

# Auftrag: Local AI Workstation in eine vollständige Desktop-Anwendung umbauen

Arbeite direkt im bestehenden Repository der **Local AI Workstation** und implementiere den Umbau vollständig. Erstelle nicht nur einen Plan und liefere nicht nur Vorschläge. Untersuche zuerst die vorhandene Architektur, führe dann die Änderungen durch, teste sie und dokumentiere am Ende die tatsächlich vorgenommenen Änderungen.

## 1. Ziel

Baue die bestehende Local AI Workstation von einer schmalen, mobil wirkenden Oberfläche zu einer modernen, professionellen Desktop-Anwendung um.

Das Zielbild ist eine zentrale Desktop-Arbeitsoberfläche mit:

- dauerhaft sichtbarer linker Navigation,
- zentralem Arbeitsbereich,
- kontextabhängiger rechter Inspector-Seitenleiste,
- oberer Werkzeugleiste,
- dauerhaft sichtbarer unterer Statusleiste,
- vollständiger Nutzung des verfügbaren Tauri-Fensters,
- moderner dunkelblauer AI-/Developer-Optik,
- echter Agentenverwaltung,
- Agentenauswahl im Chat,
- weiterhin vollständig erreichbaren bestehenden Funktionen.

Die Anwendung soll sich wie eine lokale Entwicklungsumgebung und nicht wie eine mobil optimierte Webseite anfühlen.

---

## 2. Unveränderliche technische Rahmenbedingungen

### Bestehende Architektur beibehalten

Die Anwendung verwendet:

- Tauri 2,
- Vite,
- Vanilla TypeScript,
- normales HTML und CSS,
- FastAPI/Python als lokalen Kernel,
- llama.cpp als lokale Modell-Runtime.

Es gibt kein React, Vue, Svelte oder anderes Komponentenframework.

## Verbindliche Vorgaben

- Kein React installieren.
- Kein Vue installieren.
- Kein Svelte installieren.
- Kein Tailwind installieren.
- Kein neues UI-Framework installieren.
- Keine unnötige Runtime-Abhängigkeit ergänzen.
- Vorhandene Tauri-, Vite- und TypeScript-Struktur beibehalten.
- Vorhandene API-Endpunkte nicht umbenennen.
- Vorhandene API-Datentypen nicht inkompatibel verändern.
- Bestehende Sicherheits- und Bestätigungsmechanismen nicht umgehen.
- Bestehende Funktionen nicht entfernen.
- Bestehende Backend-Funktionen nicht durch Mock-Daten ersetzen.
- Keine falschen Statusanzeigen oder erfundenen Systemwerte darstellen.
- Keine bestehende Funktion nur deshalb entfernen, weil sie nicht in das neue Layout passt.

Lies vor Änderungen vorhandene Dateien wie `README.md`, `SECURITY.md`, `AGENTS.md`, `QWEN.md`, Roadmaps und projektspezifische Regeln, sofern sie im Repository vorhanden sind.

---

## 3. Wichtige Erkenntnisse aus dem aktuellen Code

Die Desktop-Oberfläche wird derzeit überwiegend in einer sehr großen `main.ts` erzeugt. Die globale Gestaltung liegt überwiegend in einer großen `style.css`.

Der aktuelle Chat übergibt fest:

```
agent: "assistant"
```

Diese feste Zuweisung muss durch eine echte Agentenauswahl ersetzt werden.

Der Python-Kernel besitzt bereits wesentliche Agentenfunktionen. Unter anderem existieren bereits:

- eine Agenten-Registry,
- Agentendefinitionen,
- Rollen,
- Beschreibungen,
- erlaubte Werkzeuge,
- Freigabepflichten,
- Freigabestatus,
- Autorisierung,
- Entzug einer Autorisierung,
- Tool-Policy-Prüfungen,
- ein Standardagent,
- ein `agent`-Feld in der Chat-Anfrage.

Der Kernel stellt bereits sinngemäß folgende Endpunkte bereit:

```
GET    /agents
POST   /agents/{agent_name}/authorize
POST   /agents/{agent_name}/revoke
POST   /chat/stream
```

Der Chat-Request unterstützt bereits:

```
agent?: string;
```

Erfinde daher kein zweites Agentensystem. Integriere die vorhandene Agentenverwaltung korrekt in das Desktop-Frontend.

Ändere das Backend nur, wenn ein nachweisbarer Fehler oder eine wirklich fehlende, für diese Integration notwendige Schnittstelle gefunden wird.

---

## 4. Bestehende Funktionen müssen erhalten bleiben

Alle derzeit vorhandenen Bereiche und Funktionen müssen nach dem Umbau weiterhin erreichbar und funktionsfähig sein:

- Chat und Streaming-Ausgabe,
- Modellwahl,
- Projektordner öffnen,
- Projekt indexieren,
- RAG-/Projektkontext,
- Datei auswählen und anhängen,
- Self-Workspace-Kontext,
- Web-Suchbegriffe,
- Web-URLs,
- Chat abbrechen,
- Chat leeren,
- Übergabe an das Terminal,
- Dienstverwaltung beziehungsweise Workflows,
- Selbstentwicklung,
- Plugin-Verwaltung,
- deklarative Plugin-Oberflächen,
- Code-Kontrolle,
- Terminal,
- Root-Modus mit Bestätigung,
- Modelle installieren,
- Modelle aktualisieren,
- Diagnose,

- Maßnahmenplan,
- Release-Update,
- Konfigurationsanzeige,
- CPU-, GPU- und RAM-Anzeige,
- FastAPI-Status,
- llama.cpp-Status,
- Tokenzähler,
- bestehende Tauri-Ereignisse,
- bestehende Fehler-, Lade- und Offline-Zustände.

