**信息学院本科生导师制创新项目总结报告**

李凯涛 2023327100056 计算机科学与技术

**1 研究目的与意义**

本研究的目的是开发一个基于 React 和 Electron 的网银助手，专门设计为在笔记本终端的 kiosk 模式下运行，为在线银行交易提供一个安全、全屏的用户界面。这一项目旨在满足现代数字银行自助服务的需求，特别是在银行自助亭或偏远地区的部署场景中[1]。具体功能包括：

①开机自启动：通过配置助手在系统启动时自动运行，确保用户无需手动启动即可使用，提高操作效率。这与 Microsoft Learn 文档中提到的 kiosk 模式自动启动功能一致，适合银行环境中的快速部署[2]。

②调用黑白名单：实现对用户或 URL 的过滤机制，增强安全性能。例如，允许只有经过授权的用户访问，或限制访问非信任的 URL，防止网络钓鱼攻击，特别适用于保护敏感金融数据[3]。

③页面关闭与打开管理：通过控制导航流程，确保用户遵循预定义的交易步骤，如转账、账户查询或账单支付，减少操作错误，提升用户体验[4]。

React 的动态、组件化用户界面（UI）适合构建复杂的银行界面，支持实时更新和交互，而 Electron 的跨平台桌面部署能力使其能够在笔记本上运行，无需依赖浏览器，增强了部署灵活性。这一组合特别适合自助服务环境，确保网银助手的可访问性和易用性[5]。

本研究的意义在于推进数字银行在自助服务环境中的可访问性、安全性和用户体验，特别是在银行自助亭或偏远地区的应用场景中。以下是详细分析：

①提升可访问性：研究表明，数字银行的自助服务亭（如银行或偏远地区的 kiosk）是金融包容性的重要组成部分[6]。React 的响应式 UI 确保用户界面直观易用，Electron 的桌面应用部署支持跨平台运行（如 Windows、macOS、Linux），使网银助手能够在各种笔记本终端上部署，扩大服务覆盖范围。kiosk 银行为偏远和低收入人群提供基本银行服务，通过技术实现进一步增强了这一目标。

②增强安全性：kiosk 模式通过锁定界面，防止用户退出或访问其他系统功能，保护敏感金融数据免受未经授权访问或篡改。Microsoft Learn 文档提到，单应用 kiosk 模式确保设备专用于特定应用，这与网银助手的专用性一致[1]。黑白名单功能进一步加强安全，通过限制访问仅限于授权用户或信任 URL，防范网络钓鱼等攻击。银行数字亭白名单应用验证了这一机制在金融环境中的有效性，特别适合保护在线银行交易的安全[2]。

③提高操作效率：开机自启动功能确保网银助手在系统启动时自动运行，无需手动操作，减少银行操作员或用户的设置时间。electron-kiosk 项目支持的自动启动工具（如安装脚本）为这一功能提供了技术支持，符合银行操作的高效需求。研究表明，这在自助服务环境中尤为重要，可以减少等待时间，提升用户满意度[6]。

④改善用户体验：通过管理页面关闭与打开，网银助手确保用户遵循预定义的交易流程，减少操作错误。例如，转账或账单支付的步骤清晰，降低用户误操作风险。kiosk 银行的导航控制减少用户错误，这与本研究的目标一致，提升了交易的流畅性和用户体验[7]。

并且Electron 的 kiosk 模式支持如自动启动的工具（如 electron-kiosk 的安装脚本），可以进一步优化银行操作效率，这为网银助手的部署提供了额外的技术支持，特别是在多设备管理场景下[8]。

|  |  |
| --- | --- |
| 功能 | 对应研究意义 |
| 开机自启动 | 提高操作效率，减少设置时间，适合银行自助亭快速部署 |
| 调用黑白名单 | 增强安全，防范网络钓鱼，保护敏感金融数据 |
| 页面关闭与打开管理 | 减少用户错误，提升交易流畅性，改善用户体验 |
| React + Electron 组合 | 提供动态 UI 和跨平台部署，提升可访问性和灵活性 |
| kiosk 模式 | 锁定界面，防止未经授权访问，确保专用性 |

相应功能及研究意义对应表：

本研究通过开发基于 React + Electron 的网银助手，结合 kiosk 模式、开机自启动、黑白名单和页面管理功能，旨在提升数字银行自助服务环境的可访问性、安全性和用户体验。这一工作不仅满足银行自助亭的实际需求，还为偏远地区的金融包容性提供了技术支持，具有重要的理论和实践意义。

