Heaven Interface - Development Guide

Version: 0.1.0
Last Updated: October 17, 2025

Overview

Heaven Interface is the minimalist GUI for Solo Git, designed for AI-augmented solo development. This guide covers setup, development, testing, and deployment.

Table of Contents

Prerequisites

Required Software

- **Node.js:** >= 18.0.0
- **npm:** >= 9.0.0
- **Rust:** >= 1.70.0 (for Tauri)
- **Tauri CLI:** >= 1.5.0

System Requirements

- **OS:** macOS 11+, Windows 10+, or Linux (Ubuntu 20.04+)
- **RAM:** 4GB minimum, 8GB recommended
- **Disk:** 2GB free space

Backend Requirement

Heaven GUI connects to the Solo Git backend via Tauri IPC. Ensure the backend is running:

```
# Start Solo Git backend
cd solo-git
python -m evogitctl serve
```

Installation

1. Clone Repository

```
cd /path/to/solo-git
cd heaven-gui
```

2. Install Dependencies

```
npm install
```

This installs:
- React 18.2
- Monaco Editor
- Recharts
- D3.js
- Tauri API bindings

3. Verify Installation

```
npm run type-check
```

Expected output:

```
✓ No TypeScript errors
```

Development Workflow

Start Development Server

```
npm run tauri:dev
```

This launches:
1. Vite dev server (hot reload for React)
2. Tauri window with native frame
3. Backend IPC connection

Note: Backend must be running on `http://localhost:8765` (or configured port).

Development Mode Features

- ✅ Hot Module Replacement (HMR)
- ✅ React Fast Refresh
- ✅ TypeScript type checking
- ✅ Source maps for debugging
- ✅ DevTools enabled

File Watching

Vite watches these directories:
- `src/` - React components
- `src/styles/` - CSS files
- `public/` - Static assets

Changes trigger automatic reload.

Architecture Overview

Tech Stack

```
Heaven GUI
├── Frontend: React 18 + TypeScript
├── Bundler: Vite 5
├── Desktop: Tauri 1.5
├── Editor: Monaco Editor
├── Charts: Recharts
├── Graph: D3.js
└── IPC: Tauri Commands
```

Directory Structure

```
heaven-gui/
├── src/
│   ├── components/          # React components
│   │   ├── AIAssistant.tsx  # AI chat panel
│   │   ├── CodeViewer.tsx   # Monaco editor
│   │   ├── CommandPalette.tsx
│   │   ├── CommitGraph.tsx  # Git timeline
│   │   ├── ErrorBoundary.tsx
│   │   ├── FileBrowser.tsx  # File tree
│   │   ├── KeyboardShortcutsHelp.tsx
│   │   ├── NotificationSystem.tsx
│   │   ├── Settings.tsx
│   │   ├── StatusBar.tsx
│   │   ├── TestDashboard.tsx
│   │   └── WorkpadList.tsx
│   ├── hooks/               # Custom hooks
│   │   └── useKeyboardShortcuts.ts
│   ├── styles/              # CSS files
│   │   ├── App.css
│   │   └── index.css
│   ├── App.tsx              # Main app component
│   └── main.tsx             # React entry point
├── src-tauri/               # Tauri backend
│   ├── src/
│   │   └── main.rs          # Rust IPC handlers
│   ├── tauri.conf.json      # Tauri config
│   └── Cargo.toml
├── public/                  # Static assets
├── index.html               # HTML entry point
├── package.json
├── tsconfig.json
├── vite.config.ts
└── README.md
```

Data Flow

```
┌──────────────┐  ←IPC→  ┌──────────────┐  ←API→  ┌──────────────┐
│  React UI    │         │ Tauri Bridge │         │  Solo Git    │
│  (Frontend)  │         │   (Rust)     │         │  (Backend)   │
└──────────────┘         └──────────────┘         └──────────────┘
      │
      │ Updates
      ▼
┌──────────────┐
│   Monaco     │  Code Editor
│  Recharts    │  Test Charts
│     D3       │  Commit Graph
└──────────────┘
```

Component Guide

Core Components

1. App.tsx - Main Application

Purpose: Root component, manages global state and keyboard shortcuts

Key Features:
- Global state management (repo, workpad, costs)
- View mode control (idle, navigation, planning, coding, commit)
- Keyboard shortcut orchestration
- Modal state management

State Variables:

```
const [globalState, setGlobalState] = useState<GlobalState>()
const [viewMode, setViewMode] = useState<ViewMode>('idle')
const [showCommandPalette, setShowCommandPalette] = useState(false)
const [showSettings, setShowSettings] = useState(false)
const [notifications, setNotifications] = useState<Notification[]>([])
```

Auto-Refresh: Polls backend every 3 seconds for state updates.