Die **Selbstenentwicklung** muss ausdrücklich als eigener Navigationspunkt erhalten bleiben.

---

## 5. Zielarchitektur des Frontends

Die aktuelle monolithische Struktur soll kontrolliert modularisiert werden, ohne ein Framework einzuführen.

Ermittle zuerst die tatsächlichen Quellpfade. Verwende anschließend sinngemäß folgende Struktur:

```
desktop/src/
├─ main.ts
├─ api.ts
├─ agents.ts
├─ chat.ts
├─ status.ts
├─ ui/
│   ├─ app-shell.ts
│   ├─ navigation.ts
│   ├─ inspector.ts
│   ├─ status-bar.ts
│   ├─ icons.ts
│   ├─ dialogs.ts
│   └─ dom.ts
├─ pages/
│   ├─ chat-page.ts
│   ├─ agents-page.ts
│   ├─ models-page.ts
│   ├─ knowledge-page.ts
│   ├─ tools-page.ts
│   ├─ workflows-page.ts
│   ├─ self-development-page.ts
│   ├─ plugins-page.ts
│   ├─ code-review-page.ts
│   ├─ terminal-page.ts
│   └─ settings-page.ts
└─ styles/
    ├─ tokens.css
    └─ base.css
```

```

├─ shell.css
├─ components.css
└─ pages.css

```

Diese Struktur ist eine Zielrichtung, kein Grund, funktionierenden Code unnötig umzuschreiben.

## Regeln für die Modularisierung

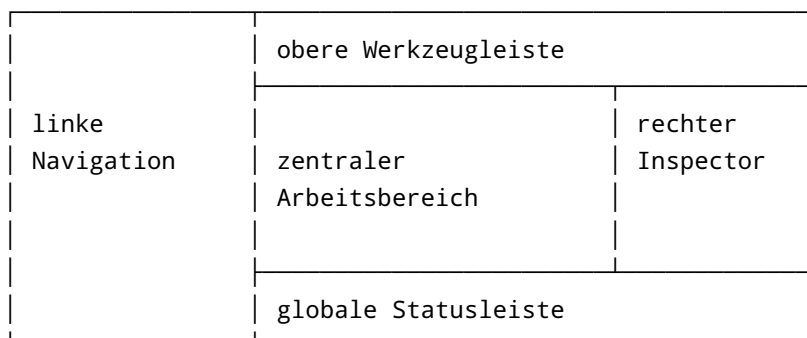
- `main.ts` soll Bootstrap, Initialisierung und übergreifende Koordination übernehmen.
- API-Kommunikation bleibt in `api.ts`.
- Parser und Typen für Agenten kommen in `agents.ts`.
- Statische UI-Templates und Shell-Elemente werden aus `main.ts` ausgelagert.
- Seitenspezifisches Rendering wird auf Seitenmodule verteilt.
- Gemeinsame SVG-Icons kommen in ein zentrales Icon-Modul.
- DOM-Selektoren sollen weiterhin typisiert werden.
- Event-Listener dürfen nicht mehrfach registriert werden.
- Bestehende Element-IDs möglichst erhalten.
- Werden IDs geändert, müssen alle zugehörigen Selektoren, Tests und Listener angepasst werden.
- Keine unsicheren dynamischen HTML-Inhalte aus API-Daten mit `innerHTML` einfügen.
- Für API-Daten `textContent`, DOM-Erzeugung und `replaceChildren` verwenden.

Der Umbau darf nicht in einem leeren Shell-Layout enden, bei dem bestehende Funktionen nicht wieder angeschlossen wurden.

## 6. Zentrales Desktop-Layout

### Grundaufbau

Die Anwendung soll das gesamte Fenster verwenden:



Empfohlene Größen:

```

--sidebar-width: 244px;
--sidebar-collapsed-width: 72px;

```

```
--inspector-width: 350px;  
--topbar-height: 68px;  
--statusbar-height: 42px;
```

## App-Shell

Sinngemäße Struktur:

```
<div class="app-shell">  
  <aside class="app-sidebar"></aside>  
  
  <section class="workspace-shell">  
    <header class="workspace-topbar"></header>  
  
    <div class="workspace-body">  
      <main class="page-host"></main>  
      <aside class="workspace-inspector"></aside>  
    </div>  
  
    <footer class="workspace-statusbar"></footer>  
  </section>  
</div>
```

## Verbindliches Verhalten

- `html`, `body` und `#app` erhalten `width: 100%` und `height: 100%`.
- `#app` darf keine maximale Breite mehr haben.
- Kein zentrierter 1080- oder 1180-Pixel-Container.
- Kein großer Außenabstand um die komplette Anwendung.
- Kein normales Scrollen des gesamten Dokuments.
- Navigation, Topbar und Statusbar bleiben sichtbar.
- Nur der aktive Inhaltsbereich scrollt.
- Der Inspector scrollt unabhängig, wenn sein Inhalt zu hoch ist.
- Keine großen ungenutzten Flächen.
- Keine mobile Hamburger-Navigation bei normalen Desktop-Breiten.
- Kein Mobile-first-Aufbau.
- Keine Karten, die unnötig die gesamte Breite untereinander belegen.