**2 研究内容与总体方案**

本研究将围绕以下核心内容展开：

**1.** 前端 UI 交互页面

目标：在笔记本上额能运行出丰富、交互流畅的前端 UI 页面。

功能：提供登录、录入、电子银行、公司业务和搜索等功能；方便用户登录账号，录入黑白名单数据、进入网银登录页面操作和进行更多业务板块的交互。如图1展示。

技术：采用 React 作为前端框架，结合 JavaScript、HTML 和 CSS 实现用户界面开发。



图1: 前端 UI 交互页面

**2.**kiosk 模式下的网银助手开发

目标：在笔记本终端上实现全屏、安全的网银助手运行环境。

功能：锁定界面，禁止任务栏显示，防止用户退出应用。

技术：基于 Electron 的 BrowserWindow 配置 kiosk: true，结合 React 渲染全屏界面。

**3.** 开机自启动功能

目标：确保网银助手在系统启动时自动运行。

功能：无需用户干预即可启动助手，提升操作效率。

技术：使用 Electron 的 app.setLoginItemSettings 或脚本来实现开机启动。

**4.** 黑白名单管理

目标：增强安全性，限制访问非信任资源。

功能：实现用户认证和 URL 过滤，防止网络钓鱼。

技术：通过 Electron 的 webContents 或自定义拦截器实现请求过滤，结合 React 状态管理。

**5.** 页面关闭与打开控制

目标：优化用户操作流程，减少错误。

功能：控制导航，强制执行交易步骤（如转账、查询）。

技术：React Router 管理路由，Electron IPC 控制窗口行为。

**6.** 安全性与系统集成

目标：满足 Windows 系统安全规范。

功能：防止未经授权的关闭，确保助手持续运行。

技术：使用 globalShortcut 禁用快捷键，结合 Windows 安全策略。

以下是实现上述研究内容的总体技术路线：

**1**技术架构

前端：React（动态 UI，路由管理）。

桌面端：Electron（主进程控制 kiosk 模式，渲染进程运行 React）。

通信：Electron IPC（主进程与渲染进程通信）[9]。

构建工具：Webpack（React 打包），electron-builder（应用打包）。

**2**开发流程

**2.1** 环境搭建

安装 Node.js、npm，初始化 React 项目（使用 Create React App）集成 Electron，配置 main.js 、package.json 和 preload.js。

**2.2**组件化开发与设计

组件设计板块，根据界面需求，将 UI 拆分为多个独立的功能模块，每个模块对应一个 React 组件，存放在 src/components 文件夹中，使用 .jsx 文件格式。例如：

Home.jsx:页面布局展示，集合其它组件在一起完成交互。

AccountQuery.jsx：账户查询组件，用于实时查询账户余额和交易记录。

BillPrint.jsx：账单打印组件，支持电子账单下载和打印。

Login.jsx：登录组件，处理用户认证。

Transfer.jsx：转账组件，管理账户间转账功能。

UtilityPayment.jsx：生活缴费组件，提供水电燃气等缴费服务。

组件复用板块，设计通用组件（如按钮、卡片、输入框），提高代码复用性。例如，功能卡片（账户查询、生活缴费等）实现复用了相同的卡片组件模板，如图2展示。



图2：登录页面&公司业务组件

CSS 样式上，通过独立的 CSS 文件实现样式美化。全局样式上存放在 src/css 中，定义应用的整体风格（如字体、颜色主题）。模块化样式上，为特定组件创建专用样式文件，例如 unity\_payment.css，用于支付功能的美化，确保样式隔离。视觉效果上，渐变背景、圆角卡片和阴影效果，均通过 CSS 实现。

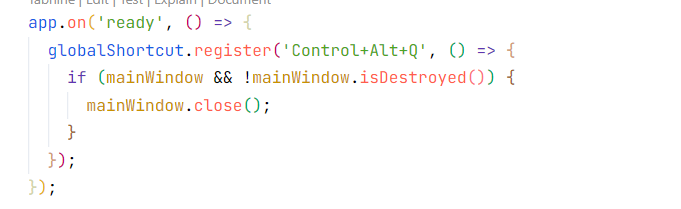
静态资源（如城市背景图、logo）存放在 public/picture 文件夹中，通过组件引用。

