件筛选处理：

├── 复制原始数据

├── 链式应用各种筛选条件

└── 排序处理

→ 发送结果给父组件 → 所有子组件响应式更新

- **防抖搜索**：关键词搜索使用 500ms 防抖，避免频繁触发页面刷新
- **数据监听**：原始数据变化时自动重新应用筛选条件
- **实时响应**：大部分条件变化立即触发筛选

3.3 数据记录批量删除

批量删除流程

进入批量模式 → 选择记录（单选/本页全选/全选） → 确认删除

→ Store 删除机制：

├── forEach 遍历选中 ID 列表

├── 逐条调用 deleteRecord

└── 数组过滤删除记录

→ 清理批量模式状态 → Pinia 响应式系统自动更新所有相关组件

3.4 数据记录编辑

记录编辑流程

点击编辑 → 复制记录数据到编辑状态 → 显示对话框填充当前值 → 用户修改数据

→ Store 更新机制：

├── 通过 ID 查找记录

├── 替换整条记录数据

├── 数据类型转换保证

└── 触发 LocalStorage 同步

→ 关闭对话框 → 通知父组件刷新 → 所有相关组件自动更新

3.5 数据可视化

采用多维度数据展示的设计理念，通过四种图表类型满足不同的分析需求：

- **分类占比分析（饼图）**：支出/收入的分类分布
- **时间趋势对比（柱状图）**：月度收支变化
- **余额趋势跟踪（趋势图）**： 财务状况变化
- **详细数据统计（表格）**： 精确的数据明细

3.5.1 关键实现逻辑

响应式数据处理

```
1.  // 所有图表数据基于 filteredRecords 计算
2.    const expenseCategories = computed(() => {
3.        const categories = {};
4.        props.filteredRecords.filter(r => r.type === '支出')
5.            .forEach(r => {
6.                categories[r.category] = (categories[r.category] || 0) + r.amount;
7.            });
8.        return Object.entries(categories).map(([category, amount]) => ({ category,
amount }));
9.    });
```

3.5.2 图表生命周期管理

- **按需渲染**：只渲染当前激活的图表，提升性能
- **资源清理**：切换图表前销毁旧实例，防止内存泄漏
- **异步处理**：使用 nextTick 确保 DOM 更新完成后操作 Canvas

3.5.3 差异化设计

- **饼图**：支出用暖色调（警示性），收入用冷色调（稳健感）
- **趋势图**：按时间排序计算累计余额，支持筛选状态下的正确计算
- **柱状图**：月度数据聚合，收入支出并列对比

4.挑战与解决方案

4.1 数据导入时的格式处理

xlsx 模板导入的日期格式与 JSON 格式不完全匹配，导致日期存储错误。并且模版导入时，同一批次的所有记录的唯一 ID 相同，影响批量管理。

处理方法：为每一条新的记录计算唯一 ID。对于日期格式转换，如果是数字形式的日期，因为 Excel 内部将日期存储为数字，表示从 1900 年 1 月 1 日开始的天数，但 JavaScript 的 Date 对象是从 1970 年 1 月 1 日（Unix 纪元）开始计算的，所以需要进行匹配对应到相同的日期，转换成我们需要的形式。对于字符，如果 Excel 中日期是文本格式，直接用 new Date()尝试解析，支持多种常见的日期字符串格式（如：2024-01-15、2024/01/15 等）。

日期格式处理

```
1. // 处理 Excel 数字格式日期
2. if (typeof row.date === 'number') {
3.     // Excel 日期转换：减去 Unix 纪元差值 25569 天，25569 是 1900 年 1 月 1 日到 1970 年 1 月 1 日之间的天数差，86400 * 1000 是一天的毫秒数。
4.     const excelDate = new Date((row.date - 25569) * 86400 * 1000);
5.     dateStr = excelDate.toISOString().split('T')[0];
6. }
7. // 处理字符串格式日期
8. else if (typeof row.date === 'string') {
9.     const date = new Date(row.date);
10.     dateStr = !isNaN(date.getTime()) ?
11.         date.toISOString().split('T')[0] :
12.         new Date().toISOString().split('T')[0]; // 容错处理
13. }
14. // 为每条记录生成唯一 ID
15.     const uniqueId = Date.now() + Math.random().toString(36).substr(2, 9);
```

4.2 页面数据刷新异常

关键词搜索时，输入框中每改变一个字符，页面的数据就会发生一次刷新，频率太高并

且没必要。解决措施，添加防抖机制：

防抖机制

```
// 处理关键词输入，添加防抖机制，输入完毕后
const handleKeywordInput = () => {
    // 清除之前的定时器
    if (keywordTimeout) {
        clearTimeout(keywordTimeout);
    }
    // 设置新的定时器，500ms 后执行筛选
    keywordTimeout = setTimeout(() => {
        applyFilters();
    }, 500);
};
```

5.项目结构

项目主要的文件夹和文件构成及其用途，简要说明如下。

项目主要结构

```
account/
……
├── src/
│   ├── assets/            # 资源文件
│   ……
│   ├── components/         # 组件目录
│   │   ├── AddRecordForm.vue     # 添加记录表单
│   │   ├── RecordList.vue        # 记录列表（含编辑）
│   │   ├── SummaryCard.vue       # 收支总览卡片
│   │   ├── FilterPanel.vue       # 筛选面板
│   │   ├── EnhancedChart.vue     # 增强图表组件
│   │   ├── ExportData.vue        # 数据导出组件
│   │   └── StatisticsChart.vue   # 基础统计图表
│   ……
│   ├── stores/            # Pinia 状态管理
│   │   └── accountStore.js      # 记账数据状态
│   ……
│   ├── App.vue            # 主应用组件
│   └── main.js            # 应用入口
├── package.json            # 项目配置
├── vite.config.js          # Vite 配置
├── jsconfig.json           # JS 配置
└── README.md              # 项目说明
```

6.学习总结与自我评估

- **主要学习收获：**

通过本项目的开发实践，我深入掌握了 Vue 3 的 Composition API 和组件化开发模式，熟练运用组件间通信机制和 Pinia 状态管理，实现了数据的响应式更新和持久化存储。在数据可视化方面，我学会了 Chart.js 的使用方法和性能优化技巧，理解了不同图表类型的适用场景和设计原则，能够将复杂的财务数据转化为直观的可视化展示。同时，项目实践大幅提升了我的前端工程化能力，从项目结构组织到代码模块化设计，从组件复用到功能解耦，都体现了现代前端开发的规范化思维。此外，在用户体验设计方面，我注重交互细节优化，实现了响应式布局适配和界面美化，通过合理的信息架构和视觉设计，提升了应用的易用性和美观性，培养了以用户为中心的设计意识。

- **项目评估：**

完成度：85%以上
- 基础功能完整实现
- 界面设计美观实用
- 数据管理功能完善
- 可视化展示丰富

- **未来改进方向：**

- 添加用户认证系统
- 实现数据云同步
- 增加预算管理功能
- 优化移动端适配
- 添加数据备份功能

- **额外说明：**

本项目为个人独立完成，使用了以下第三方库：
- Vue 3：前端框架
- Pinia：状态管理
- Chart.js：数据可视化
- XLSX.js：Excel 处理
- FileSaver.js：文件下载

所有代码均为原创，使用工具 cusor 帮助修改和优化代码，参考了官方文档和社区最佳实践。