## Fenstergrößen

Das Layout muss mindestens bei folgenden Größen funktionieren:

- 1100 × 700,
- 1280 × 720,
- 1440 × 900,
- 1920 × 1080.

Bei schmaleren Desktop-Fenstern:

- linke Navigation auf Icon-Leiste reduzieren,
- rechten Inspector über einen Toggle ein- und ausblendbar machen,
- zentrale Chatfläche priorisieren,
- keine horizontale Scrollbar erzeugen.

Falls die Tauri-Konfiguration eine Mindestgröße definiert oder sinnvoll angepasst werden kann, verwende ungefähr:

```
minWidth: 1100  
minHeight: 700
```

Native Fensterdekorationen nicht entfernen, sofern das Projekt nicht bereits eine eigene Tauri-Titelleiste verwendet. Keine funktionslosen Fensterbuttons nachbilden.

---

## 7. Visuelle Gestaltung

### Stil

Die Oberfläche soll wie eine moderne lokale AI- und Developer-Workstation wirken:

- dunkles Navy statt des bisherigen grünlichen Themes,
- klare Flächenhierarchie,
- dezente violette und cyanfarbene Akzente,
- kompakte professionelle Typografie,
- feine Rahmen,
- leicht abgesetzte Panels,
- subtile Schatten,
- sparsame Verläufe,
- keine übertriebene Neon-Optik,
- kein großflächiges Glassmorphism,
- keine Emoji-Icons,
- keine riesigen Überschriften,
- keine mobile Kartenoptik.

### Design-Tokens

Verwende zentral definierte CSS-Variablen, etwa:

```
:root {  
  color-scheme: dark;  
  
  --color-bg-app: #070b14;  
  --color-bg-sidebar: #090f1b;  
  --color-bg-topbar: #0b1120;  
  --color-bg-panel: #0f1728;
```

```

--color-bg-panel-raised: #131d30;
--color-bg-input: #0b1322;
--color-bg-hover: #17223a;
--color-bg-active: #1d2850;

--color-border: #202d46;
--color-border-strong: #30405f;

--color-text: #edf2ff;
--color-text-secondary: #b8c2d8;
--color-text-muted: #7f8ba4;

--color-accent: #6d5dfc;
--color-accent-hover: #7c6cff;
--color-accent-secondary: #22c9e8;

--color-success: #22c55e;
--color-warning: #eab308;
--color-danger: #ef4444;
--color-info: #38bdf8;

--radius-small: 6px;
--radius-medium: 10px;
--radius-large: 14px;

--shadow-panel: 0 12px 32px rgb(0 0 0 / 18%);
}

```

Die genauen Werte dürfen leicht angepasst werden, die visuelle Richtung muss jedoch erhalten bleiben.

## Typografie

```

font-family:
  Inter,
  "Noto Sans",
  system-ui,
  -apple-system,
  BlinkMacSystemFont,
  "Segoe UI",
  sans-serif;

```

Für Code und Terminal:

```

font-family:
  "JetBrains Mono",
  "Fira Code",

```

```
"Cascadia Code",  
monospace;
```

Keine externen Webfonts zur Laufzeit herunterladen.

## Icons

- Einheitliche Inline-SVG-Icons verwenden.
- Größe überwiegend 17 bis 19 Pixel.
- Strichstärke ungefähr 1.7 bis 2.
- Einheitlicher Stil.
- Keine Unicode-Symbole als Ersatz für UI-Icons.
- Keine Emojis.
- Keine neue Icon-Bibliothek installieren, wenn die benötigten Icons als zentral verwaltete SVGs umgesetzt werden können.

## 8. Linke Navigation

### Kopfbereich

Oben links:

- vorhandenes Local-AI-Logo verwenden, falls im Repository vorhanden,
- ansonsten eine kompakte Hexagon-/Node-Marke als Inline-SVG,
- Text:

```
Local AI  
WORKSTATION
```

**WORKSTATION** wird klein, mit erhöhter Zeichenweite und gedeckter Farbe dargestellt.

### Navigationspunkte

Die Navigation soll sich am Zielbild orientieren und gleichzeitig alle bestehenden Funktionen abbilden:

```
Chat  
Agenten  
Modelle  
RAG / Wissen  
Tools  
Workflows  
Selbstentwicklung  
Plugins  
Code-Kontrolle
```

## Zuordnung bestehender Funktionen

- **Chat** → bestehender Chat.
- **Agenten** → neue Agentenverwaltung.
- **Modelle** → bestehende Modellübersicht, Installation und Aktualisierung.
- **RAG / Wissen** → bestehende Projekt-, Index-, Datei- und Kontextfunktionen.
- **Tools** → Übersicht tatsächlich vorhandener Kernel-Werkzeuge und Tool-Berechtigungen.
- **Workflows** → bisherige Dienstverwaltung beziehungsweise Automation.
- **Selbstentwicklung** → vollständig erhalten.
- **Plugins** → vollständig erhalten.
- **Code-Kontrolle** → vollständig erhalten.
- **Terminal** → vollständig erhalten.
- **Einstellungen** → Status, Diagnose, Konfiguration, Release und allgemeine Einstellungen.

