Perfect World Account Manager

Bug Fix and Enhancement Tasks for AI Coding Agent

Project Overview

Application: Perfect World Account Manager
Current State: Functional but with several critical issues and UI/UX improvements needed
Target: Stable, user-friendly account management tool with modern interface
Priority: High - Application has core functionality issues affecting user experience

---

🐛 Critical Bug Fixes

Task 1: Settings Persistence in Standalone EXE

Priority: CRITICAL
Issue: When application is compiled as standalone .exe file, settings are not being saved/loaded properly.

Technical Details:

- Settings likely not persisting due to file path issues in compiled environment
- May be trying to write to protected directories or incorrect relative paths
- Need to implement proper data directory detection for executable vs development environment

Requirements:

- [ ] Detect if running from .exe or development environment
- [ ] Use appropriate data storage location (AppData/Documents for .exe)
- [ ] Implement fallback mechanism for settings storage
- [ ] Test settings persistence across application restarts in compiled form
- [ ] Ensure configuration files are created with proper permissions

Expected Outcome: Settings save and load correctly in both development and compiled .exe versions

---

Task 2: Three Dots Menu Functionality

Priority: HIGH
Issue: Context menu (three dots menu) is not responding to user interactions.

Technical Details:

- Menu appears to be rendered but click events not properly handled
- Could be z-index issues, event propagation problems, or missing event handlers
- May need to check if menu is properly positioned and accessible

Requirements:

- [ ] Debug menu click event handlers
- [ ] Verify menu positioning and visibility
- [ ] Check for conflicting event listeners
- [ ] Ensure proper menu item functionality
- [ ] Test menu behavior across different screen resolutions
- [ ] Implement proper menu dismissal on outside clicks

Expected Outcome: Three dots menu fully functional with all menu items working correctly

---

Task 3: Process Launch Delay Management

Priority: HIGH
Issue: When launching multiple game windows simultaneously, process IDs are not being saved correctly due to race conditions.

Technical Details:

- Concurrent launches causing process tracking conflicts
- Need sequential launching with proper delays
- Process ID assignment getting mixed up between instances

Requirements:

- [ ] Implement queue-based launching system
- [ ] Add configurable delay between consecutive launches
- [ ] Ensure each process is fully started before launching next
- [ ] Implement proper process ID tracking and assignment
- [ ] Add launch status feedback to user
- [ ] Handle launch failures gracefully without affecting other launches

Expected Outcome: Multiple account launches work reliably with correct process tracking

---

⚙️ Feature Enhancements

Task 4: Configurable Launch Timing Settings

Priority: MEDIUM
Issue: Users need ability to customize timing between launches for their system performance.

Technical Details:

- Add settings option for launch delay configuration
- Should integrate with Task 3's delay system
- Need intuitive UI for timing configuration

Requirements:

- [ ] Add "Launch Delay" setting in Settings panel
- [ ] Implement slider or input field (range: 1-30 seconds)
- [ ] Set reasonable default value (3-5 seconds)
- [ ] Apply delay setting to launch queue system
- [ ] Add setting description/tooltip for user guidance
- [ ] Validate input ranges and handle edge cases

Expected Outcome: Users can customize launch delays based on their system capabilities

---

Task 5: Game Window Name Update

Priority: LOW
Issue: Application needs to recognize game windows with name "Asgard Perfect World".

Technical Details:

- Update window detection logic to match correct game window title
- May affect process tracking and window management features

Requirements:

- [ ] Update window title detection from current name to "Asgard Perfect World"
- [ ] Test window detection and focusing functionality
- [ ] Update any hardcoded window name references
- [ ] Ensure process-to-window mapping works correctly
- [ ] Verify window management features still function

Expected Outcome: Application correctly identifies and manages "Asgard Perfect World" game windows

---

Task 6: Server Selection Field Enhancement

Priority: MEDIUM
Issue: Need additional data field for server selection to improve account organization.