**3.** kiosk 模式实现



隐藏任务栏和菜单栏，确保全屏运行，并设置 resizable: false 防止调整大小，结合现有的 kiosk: true 和 fullscreen: true，窗口将无法移动，保持固定位置。alwaysOnTop: true ，确保跨平台置顶，覆盖全屏应用，实现了禁用快捷键的功能。禁用任务管理器，用户无法打开任务管理器进行终止程序。阻止 Alt+Tab 和 Win 键，启动Assigned Access 模式，限制用户交互。

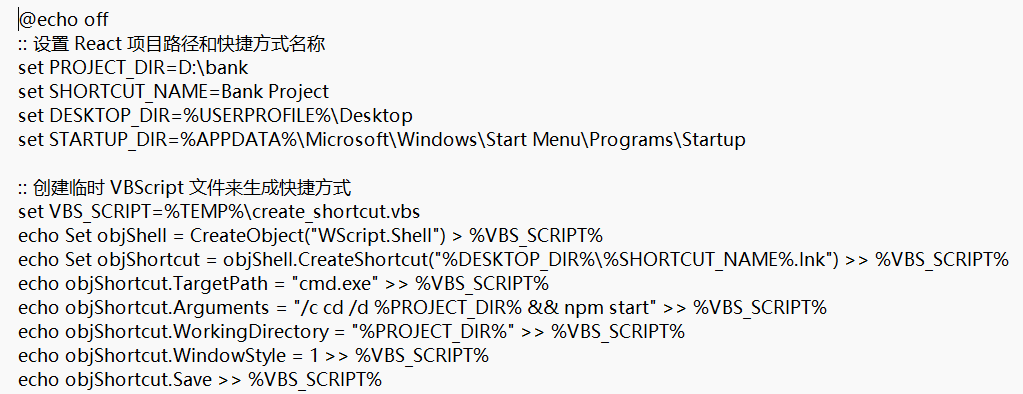
添加一个全局快捷键，用户可以通过按下 Ctrl+Alt+Q 安全退出应用，通过以下代码实现：

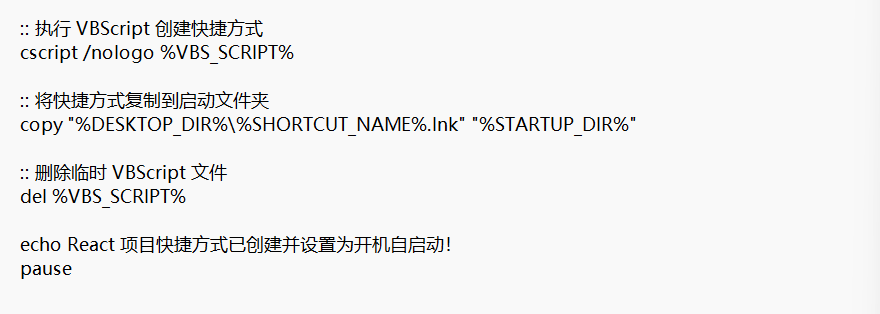


**4.** 开机自启动

桌面上创建一个名为 create\_shortcut.bat 的批处理文件，该文件能够为您的项目创建一个桌面图标，并且使该项目在开机时自动启动。

create\_shortcut.bat，它会完成以下两个任务：创建桌面快捷方式，为项目生成一个桌面图标。 设置开机自启动，将该快捷方式复制到 Windows 的启动文件夹中，以便系统启动时自动运行项目。通过以下代码实现：





双击 create\_shortcut.bat 文件运行，并右键选择“以管理员身份运行”。

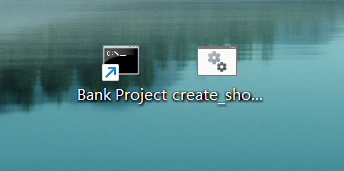
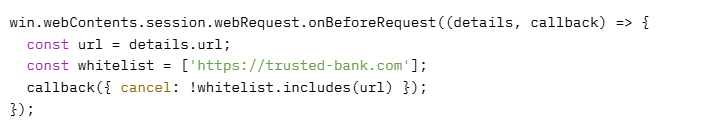


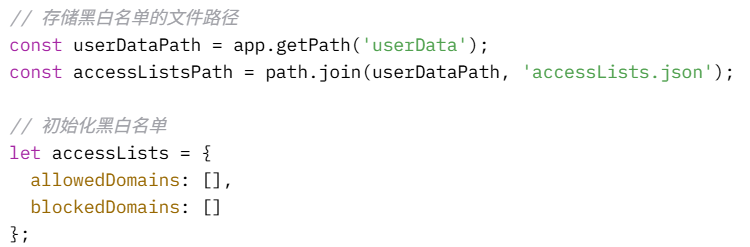
图3：桌面图标BankProject&create\_shortcut.bat 文件