Die bisherige Bezeichnung **Dienstverwaltung** darf im Seiteninhalt ergänzend erscheinen, der Navigationspunkt soll jedoch **Workflows** heißen.

Keine leeren oder funktionslosen Fake-Seiten erzeugen.

Ein Bereich **Prompts** soll in diesem Auftrag nur dann ergänzt werden, wenn im bestehenden Projekt bereits eine echte Prompt-Verwaltung existiert. Keine leere Prompt-Seite nur zur optischen Übereinstimmung erzeugen.

## Unterer Sidebar-Bereich

Unten in der Navigation:

- kompakter Systemstatus,
- grüner Punkt bei funktionsfähigem Kernel,
- Text wie **Alle Systeme betriebsbereit**,
- Status von FastAPI und llama.cpp,
- Desktop-/Kernel-Version,
- Button zum Ein- und Ausklappen der Sidebar.

## Aktiver Zustand

Der aktive Navigationspunkt erhält:

- leicht violett-blauen Hintergrund,
- farbige linke Markierung oder weichen inneren Akzent,
- hellen Text,
- farbiges Icon.

Andere Punkte bleiben zurückhaltend.

## 9. Obere Werkzeugleiste

Die Topbar zeigt abhängig von der aktiven Seite:

- Seitentitel,
- kurze Unterzeile,
- seitenspezifische Aktionen.

Für den Chat:

Chat  
Dein lokaler AI-Arbeitsbereich

Rechts:

- Neuer Chat,
- Importieren,
- Exportieren,
- Inspector ein-/ausblenden,
- gegebenenfalls kompakter Aktualisieren-Button.

### Chat-Sitzungen

Implementiere eine einfache lokale Sitzungsverwaltung, sofern dies ohne Eingriff in den Kernel möglich ist:

- lokale Speicherung in `localStorage`,
- maximal 20 Sitzungen,
- Titel aus der ersten Nutzernachricht,
- Änderungszeitpunkt,
- Neuer-Chat-Funktion,
- aktuelle Sitzung in der Sidebar anzeigen,
- ältere Sitzungen unter `Letzte Chats`,
- Sitzung auswählbar,
- Sitzung löschtbar.

Speichere keine angehängten Dateiinhalte und keine geheimen Daten automatisch. Gespeichert werden dürfen nur Chatnachrichten und unkritische UI-Metadaten wie gewähltes Modell und gewählter Agent.

Import und Export:

- versioniertes JSON-Format,
- optional zusätzlicher Markdown-Export,
- Eingabedaten validieren,
- ungültige Dateien verständlich ablehnen,
- vorhandene Tauri-Dialog- und Dateisystemfunktionen verwenden.

Falls dieser Teil den stabilen Kernumbau gefährdet, zuerst Shell, Agenten und vorhandene Funktionen vollständig abschließen. Die Anwendung darf nicht wegen der Sitzungsverwaltung unvollständig bleiben.

---

## 10. Chat-Seite

### Dreispaltige Hauptansicht

Auf der Chat-Seite:

1. linke globale Navigation,
2. zentraler Chat,
3. rechter Chat-Inspector.

### Chat-Kopf

Oben im zentralen Chatpanel:

- Agentenauswahl,
- Modellauswahl,
- Status des aktiven Runners,
- Projektstatus,
- kompakter Mehr-Aktionen-Button.

Agent und Modell müssen getrennt auswählbar sein.

Beispiel:

```
Agent: Coding Assistant  
Modell: Qwen2.5-Coder-7B  
Runner: llama.cpp · aktiv
```

### Nachrichten

Nachrichten sollen klarer als bisher dargestellt werden:

- Nutzer-Avatar beziehungsweise neutrales Nutzer-Icon,
- Agenten-Icon,
- Anzeigename `Du`,
- Agentenname statt nur `Assistent`,
- Zeitstempel,
- ausreichend breite Textfläche,
- Codeblöcke mit Monospace-Schrift,
- Kopierbutton bei Codeblöcken,
- keine schmalen Chatblasen wie bei einem Messenger,
- Antworten dürfen nahezu die gesamte Chatbreite verwenden.

Die bestehende Streaming-Logik muss erhalten bleiben.

## Eingabebereich

Der Composer bleibt am unteren Rand des zentralen Chatbereichs sichtbar.

Enthalten:

- mehrzeiliges Texteingabefeld,
- Projektordner öffnen,
- Datei anhängen,
- an Terminal übergeben,
- Stoppen,
- Senden,
- optionale erweiterte Kontextoptionen.

`Enter` sendet weiterhin.

`Shift + Enter` erzeugt weiterhin eine neue Zeile.

## Projekt- und Kontextfunktionen

Die bestehenden Projektfunktionen dürfen nicht entfernt werden.