---

2. CodeViewer - Monaco Editor Integration

Purpose: Display and edit code files

Features:
- Syntax highlighting (20+ languages)
- Custom Heaven Dark theme
- Line numbers + minimap
- Auto-detect language from file extension

Usage:

```
<CodeViewer
  repoId="abc123"
  filePath="src/main.ts"
/>
```

Theme Config:

```
monaco.editor.defineTheme('heaven-dark', {
  base: 'vs-dark',
  colors: {
    'editor.background': '#1E1E1E',
    'editor.foreground': '#DDDDDD',
  }
})
```

---

3. AIAssistant - AI Chat Panel

Purpose: Interact with AI models (GPT-4, Claude, OSS-120B)

Features:
- Real-time chat
- Operation history
- Cost tracking
- Model selection
- Collapsible sidebar

Tabs:
- **Chat:** Live conversation with AI
- **History:** Past operations
- **Cost:** Budget tracking

API Calls:

```
await invoke('ai_chat', {
  repoId,
  workpadId,
  prompt: 'Add authentication',
  model: 'gpt-4',
})
```

---

4. CommandPalette - Quick Actions

Purpose: Keyboard-driven command interface (Cmd+P)

Features:
- Fuzzy search
- Keyboard navigation (↑↓)
- Categorized commands
- Visual shortcuts hints

Usage:

```
const commands = [
  {
    id: 'run-tests',
    label: 'Run Tests',
    description: 'Execute test suite',
    category: 'Testing',
    shortcut: 'Cmd+T',
    action: () => runTests(),
  }
]
```

---

5. TestDashboard - Metrics Visualization

Purpose: Display test results and trends

Features:
- Pass/fail trends (bar chart)
- Duration over time (line chart)
- Coverage placeholder
- Recent runs list

Charts:
- Built with Recharts
- Responsive design
- Dark theme integration

Data:

```
interface TestRun {
  test_run_id: string
  status: 'passed' | 'failed' | 'running'
  total_tests: number
  passed_tests: number
  duration_ms: number
  timestamp: string
}
```

---

6. FileBrowser - File Tree

Purpose: Navigate codebase

Features:
- Lazy loading (directories load on expand)
- Icon indicators (📁 folder, 📄 file)
- Click to open in editor
- Refresh button

Tree Structure:

```
interface FileNode {
  name: string
  path: string
  type: 'file' | 'directory'
  children?: FileNode[]
  expanded?: boolean
}
```

---

7. CommitGraph - Git Timeline

Purpose: Visualize commit history

Features:
- Linear timeline (Solo Git = no branches)
- Test status icons (✓ ✗ ◉)
- CI status integration
- Auto-refresh every 5 seconds

Indicators:
- ● = Trunk commit
- ○ = Workpad commit
- ✓ = Tests passed
- ✗ = Tests failed

Supporting Components

8. Settings - Configuration Panel

Sections:
- Editor (font, minimap, vim mode)
- AI & Models (default model, cost limits)
- Notifications (enable/disable)
- Appearance (theme)

9. NotificationSystem - Toast Alerts

Types:
- Success (green)
- Error (red)
- Warning (orange)
- Info (blue)

Auto-dismiss: 5 seconds default

10. KeyboardShortcutsHelp - Cheatsheet

Trigger: Press `?` key

Categories:
- Command Palette
- AI Assistant
- Testing
- Editor
- Navigation
- General

Testing Instructions

Manual Testing Scenarios

Scenario 1: Initialize Repository

Steps:
1. Launch Heaven GUI: `npm run tauri:dev`
2. Backend should show error: “No State Found”
3. Open terminal, run: `evogitctl repo init --zip test-app.zip`
4. GUI should auto-refresh and show repo info in status bar

Expected:
- ✅ Status bar shows repo ID
- ✅ File browser loads file tree

- ✅ Commit graph shows initial commit
- ✅ No errors in console

Scenario 2: Code Viewing

Steps:
1. Click a file in file browser (e.g., `src/main.ts`)
2. Code should appear in Monaco editor
3. Verify syntax highlighting
4. Check minimap on right side
5. Test scrolling and line numbers

Expected:
- ✅ Code loads within 500ms
- ✅ Correct language detected
- ✅ Line numbers visible
- ✅ Minimap shows document outline

Scenario 3: AI Assistant

Steps:
1. Press `Cmd+/` to open AI panel
2. Type prompt: “Explain this codebase”
3. Select model: GPT-4
4. Click Send (or press Enter)
5. Watch for streaming response