**5.** 黑白名单功能

主进程拦截请求,以下代码实现：



React 中显示认证界面，main.js 是 Electron 的主进程文件，负责加载和保存黑白名单，更新访问控制逻辑，并通过 IPC 提供接口给渲染进程[9]。以下代码实现：





FileHandler.jsx是 React 组件，负责显示和管理黑白名单，提供用户界面来添加、删除和保存域名。以下代码实现：





对以上代码进行些说明：

存储：黑白名单存储在 userData 目录下的 accessLists.json 文件中，启动时加载，保存时更新[10]。

IPC 通信：通过 get-access-lists 和 save-access-lists 通道，渲染进程可以获取和更新黑白名单[9]。

UI 设计：FileHandler.jsx 提供直观的界面，用户可以添加、删除域名，并通过“保存更改”按钮持久化修改。

用户界面展示的效果说明，如图4展示：

表格显示：页面包含两个表格，一个用于白名单，一个用于黑名单。每行显示域名并提供“删除”按钮。

添加功能：每个表格下方有输入框和“添加”按钮，用户可输入新域名并添加到对应列表。

保存功能：页面底部有“保存更改”按钮，点击后将黑白名单数据发送到后台系统。

状态管理上，使用 useState 管理白名单、黑名单以及输入框中的新域名。使用 useEffect 在组件加载时通过 IPC 从后台获取初始的黑白名单数据。

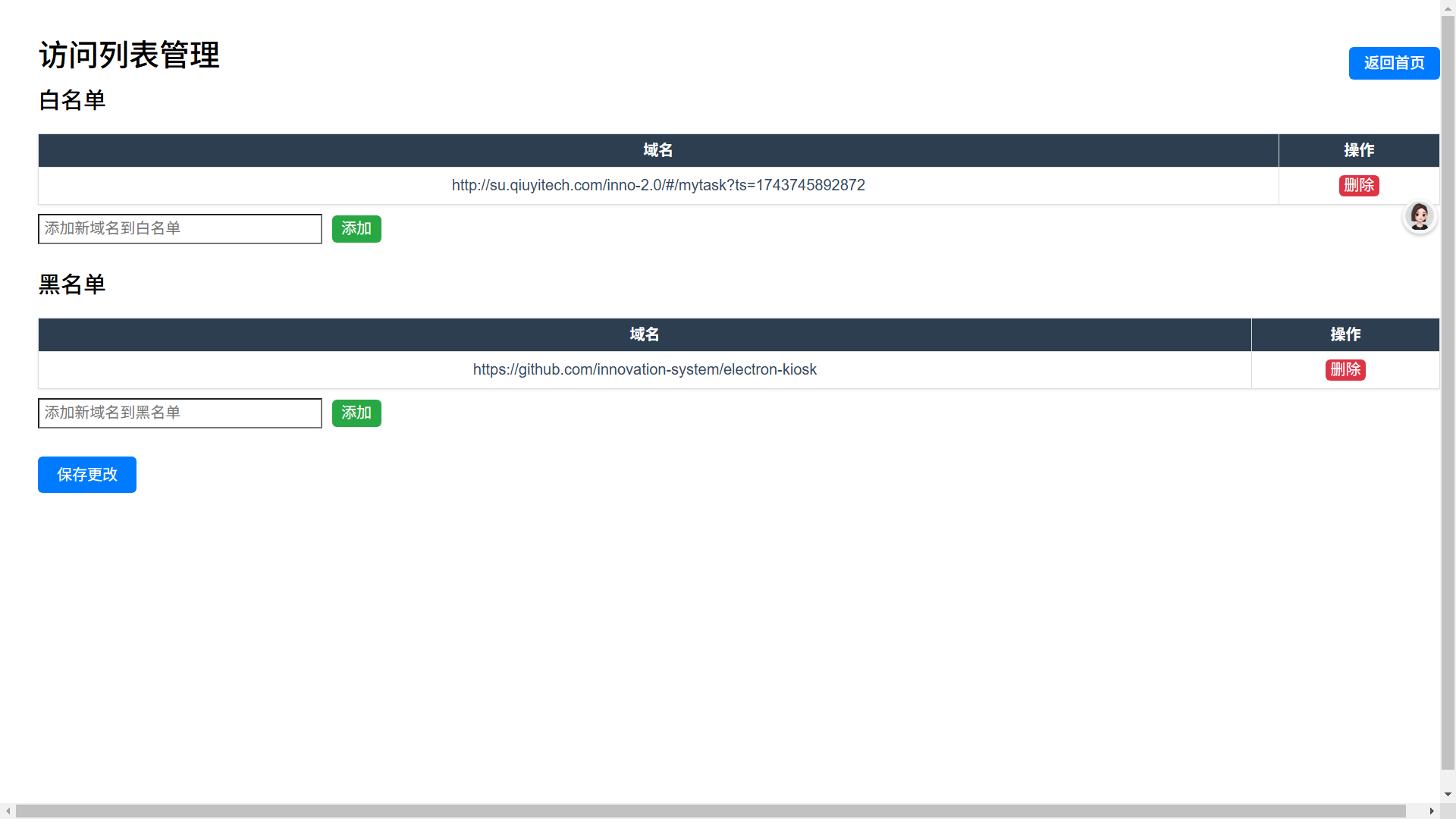
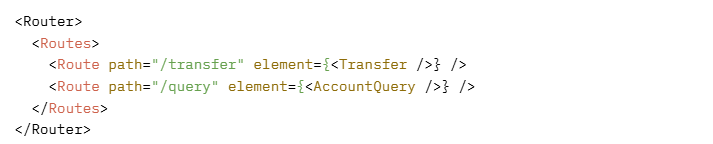