Ordne sie kompakter:

- Projekt öffnen,
- Projekt indexieren,
- Datei auswählen,
- Datei verifizieren/anhängen,
- RAG-Status,
- Projektpfad,
- Self-Workspace,
- Web-Suchbegriffe,
- Web-URLs.

Erweiterte Kontextoptionen können in einem aufklappbaren Panel oder einem Inspector-Abschnitt liegen. Der Benutzer muss aber weiterhin erkennen können:

- welches Projekt geöffnet ist,
- ob der Index aktiv ist,
- ob eine Datei verifiziert wurde,
- welche Datei angehängt wurde.

## Untere Chat-Statuszeile

Im Chatpanel:

- aktives Modell,
- aktiver Agent,
- Prompt-Tokens,

- Antwort-Tokens,
- Gesamt-Tokens,
- Projektstatus,
- gegebenenfalls Antwortdauer, sofern real messbar.

Keine Werte erfinden.

## 11. Agentenfunktion

### Frontend-Datentypen

Erstelle ein Modul `agents.ts` mit strikt validierten Typen und Parsern.

Sinngemäß:

```
export interface AgentStatus {
  name: string;
  role: string;
  description: string;
  allowed_tools: string[];
  approval_required: boolean;
  approved: boolean;
  state: string;
}

export interface AgentReport {
  default_agent: string;
  agents: AgentStatus[];
}
```

API-Daten müssen wie bei den bestehenden Modulen zur Laufzeit validiert werden.

### API-Funktionen

Ergänze in `api.ts`:

```
loadAgents(): Promise<AgentReport>

authorizeAgent(
  agentName: string,
  confirmed: boolean,
): Promise<AgentStatus>

revokeAgent(
```

```
agentName: string,  
) : Promise<AgentStatus>
```

Verwende die bestehenden Endpunkte.

Fehlerantworten müssen über die vorhandene API-Fehlerbehandlung ausgegeben werden.

## Agenten-Seite

Die neue Seite **Agenten** enthält:

### Kopfbereich

Agenten  
Lokale Rollen, Berechtigungen und Werkzeuge verwalten

Aktionen:

- **Neu laden**,
- optional **Mit ausgewähltem Agenten chatten**.

Keinen **Agent erstellen**-Button anzeigen, solange keine echte sichere API zum Bearbeiten der Agentenkonfiguration existiert.

### Agentenkarten

Für jeden Agenten anzeigen:

- Name,
- Rolle,
- Beschreibung,
- Standardagent-Markierung,
- Status,
- Freigabepflicht,
- erlaubte Werkzeuge,
- Anzahl der Werkzeuge,
- **Freigeben** oder **Freigabe entziehen**,
- **Im Chat verwenden**.

Statusdarstellung:

- **Freigegeben**,
- **Freigabe erforderlich**,
- **Ohne Freigabe nutzbar**,
- Fehlerzustand, falls Konfiguration ungültig ist.

Werkzeuge als kompakte Chips darstellen.

Agenten dürfen nicht hardcodiert werden. Die Darstellung wird vollständig aus `GET /agents` erzeugt.

## Autorisierung

Bei `Freigeben`:

- explizite Bestätigung anzeigen,
- Agentenname nennen,
- Rolle nennen,
- erlaubte Werkzeuge auflisten,
- anschließend `{ confirmed: true }` an die vorhandene API senden.

Bei `Freigabe entziehen`:

- verständlich bestätigen lassen,
- danach den Agentenstatus neu rendern.

Keine automatische Freigabe beim Laden der Anwendung.

## Agentenauswahl im Chat

Im Chat wird ein Agenten-Select ergänzt.

Regeln:

- Standardwert ist `default_agent` aus der API.
- Nicht freigegebene, freigabepflichtige Agenten dürfen nicht ohne Freigabe ausgeführt werden.
- Sie können deaktiviert im Select erscheinen oder einen klaren Hinweis erhalten.
- Über einen danebenliegenden Button kann die Freigabe gestartet werden.
- Nicht freigabepflichtige Agenten sind direkt auswählbar.
- Auswahl in `localStorage` merken.
- Gespeicherte Auswahl beim Start gegen die aktuelle Agentenliste validieren.
- Ist der gespeicherte Agent nicht mehr verfügbar, Backend-Standardagent verwenden.
- Der aktive Agent wird im rechten Inspector und in der Chatnachricht angezeigt.

Ersetze die feste Chatzuweisung:

```
agent: "assistant"
```

durch:

```
agent: selectedAgentName
```

Ein sinnvoller sicherer Fallback auf den vom Backend gemeldeten Standardagenten ist erlaubt.

## Agenten und Werkzeuge

Der rechte Inspector zeigt für den aktiven Agenten:

- Rolle,
- Status,
- erlaubte Werkzeuge,
- Freigabestatus.

Keine Werkzeuge als **Aktiv** anzeigen, wenn sie vom Agenten nicht erlaubt sind.

Keine autonome Hintergrundaufführung ergänzen. Bestehende Bestätigungs- und Tool-Policies bleiben verbindlich.

---

## 12. Rechter Inspector

Der Inspector ist kontextabhängig.

