Digital Quill Publishing Desktop Application - Windows Project Structure

Overview

This document outlines the project structure for the Windows-specific version of the Digital Quill Publishing desktop application, focusing on the agent interaction system with priority on the Literary Agent module.

Project Structure

```
digital-quill-windows/
├── .vscode/                  # VS Code configuration
│   ├── launch.json           # Debugging configuration
│   └── settings.json         # Editor settings
├── assets/                   # Static assets
│   ├── icons/                # Application icons
│   │   ├── app-icon.ico      # Windows application icon
│   │   └── tray-icon.png     # System tray icon
│   └── avatars/              # Agent avatar images
│       ├── literary-agent.png    # Literary Agent avatar
│       ├── front-desk.png        # Front Desk Assistant avatar
│       └── acquisition-editor.png # Acquisition Editor avatar
├── build/                    # Build configuration
│   ├── installer.nsh         # NSIS installer script
│   └── entitlements.plist    # macOS entitlements (for cross-platform)
├── dist/                     # Compiled output (generated)
├── node_modules/             # Dependencies (generated)
├── release/                  # Packaged applications (generated)
├── src/                      # Source code
│   ├── main/                 # Electron main process
│   │   ├── index.ts          # Main entry point
│   │   ├── ipc-handlers.ts   # IPC communication handlers
│   │   └── window-manager.ts # Window management
│   ├── renderer/             # Electron renderer process
│   │   ├── index.html        # HTML entry point
│   │   ├── index.tsx         # React entry point
│   │   ├── app.tsx           # Main application component
│   │   ├── components/       # UI components
│   │   │   ├── common/       # Shared components
│   │   │   │   ├── avatar.tsx    # Agent avatar component
│   │   │   │   ├── chat.tsx      # Chat interface component
```

```
│   │   │   │   └── terminal.tsx  # Terminal-like interface component
│   │   │   └── agents/          # Agent-specific components
│   │   │       ├── agent-view.tsx # Generic agent view
│   │   │       └── agent-computer.tsx # Agent's computer view
│   │   ├── styles/              # CSS/SCSS styles
│   │   │   ├── main.scss        # Main stylesheet
│   │   │   └── themes/          # Theme stylesheets
│   │   └── utils/               # Utility functions
│   │       ├── ipc.ts           # IPC communication utilities
│   │       └── theme.ts         # Theme management
│   ├── shared/                  # Shared between main and renderer
│   │   ├── types.ts             # TypeScript type definitions
│   │   ├── constants.ts         # Shared constants
│   │   └── utils.ts             # Shared utilities
│   └── agents/                  # Agent modules
│       ├── agent-module-system.ts # Agent module system
│       ├── communication-bus.ts  # Inter-agent communication
│       ├── front-desk/          # Front Desk Assistant module
│       │   ├── index.ts         # Module entry point
│       │   ├── ui/              # UI components
│       │   └── handlers/        # Message handlers
│       ├── literary-agent/      # Literary Agent module (priority)
│       │   ├── index.ts         # Module entry point
│       │   ├── ui/              # UI components
│       │   │   ├── avatar.tsx    # Agent avatar
│       │   │   ├── chat.tsx      # Chat interface
│       │   │   └── computer.tsx  # Agent's computer view
│       │   ├── handlers/        # Message handlers
│       │   │   ├── manuscript-analysis.ts # Manuscript analysis
│       │   │   ├── market-trends.ts      # Market trend analysis
│       │   │   └── query-handling.ts     # User query handling
│       │   ├── services/        # Agent-specific services
│       │   │   ├── market-analysis.ts    # Market analysis service
│       │   │   └── genre-evaluation.ts   # Genre evaluation service
│       │   └── types.ts         # Agent-specific types
│       └── acquisition-editor/   # Acquisition Editor module
│           ├── index.ts         # Module entry point
│           ├── ui/              # UI components
│           └── handlers/        # Message handlers
├── tests/                       # Test files
│   ├── unit/                    # Unit tests
│   │   ├── agent-module-system.test.ts
│   │   └── communication-bus.test.ts
│   └── integration/             # Integration tests
│       └── agent-interaction.test.ts
├── .eslintrc.js                 # ESLint configuration
├── .gitignore                   # Git ignore file
├── .prettierrc                  # Prettier configuration
├── electron-builder.yml         # Electron builder configuration
├── package.json                 # NPM package configuration
```

```
├── tsconfig.json              # TypeScript configuration
├── webpack.main.config.js     # Webpack config for main process
├── webpack.renderer.config.js # Webpack config for renderer process
├── build.bat                  # Windows build script
└── README.md                  # Project documentation
```

Key Components

1. Agent Module System

The agent module system provides the core architecture for pluggable AI agent modules:

- **AgentModule Interface**: Defines the standard interface all agent modules must implement
- **Message Interface**: Standardized format for inter-agent communication
- **AgentState Interface**: Defines the structure of agent state data
- **AgentModuleManager**: Responsible for loading, managing, and communicating with agent modules

2. Communication Bus

The communication bus enables agents to exchange messages and collaborate:

- **Message Routing**: Routes messages between agents
- **Subscription System**: Allows agents to subscribe to specific message types
- **Conversation History**: Maintains history of agent conversations
- **Request-Response Handling**: Manages request-response patterns between agents

3. Literary Agent Module (Priority)

The Literary Agent module evaluates manuscript commercial potential and provides market feedback:

- **Manuscript Analysis**: Analyzes manuscripts for market potential
- **Market Trend Analysis**: Evaluates current market trends in publishing
- **Genre Evaluation**: Assesses manuscript fit within genre expectations
- **Query Handling**: Processes and responds to user queries
- **UI Components**: Custom avatar, chat interface, and computer view

4. Build System

The build system is optimized for Windows:

- **build.bat**: Windows-specific build script with proper error handling
- **electron-builder.yml**: Configuration for creating Windows installers
- **webpack configs**: Separate configurations for main and renderer processes

Windows-Specific Considerations

1. **File Paths**: Use path.join() for cross-platform file path handling
2. **Build Script**: Windows-compatible batch script with proper error handling
3. **Icon Formats**: Windows-specific .ico format for application icons
4. **Installer**: NSIS installer configuration for Windows
5. **Environment Variables**: Windows-compatible environment variable handling
6. **Process Management**: Proper process spawning and management for Windows

Development Workflow

1. **Setup**: Install dependencies with `npm install`
2. **Development**: Run in development mode with `npm run dev`
3. **Building**: Build the application with `build.bat`
4. **Testing**: Run tests with `npm test`
5. **Packaging**: Create installable package with `npm run dist`

This project structure provides a solid foundation for the Windows-specific version of the Digital Quill Publishing desktop application, with a focus on the agent interaction system and prioritizing the Literary Agent module.