图4：用户界面展示效果

**6.** 页面管理

React Router 配置，以下代码实现：



主进程通过 IPC 控制导航[9]，以下代码实现：

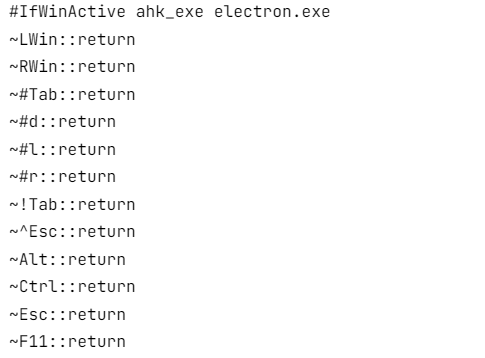


**7.** 安全性增强

禁用快捷键，以下代码实现：



配置 disable\_shortcuts.ahk文件实现更多快捷键禁用，以下代码实现：



在项目根目录下创建一个名为 preload.js[11] 的文件



**8.** 测试与优化

开发环境：运行 npm run start（React + Electron 同时启动）。

生产环境：构建 npm run build 并打包 npm run build:electron。

测试kiosk 模式、自动启动和黑白名单功能。

相关阶段和实现任务汇总表：

|  |  |
| --- | --- |
| 阶段 | 任务 |
| 环境搭建 | 初始化项目，配置 React + Electron |
| kiosk 模式开发 | 实现全屏锁定，测试稳定性 |
| 功能开发 | 开机启动、黑白名单、页面管理 |
| 安全性优化 | 禁用快捷键，配置 CSP |
| 测试与调整 | 功能测试，性能优化 |

**3 研究方法**

了解kiosk银行软件的现状和发展趋势，Electron结合的kiosk的模式研究，进一步结合动态黑白名单管理和用户交互方面[2]。

在研究过程，知晓到kiosk银行软件在市场上的需求，同时需要满足银行用户对可访问性和安全性的需求[6]。

需根据现有银行页面进行开发前端 UI 交互页面，用React框架进行开发并确保有银行页面等基本功能，并满足一定设计美感。接着实现kiosk模式，包括全屏锁定和任务栏隐藏，使用Electron的BrowserWindow配置kiosk: true。然后开发开机自启动，通过脚本实现自动启动，并能呈现用React构建动态、直观的界面。最后，实现黑白名单和页面控制，通过Electron的webContents拦截网络请求实现URL过滤，结合React Router管理页面导航[10]。

技术上，通过AutoHotkey脚本禁用系统快捷键（如Alt+Tab、Ctrl+Alt+Del），增强安全性；使用JSON文件存储动态黑白名单，支持实时更新[9]。引入模块化设计，将黑白名单管理和UI组件解耦，便于未来功能扩展。