### Auf der Chat-Seite

#### Panel: Modell & Runner

Anzeigen:

- gewähltes Modell,
- aktives Modell,
- Runner `llama.cpp`,
- FastAPI-Status,
- Kontextlänge aus echten Einstellungen, sofern vorhanden,
- Button zur Modellseite.

Keine Kontextlänge hardcodieren.

#### Panel: Systemressourcen

Anzeigen:

- CPU,
- RAM,
- GPU, sofern verfügbar,
- Speichernutzung nur, wenn sie wirklich vom Backend geliefert oder sauber ergänzt wird.

Für CPU, RAM und GPU dürfen clientseitig kleine Sparklines aus den letzten Statuswerten geführt werden:

- maximal ungefähr 30 Messpunkte,
- reines SVG oder Canvas,
- keine Chart-Bibliothek,
- keine erfundenen historischen Werte,

- Verlauf beginnt erst mit real empfangenen Daten.

Status ungefähr alle fünf Sekunden aktualisieren, solange das Fenster sichtbar ist. Polling pausieren, wenn `document.hidden` wahr ist.

## Panel: Werkzeuge

Zeige die tatsächlich erlaubten Tools des aktiven Agenten.

Technische Namen sinnvoll übersetzen, zum Beispiel:

|                                    |                        |
|------------------------------------|------------------------|
| <code>chat</code>                  | → Chat                 |
| <code>workspace.inspect</code>     | → Projekt prüfen       |
| <code>workspace.read</code>        | → Dateien lesen        |
| <code>workspace.index</code>       | → Projekt indexieren   |
| <code>workspace.search</code>      | → Projekt durchsuchen  |
| <code>workspace.patch</code>       | → Dateien ändern       |
| <code>terminal.execute</code>      | → Terminal             |
| <code>installation.plan</code>     | → Installationsplan    |
| <code>installation.validate</code> | → Plan validieren      |
| <code>service.status</code>        | → Dienststatus         |
| <code>health.check</code>          | → Systemprüfung        |
| <code>model.install</code>         | → Modelle installieren |

Unbekannte Toolnamen weiterhin anzeigen, aber lesbar formatieren.

Die im visuellen Zielbild vorkommenden Bezeichnungen wie:

- Web-Suche,
- Dateisystem,
- Code-Ausführung,
- Terminal,
- Bildgenerierung

dürfen nur als `Aktiv` erscheinen, wenn tatsächlich eine entsprechende Funktion oder ein entsprechender Tool-Adapter vorhanden ist.

Nicht konfigurierte Funktionen mit `Nicht verfügbar` oder `Nicht konfiguriert` markieren. Keine Funktion vortäuschen.

## Andere Seiten

Der Inspector zeigt jeweils passende Informationen:

- Agenten → Details des ausgewählten Agenten,
- Modelle → aktives Modell und Ressourcen,
- RAG/Wissen → Projektpfad, Indexstatus und Dateikontext,
- Tools → Tool-Berechtigungen,
- Workflows → ausgewählter Workflow und Freigabestatus,

- Selbstentwicklung → Status, Modelle und Checkpoint,
  - Plugins → Pluginstatus und Interfaces,
  - Terminal → Arbeitsverzeichnis und Root-Modus,
  - Einstellungen → Versions- und Dienststatus.
- 

## 13. Modelle-Seite

Extrahiere die bestehende Modellverwaltung aus der allgemeinen Statusseite in einen eigenen Navigationsbereich.

Anzeigen:

- aktives Modell,
- Standardmodell,
- Rollen,
- Dateiname,
- Größe,
- Verfügbarkeit,
- Server-ID,
- Installation,
- Aktualisierung,
- Runtime-Status.

Bestehende Installations- und Aktualisierungsfunktionen unverändert anbinden.

Keine Modellnamen oder Größen hardcodieren.

---

## 14. RAG-/Wissen-Seite

Nutze vorhandene Workspace-Funktionen:

- Projektordner öffnen,
- Struktur einlesen,
- Projekt indexieren,
- Dateien anzeigen,
- Datei lesen,
- verifizierten Dateikontext darstellen,
- Indexstatus anzeigen.

Die Seite soll keine zweite unabhängige Projektverwaltung einführen. Chat, Code-Kontrolle, Terminal und RAG-Seite müssen denselben aktuellen Workspace-Zustand verwenden.

Zeige deutlich:

- kein Projekt geöffnet,
- Projekt geöffnet, aber nicht indexiert,
- Index aktiv,

- Datei ausgewählt,
  - Datei verifiziert,
  - Datei gekürzt,
  - Zugriff abgelehnt.
- 

## 15. Tools-Seite

Die Tools-Seite ist eine echte Übersicht vorhandener Kernel- und Agentenwerkzeuge.

Anzeigen:

- technischer Toolname,
- lesbarer Name,
- Beschreibung,
- vom aktiven Agenten erlaubt oder nicht erlaubt,
- gegebenenfalls Plugin-Interface,
- Freigabepflicht,
- verfügbar oder nicht verfügbar.

Keine generische freie Shell-Ausführung ergänzen.

Keine Bildgenerierung implementieren, wenn dafür kein Backend existiert.

Keine Code-Interpreter-Sandbox vortäuschen.