Expected:
- ✅ Panel slides in smoothly
- ✅ Model selector shows options
- ✅ Response appears in chat
- ✅ Cost displayed in message
- ✅ History tab shows operation

Scenario 4: Command Palette

Steps:
1. Press `Cmd+P`
2. Palette appears centered
3. Type: “run tests”
4. Command filters in real-time
5. Press Enter or click

Expected:
- ✅ Palette opens with animation
- ✅ Fuzzy search works
- ✅ Keyboard navigation (↑↓)

- ✅ ESC closes palette
- ✅ Command executes

Scenario 5: Test Dashboard

Steps:
1. Ensure workpad is active
2. Run tests: `evogitctl test fast`
3. Dashboard auto-refreshes
4. Click tabs: Trends, Duration, Coverage

Expected:
- ✅ Stats update (pass rate, duration)
- ✅ Bar chart shows pass/fail
- ✅ Line chart shows duration trend
- ✅ Recent runs list updates

Scenario 6: Keyboard Shortcuts

Test Each:
- `Cmd+P` → Command Palette ✓
- `Cmd+K` → Quick Search ✓
- `Cmd+B` → Toggle Sidebar ✓
- `Cmd+/` → Toggle AI ✓
- `Cmd+,` → Settings ✓
- `Cmd+E` → Zen Mode ✓
- `Cmd+T` → Run Tests ✓
- `?` → Shortcuts Help ✓
- `ESC` → Close Modals ✓

Expected:
- ✅ All shortcuts respond instantly
- ✅ No conflicts
- ✅ Help modal lists all shortcuts

Scenario 7: Settings Panel

Steps:
1. Press `Cmd+,`
2. Change font size to 16
3. Toggle minimap off
4. Change default model to GPT-3.5
5. Click Save
6. Verify changes persist

Expected:
- ✅ Settings modal opens
- ✅ Changes apply immediately

- ✅ Settings saved to backend
- ✅ Editor reflects font changes

Scenario 8: Error Handling

Steps:
1. Stop Solo Git backend
2. GUI should show connection error
3. Restart backend
4. GUI auto-reconnects

Expected:
- ✅ Error boundary doesn't crash app
- ✅ Helpful error message shown
- ✅ Retry button works
- ✅ Auto-reconnect after 3 seconds

Scenario 9: Zen Mode

Steps:
1. Open a file
2. Press `Cmd+E`
3. Both sidebars hide
4. Editor expands full-width
5. Press `Cmd+B` to restore

Expected:
- ✅ Smooth sidebar collapse
- ✅ Editor remains centered
- ✅ Status bar still visible
- ✅ Toggle restores layout

Scenario 10: Notifications

Steps:
1. Trigger action (e.g., run tests)
2. Notification appears top-right
3. Wait 5 seconds
4. Notification auto-dismisses
5. Click × to dismiss manually

Expected:
- ✅ Slide-in animation
- ✅ Correct type (success/error)
- ✅ Auto-dismiss after 5s
- ✅ Manual dismiss works
- ✅ Multiple notifications stack

Performance Testing

Metrics to Verify

Metric	Target	How to Test
Initial Load	< 2s	Open DevTools → Network tab
File Load	< 500ms	Click file, check response time
Command Palette	< 100ms	Press Cmd+P, measure to visible
Chart Render	< 1s	Switch dashboard tabs
Auto-Refresh	3s interval	Watch Network tab for polls
Memory Usage	< 500MB	Chrome DevTools → Performance Monitor

Load Testing

Large File (10,000 lines):

```
# Generate test file
seq 1 10000 | awk '{print "console.log(\"line "$1"\");"}' > large.js
```

Open in CodeViewer:
- ✅ Should load within 1 second
- ✅ Scrolling should be smooth (60 FPS)
- ✅ Minimap should render

Large File Tree (1,000 files):
- ✅ Lazy loading keeps initial render fast
- ✅ Expanding directory loads in < 500ms

Automated Testing (Future)

Recommended Tools:
- **Unit Tests:** Vitest
- **Component Tests:** React Testing Library
- **E2E Tests:** Playwright
- **Visual Regression:** Percy or Chromatic

Example Test:

```
// CodeViewer.test.tsx
import { render, screen } from '@testing-library/react'
import CodeViewer from './CodeViewer'

test('renders file content', async () => {
  render(<CodeViewer repoId="test" filePath="main.ts" />)

  await waitFor(() => {
    expect(screen.getByText(/console.log/)).toBeInTheDocument()
  })
})
```