Technical Details:

- Add new dropdown field to account entities
- Integrate with existing data model and storage
- Update UI to accommodate new field

Requirements:

- [ ] Add "Server" field to account data model
- [ ] Create dropdown component with options: "X" and "Main"
- [ ] Update database/storage schema to include server field
- [ ] Implement data migration for existing accounts
- [ ] Add server field to Add/Edit account dialogs
- [ ] Update account display to show server information
- [ ] Set appropriate default value for new accounts

Expected Outcome: Users can categorize accounts by server (X or Main) for better organization

---

Task 8: Table Header Select/Unselect All Functionality

Priority: MEDIUM
Issue: Users need ability to quickly select or unselect all accounts in the table for bulk operations.

Technical Details:

- Add checkbox in table header row
- Implement tri-state checkbox logic (unchecked, checked, indeterminate)
- Connect to existing row selection system
- Update selection state management

Requirements:

- [ ] Add master checkbox to table header
- [ ] Implement select all functionality when master checkbox is checked
- [ ] Implement unselect all functionality when master checkbox is unchecked
- [ ] Handle indeterminate state when some (but not all) rows are selected
- [ ] Update master checkbox state when individual row selections change
- [ ] Ensure proper styling and visual feedback for all checkbox states
- [ ] Test with large numbers of accounts for performance
- [ ] Update existing bulk operations to work with select all functionality

Technical Implementation:

- [ ] Add checkbox component to table header
- [ ] Implement tri-state logic: false (none selected), true (all selected), null (some selected)
- [ ] Create event handlers for master checkbox state changes
- [ ] Update row selection handlers to sync with master checkbox
- [ ] Ensure accessibility compliance with proper ARIA labels

Expected Outcome: Users can efficiently select/unselect all accounts with a single click, with clear visual indication of selection states

---

Task 9: Import/Export JSON Functionality

Priority: MEDIUM
Issue: Users need ability to backup and transfer account data between installations or share configurations.

Technical Details:

- Implement secure JSON export with optional data encryption
- Create import system with validation and conflict resolution
- Allow selective export of specific accounts
- Ensure data integrity and backward compatibility

Requirements:

Export Functionality:

- [ ] Add "Export" button/menu option in main interface
- [ ] Create export dialog with account selection options
- [ ] Implement "Export All" option for complete data backup
- [ ] Implement "Export Selected" option for partial exports
- [ ] Generate JSON with proper data structure and formatting
- [ ] Include metadata (export date, version, account count)
- [ ] Add option to exclude sensitive data (passwords) from export
- [ ] Implement file save dialog with suggested filename format
- [ ] Add progress indicator for large exports

Import Functionality:

- [ ] Add "Import" button/menu option in main interface
- [ ] Create file selection dialog for JSON import
- [ ] Implement JSON validation and error handling
- [ ] Add import preview showing accounts to be imported
- [ ] Handle duplicate account detection and resolution options
- [ ] Provide merge strategies: skip duplicates, overwrite, or create new
- [ ] Add import progress indicator and status feedback
- [ ] Implement rollback capability in case of import errors
- [ ] Validate data integrity and format compatibility

Data Security:

- [ ] Implement optional password encryption for sensitive exports
- [ ] Add warning messages about data sensitivity
- [ ] Validate JSON structure and required fields
- [ ] Sanitize imported data to prevent injection attacks
- [ ] Add checksums or integrity verification

User Interface:

- [ ] Create intuitive export/import dialogs
- [ ] Add clear instructions and help text
- [ ] Implement proper error messaging and validation feedback
- [ ] Show export/import statistics (accounts processed, errors, etc.)
- [ ] Add confirmation dialogs for potentially destructive operations

JSON Structure Specification:

```json
{
  "metadata": {
    "version": "1.0",
    "exportDate": "2025-06-12T10:30:00Z",
    "accountCount": 5,
    "encrypted": false
  },
  "accounts": [
    {
      "login": "username1",
      "password": "encrypted_or_excluded",
      "characterName": "CharName",
      "description": "Account description",
      "owner": "Player1",
      "server": "Main"
    }
  ]
}
```

Expected Outcome: Users can easily backup, transfer, and share account configurations with flexible export options and safe import with validation

---

🎨 UI/UX Improvements

Task 7: Modern UI Design Overhaul

Priority: HIGH
Issue: Current interface appears dark, blank, and unfriendly. Needs complete design modernization.