Das vorhandene kontrollierte Terminal bleibt der reale Ausführungskanal.

---

## 16. Workflows, Selbstentwicklung und übrige Seiten

### Workflows

Die bisherige Dienstverwaltung wird visuell in den neuen Desktop-Stil überführt.

Erhalten bleiben:

- Workflow-Gruppen,
- Schritte,
- Freigabepflicht,
- Warnungen,
- Ausführung,
- Ergebnisse,
- Fehlerzustände.

## Selbstentwicklung

Der Navigationspunkt muss erhalten bleiben.

Darstellen:

- Codemodell,
- Kontrollmodell,
- Projektordner,
- Ideen und Aufgaben,
- Start,
- Stop,
- Checkpoint,
- Analyse-Log,
- Review-Log,
- Umsetzungs-Log,
- Entwürfe,
- detailliertes Log.

Die Selbstentwicklung soll den zentralen Arbeitsbereich gut ausnutzen und nicht in schmalen mobilen Karten untereinander erscheinen.

## Plugins

Erhalten:

- Suche,
- Statusfilter,
- Freigabefilter,
- Aktivieren,
- Deaktivieren,
- Interfaces,
- deklarative Plugin-Panels.

## Code-Kontrolle

Erhalten:

- Schreibmodell,
- Reviewmodell,
- Projekt öffnen,
- Projekt indexieren,
- Datei laden,
- Codeeingabe,
- Ziel der Prüfung,
- Review-Hinweise,
- überarbeitete Version,
- Speichern unter.

Nutze auf großen Fenstern zwei echte nebeneinanderliegende Editor-/Ergebnisbereiche.

## Terminal

Erhalten:

- Arbeitsverzeichnis,
- Root-Modus,
- Bestätigungsdialog,
- Befehlsfeld,
- Ausführen,
- Policy-Hinweis,
- Exit-Code,
- Ausgabe.

Terminalausgabe soll den verfügbaren Arbeitsbereich vollständig nutzen.

## Einstellungen

Fasse hier zusammen:

- Systemstatus,
- Diagnose,
- Maßnahmenplan,
- Release-Update,
- Konfiguration,
- Versionen,
- Dienststatus.

Keine dieser bestehenden Funktionen entfernen.

---

## 17. Globale Statusleiste

Die bisherige globale Statusanzeige wird zu einer echten schmalen Desktop-Statusbar am unteren Fensterrand.

Anzeigen:

- CPU,
- GPU,
- RAM,
- aktives Modell,
- aktiver Agent,
- FastAPI,
- llama.cpp,
- Gesamt-Tokens,
- Desktop-Version oder Kernel-Version kompakt.

Die Leiste:

- ist nicht als große Karte gestaltet,

- hat keine Pillen um jeden einzelnen Wert,
- verwendet kleine Trennlinien oder Abstände,
- bleibt am unteren Rand,
- nimmt ungefähr 36 bis 44 Pixel Höhe ein,
- zeigt Grün nur für positive Statuswerte,
- zeigt Rot nur für echte Fehler.

---

## 18. CSS-Bereinigung

Die bestehende `style.css` enthält mehrere nacheinander ergänzte Layoutphasen und überschreibende Regeln.

Führe eine kontrollierte Bereinigung durch:

- alte mobile Layoutregeln entfernen, wenn sie nicht mehr benötigt werden,
- doppelte Selektoren zusammenführen,
- widersprüchliche `#app`-Definitionen entfernen,
- alte grüne Farbwerte durch Design-Tokens ersetzen,
- alte `Compact desktop layout`-Überschreibungen nicht zusätzlich weiter überschreiben,
- ungültige oder verwaiste Klammern korrigieren,
- Seitengrößen über Grid und Flex statt über zahlreiche `calc(100vh - ...)`-Einzelwerte steuern,
- Komponentenklassen wiederverwendbar gestalten,
- Fokus-, Hover-, Active- und Disabled-Zustände vereinheitlichen.

Keine neue dritte CSS-Schicht einfach ans Dateieinde hängen. Das bestehende CSS muss tatsächlich bereinigt oder sinnvoll aufgeteilt werden.

---

## 19. Zustände und Fehlerbehandlung

Jeder datenabhängige Bereich benötigt:

- Ladezustand,
- leerer Zustand,
- Erfolgszustand,
- Fehlerzustand,
- Offline-Zustand,
- deaktivierten Zustand.

Beispiele:

```
Agenten werden geladen ...
Keine Agenten konfiguriert.
Agent wurde freigegeben.
Agentenstatus konnte nicht geladen werden.
```

FastAPI ist nicht erreichbar.  
Modell-Runtime ist offline.

Fehler dürfen die restliche Navigation nicht blockieren.

Wenn FastAPI offline ist:

- Shell und Navigation bleiben sichtbar,
- Agentenseite zeigt Offline-Status,
- Chat-Senden wird deaktiviert,
- Einstellungen und lokale UI bleiben erreichbar,
- keine nicht behandelten Promise-Rejections,
- keine leere weiße oder schwarze Ansicht.