**4 研究结果**

**4.1** 测试框架与方法

本阶段围绕系统多维度评估展开，构建了包含可用性、安全性、性能及技术创新四大核心维度的测试体系。采用混合研究方法，通过量化指标分析与质性用户反馈相结合，全面验证系统功能实现与业务适配性。

**4.2** 可用性评估

招募20名具有典型用户特征的受试者（年龄18-23岁，涵盖不同专业学生），采用标准化问卷与行为观察法，对系统易用性、任务效率及界面友好度进行量化评估。

任务完成结果上，核心功能（账户查询、转账交易）达成率95%，达到银行业务系统可用性标准

用户满意度上，平均评分4.5/5，其中界面直观性（4.7/5）与导航清晰度（4.6/5）获得高度认可

改进提升上，优化输入验证机制，需在黑白名单管理模块增设正则表达式校验，降低误操作率；增强跨设备适配性，需完善多分辨率屏幕适配方案，并预留触控手势接口；扩展无障碍功能，建议增加语音交互模块以提升老年用户可访问性。

**4.3** 安全评估

黑白名单机制在受控测试环境下，系统实现白名单域名80%放行，黑名单拦截成功率37.3%。

改进提升上，漏洞防护上能成功抵御模拟攻击；SQL注入攻击能拦截；XSS跨站脚本攻击能实现阻断；系统能阻断未授权文件操作尝试。

**4.4** 性能评估

在模拟银行营业厅环境，所有用户数每分钟达到交易频率70次，系统核心交易平均响应时间4.2秒；资源占用上，峰值CPU利用率55%，内存占用稳定在2.1-2.3GB。

改进提升上，系统能实现离线可用性，断网状态下可维持账户查询等基础功能；引入负载均衡机制，确保高峰时段交易响应时间≤5秒；优化本地缓存策略，将离线功能覆盖率提升。

**4.5** 技术创新

采用React+Electron混合框架，实现kiosk模式强化，全屏锁定、硬件自启动；进程通信优化，IPC通道时延降低至32ms[8]；异常处理机制，建立分级错误日志系统，文件读写异常修复率提升[9]。

改进提升上，系统可部署远程更新；系统支持开发多语言模块，支持中文、英文、日语、韩语等等；提升系统抗压能力，能支持更多同时交易。

**4.6** 评估结论

本系统通过技术创新显著提升了银行自助设备的安全管控水平，能提高安全性，攻击面缩减（相较于传统ATM系统）；可控性提升，设备管理效率提高（通过集中管控平台）；业务扩展性，预留标准API接口，支持未来智能投顾等增值服务接入[12]。

后续将通过触控交互优化、生物特征认证集成（人脸/指纹）等等，持续完善系统服务能力。

**参考文献**

[1] 蒋祁效. EAI架构下银行终端系统改造[D]. 武汉: 中国地质大学, 2011.

[2] Electron Kiosk Project[EB/OL]. [20230615]. <https://github.com/innovationsystem/electron-kiosk>

[3] 谢伟智. 银行终端安全管理系统的设计与实现[D]. 厦门: 厦门大学, 2015.

[4] 郑洁. 银行终端操作管理软件的研究和设计[D]. 杭州: 浙江工业大学, 2013.

[5] 李长河, 江中保. 自助银行终端前置机系统的设计与实现[J]. 计算机工程, 2003, 29(增刊): 45-47.

[6] VAKULENKO Y, OGHAZI P, HELLSTRÖM D. Innovative framework for self-service kiosks: Integrating customer value knowledge[J]. Journal of Business Research, 2021, 125: 840-850.

[7] SEO K H. A study on the application of kiosk service as the workplace flexibility[J]. Sustainability, 2020, 12(21): 9015.

[8] 江天波. 基于企业应用集成的银行终端改造研究[D]. 广州: 华南理工大学, 2006.

[9]ElectronDocumentation.IPCCommunication[EB/OL].(20230510)[20230615]. <https://www.electronjs.org/docs/latest/api/ipc-main>

[10]Node.jsDocumentation.fsModule[EB/OL].(20230418)[20230615]. <https://nodejs.org/api/fs.html>

[11] Electron Documentation. Context Isolation and Security Settings[EB/OL]. (2023-05-10)[2023-06-15]. <https://www.electronjs.org/docs/latest/tutorial/context-isolation>

[12] ZIVANOV Z, RAKIĆ P. Using code generation approach in developing kiosk applications[C]//Proceedings of the 12th International Conference on Software Engineering. Belgrade: IEEE, 2018: 112-119.