Building for Production

Development Build

```
npm run build
```

Output: `dist/` directory

Tauri Production Build

```
npm run tauri:build
```

Creates platform-specific bundles:
- **macOS:** `.app` + `.dmg` in `src-tauri/target/release/bundle/`
- **Windows:** `.msi` + `.exe`
- **Linux:** `.AppImage` + `.deb`

Build Configuration

Tauri Config: `src-tauri/tauri.conf.json`

```
{
  "build": {
    "distDir": "../dist",
    "devPath": "http://localhost:5173"
  },
  "package": {
    "productName": "Heaven",
    "version": "0.1.0"
  },
  "tauri": {
    "bundle": {
      "identifier": "com.sologit.heaven",
      "icon": [
        "icons/icon.icns",
        "icons/icon.ico",
        "icons/icon.png"
      ]
    }
  }
}
```

Optimization Checklist

- [x] Code splitting (Vite automatic)
- [x] Tree shaking (Vite automatic)
- [x] Minification (Vite –minify)
- [x] Source maps disabled in prod
- [ ] Bundle analyzer (optional)
- [ ] Preloading critical assets

Bundle Size Targets:
- Vendor chunk: < 1MB
- App chunk: < 500KB
- Total: < 2MB uncompressed

Troubleshooting

Issue: GUI shows “No State Found”

Cause: Backend not running or wrong port

Fix:

```
# Check backend
curl http://localhost:8765/health

# Start backend
cd solo-git
python -m evogitctl serve
```

Issue: Monaco Editor not loading

Cause: Vite config issue or CDN blocked

Fix:

```
// vite.config.ts
export default defineConfig({
  optimizeDeps: {
    include: ['@monaco-editor/react', 'monaco-editor']
  }
})
```

Issue: Hot reload not working

Cause: Vite cache corruption

Fix:

```
rm -rf node_modules/.vite
npm run dev
```

Issue: Keyboard shortcuts not working

Cause: Input field has focus

Fix: Click outside input or press ESC first. Shortcuts disabled when typing.

Issue: Charts not rendering

Cause: Missing data or Recharts version mismatch

Fix:

```
npm install recharts@latest
```

Check data format:

```
// Must have at least 1 data point
const chartData = testRuns.map(run => ({
  name: `Run ${index}`,
  passed: run.passed_tests,
  failed: run.failed_tests,
}))
```

Issue: Memory leak on auto-refresh

Cause: Interval not cleaned up

Fix:

```
useEffect(() => {
  const interval = setInterval(loadState, 3000)
  return () => clearInterval(interval)  // Must cleanup!
}, [])
```

Issue: Build fails with TypeScript errors

Cause: Missing type definitions

Fix:

```
npm install --save-dev @types/d3 @types/react @types/react-dom
npm run type-check
```

Issue: Tauri window too small on launch

Cause: tauri.conf.json window size

Fix:

```
{
  "tauri": {
    "windows": [{
      "width": 1400,
      "height": 900,
      "minWidth": 1024,
      "minHeight": 768
    }]
  }
}
```

Development Tips

1. Use React DevTools

```
# Install Chrome extension
https://chrome.google.com/webstore/detail/react-developer-tools
```

Inspect component state and props in real-time.

---

2. Vite Dev Server Proxying

If backend is on different port:

```
// vite.config.ts
export default defineConfig({
  server: {
    proxy: {
      '/api': 'http://localhost:8765'
    }
  }
})
```

---

3. Tauri Debugging

Enable Rust backtrace:

```
RUST_BACKTRACE=1 npm run tauri:dev
```

---

4. CSS Live Editing

Vite hot-reloads CSS without refresh. Keep DevTools open to tweak styles.

---

5. Component Isolation

Test components in isolation:

```
// Test.tsx
export default function Test() {
  return (
    <div style={{ padding: 20, background: '#1E1E1E' }}>
      <CodeViewer repoId="test" filePath="main.ts" />
    </div>
  )
}
```

Change `App.tsx` to render `<Test />` temporarily.

Next Steps

1. **Implement Accessibility Fixes** (from UX Audit)
2. **Add Unit Tests** (React Testing Library)
3. **Add E2E Tests** (Playwright)
4. **Optimize Performance** (debounce, memoization)
5. **Add Light Theme** (optional)
6. **User Onboarding** (first-time guide)
7. **Telemetry** (optional, privacy-focused)

Resources

- **Tauri Docs:** https://tauri.app/v1/guides/
- **React Docs:** https://react.dev/
- **Monaco Editor:** https://microsoft.github.io/monaco-editor/
- **Recharts:** https://recharts.org/
- **D3.js:** https://d3js.org/

Questions? Open an issue on GitHub or contact the Solo Git team.

Last Updated: October 17, 2025
Version: 0.1.0