Technical Details:

- Current dark theme lacks visual hierarchy and modern styling
- Interface appears empty and uninviting
- Need comprehensive design system implementation

Design Requirements:

Color Scheme:

- [ ] Implement modern, accessible color palette
- [ ] Light theme as default with optional dark theme
- [ ] Proper contrast ratios for accessibility
- [ ] Consistent color usage across components

Layout Improvements:

- [ ] Add proper spacing and padding throughout interface
- [ ] Implement card-based design for account entries
- [ ] Create clear visual hierarchy with typography
- [ ] Add appropriate icons and visual elements
- [ ] Implement responsive layout principles

Component Enhancements:

- [ ] Style buttons with modern hover and active states
- [ ] Improve form inputs with proper styling and validation feedback
- [ ] Add loading states and progress indicators
- [ ] Implement proper dropdown and menu styling
- [ ] Create consistent spacing system

User Experience:

- [ ] Add welcome/empty state when no accounts are loaded
- [ ] Implement proper error and success messaging
- [ ] Add tooltips and help text where appropriate
- [ ] Create intuitive navigation and information architecture
- [ ] Add confirmation dialogs for destructive actions

Visual Elements:

- [ ] Design appropriate application icon
- [ ] Add subtle animations and transitions
- [ ] Implement proper focus states for accessibility
- [ ] Create visual feedback for user actions
- [ ] Add status indicators for account states

Expected Outcome: Modern, user-friendly interface that feels professional and intuitive

---

🧪 Testing Requirements

For Each Task:

- [ ] Unit tests for core functionality
- [ ] Integration tests for cross-component features
- [ ] Manual testing across different Windows versions
- [ ] Performance testing with multiple accounts
- [ ] Accessibility testing for UI improvements
- [ ] User acceptance testing for design changes

Specific Test Cases:

- [ ] Settings persistence across application restarts
- [ ] Multiple account launching under various conditions
- [ ] UI responsiveness across different screen sizes
- [ ] Error handling and edge cases
- [ ] Data migration scenarios
- [ ] Select all/unselect all functionality with various account counts
- [ ] Import/export JSON with different data sizes and formats
- [ ] Import validation and error handling with malformed JSON files
- [ ] Export security and data integrity verification

📋 Implementation Guidelines

Development Standards:

- Follow existing code architecture and patterns
- Maintain backward compatibility where possible
- Implement proper error handling and logging
- Use consistent naming conventions
- Add appropriate code documentation

Technical Considerations:

- **Framework**: Maintain current technology stack
- **Performance**: Ensure changes don't impact application performance
- **Security**: Validate all user inputs and handle sensitive data appropriately
- **Compatibility**: Test on Windows 10/11 environments
- **Packaging**: Verify all changes work in compiled .exe format

Code Quality:

- [ ] Code review and refactoring where needed
- [ ] Remove any deprecated or unused code
- [ ] Optimize database queries and file operations
- [ ] Implement proper exception handling
- [ ] Add logging for debugging and monitoring

🚀 Delivery Expectations

Phase 1: Critical Fixes (Priority: CRITICAL/HIGH)

Tasks: 1, 2, 3, 7

Timeline: Complete critical bugs and basic UI improvements

Phase 2: Feature Enhancements (Priority: MEDIUM)

Tasks: 4, 6, 8, 9

Timeline: Add new features, data management, and configuration options

Phase 3: Polish and Optimization (Priority: LOW)

Tasks: 5, Testing, Documentation

Timeline: Final optimizations and comprehensive testing

Final Deliverables:

- [ ] Updated source code with all fixes implemented
- [ ] Compiled .exe file ready for distribution
- [ ] Updated documentation and user guide
- [ ] Test results and validation reports
- [ ] Migration guide for existing users

📝 Additional Notes

Known Technical Constraints:

- Must maintain compatibility with existing account data
- Cannot break existing user workflows
- Need to preserve current feature set while improving

User Impact Considerations:

- Minimize disruption to existing users
- Provide clear upgrade path
- Maintain familiar functionality while improving experience

Success Criteria:

- All critical bugs resolved
- Significantly improved user experience
- Stable performance with multiple accounts
- Positive user feedback on design improvements
- Successful deployment as standalone executable
- Efficient bulk account management capabilities
- Reliable data backup and transfer functionality

Document Version: 1.0
Created: June 2025
Last Updated: June 2025
Status: Ready for Implementation