---

## 20. Barrierefreiheit und Bedienung

- Buttons benötigen verständliche `aria-label`-Attribute, wenn sie nur ein Icon zeigen.
- Aktiver Navigationspunkt verwendet `aria-current="page"`.
- Formfelder besitzen sichtbare oder zugängliche Labels.
- Fokuszustände müssen deutlich sichtbar sein.
- Kontrast muss im dunklen Theme ausreichend sein.
- Bedienung mit Tastatur erhalten.
- Escape schließt Dialoge und mobile Inspector-Overlays.
- Bestätigungsdialoge nennen die konkrete Aktion.
- Tooltips nur ergänzend, nicht als einzige Beschriftung.

---

## 21. Tests

### Frontend

Ergänze Vitest-Tests mindestens für:

- Parser von `AgentReport`,
- Parser von `AgentStatus`,
- ungültige Agenten-API-Daten,
- Auswahl des Standardagenten,
- Fallback bei nicht mehr vorhandenem gespeicherten Agenten,
- Auswahl eines freigegebenen Agenten,
- Blockierung eines nicht freigegebenen Agenten,
- Chat-Request enthält den ausgewählten Agenten,
- Autorisierungs-Request,
- Revoke-Request beziehungsweise zugehörige Logik,
- Sitzungsimport, falls implementiert.

Bestehende Tests müssen weiterhin bestehen.

Ausführen:

```
npm test
npm run build
```

Nutze die tatsächlichen Projektpfade und Paketmanager-Konventionen des Repositories.

## Backend

Da die Agentenendpunkte bereits existieren, ändere sie nicht unnötig.

Falls Backend-Code geändert wird:

- vorhandene Python-Tests ausführen,
- Agenten-Autorisierung testen,
- Agenten-Revoke testen,
- Tool-Policy nicht abschwächen,
- Chat mit ausgewähltem Agenten testen.

---

## 22. Verbindliche Abnahmekriterien

Der Auftrag ist erst abgeschlossen, wenn alle folgenden Punkte erfüllt sind:

1. Die Anwendung nutzt die gesamte Breite und Höhe des Tauri-Fensters.
2. `#app` besitzt keine maximale Breite mehr.
3. Die Navigation ist auf Desktop dauerhaft sichtbar.
4. Der Chat befindet sich im zentralen Arbeitsbereich.
5. Der rechte Inspector ist vorhanden und kontextabhängig.
6. Die Statusleiste befindet sich dauerhaft unten.
7. Alle bisherigen Navigationsbereiche und Funktionen sind weiterhin erreichbar.
8. Selbstentwicklung wurde nicht entfernt.
9. Eine neue Agenten-Seite ist vorhanden.
10. Agenten werden dynamisch über `GET /agents` geladen.
11. Agenten können über die vorhandene API freigegeben werden.
12. Agentenfreigaben können über die vorhandene API entzogen werden.
13. Der Standardagent des Backends wird berücksichtigt.
14. Im Chat kann ein freigegebener Agent ausgewählt werden.
15. Der ausgewählte Agent wird an `/chat/stream` übergeben.
16. Die feste Zuweisung `agent: "assistant"` ist entfernt.
17. Nicht freigegebene Agenten werden nicht unbemerkt ausgeführt.
18. Tool-Berechtigungen werden wahrheitsgemäß dargestellt.
19. Keine Fake-Systemmetriken werden angezeigt.
20. Keine bestehende Sicherheitsbestätigung wurde entfernt.
21. Kein React, Vue, Svelte oder Tailwind wurde eingeführt.
22. Keine neue unnötige UI-Abhängigkeit wurde installiert.
23. Das alte grün dominierte Theme wurde durch das Navy-/Violett-/Cyan-Theme ersetzt.
24. Das CSS enthält keine konkurrierenden alten Desktop- und Mobile-Layoutblöcke mehr.

25. Bei 1280 × 720 entsteht keine horizontale Scrollbar.
  26. Bei 1920 × 1080 entstehen keine großen ungenutzten Außenflächen.
  27. Chat-Streaming funktioniert weiterhin.
  28. Projektindexierung und Dateianhang funktionieren weiterhin.
  29. Terminal und Root-Bestätigung funktionieren weiterhin.
  30. Modellinstallation und Modellaktualisierung funktionieren weiterhin.
  31. Plugins, Workflows, Code-Kontrolle und Selbstentwicklung funktionieren weiterhin.
  32. Offline-Zustände erzeugen keine leere Anwendung.
  33. `npm test` ist erfolgreich.
  34. `npm run build` ist erfolgreich.
  35. Im Browser-/Tauri-Log erscheinen keine neuen unbehandelten Fehler.
- 

## 23. Arbeitsweise

Gehe in dieser Reihenfolge vor:

1. Repository-Regeln lesen.
2. Frontend-, Backend- und Tauri-Struktur prüfen.
3. Vorhandene Agenten-API verifizieren.
4. Aktuelle Tests ausführen.
5. Agententypen und Agenten-API im Frontend integrieren.
6. Agentenauswahl im Chat funktionsfähig machen.
7. Desktop-App-Shell erstellen.
8. bestehende Seiten in die neue Shell übertragen.
9. rechten Inspector implementieren.
10. globale Statusbar umbauen.
11. CSS bereinigen und Design-Tokens einführen.
12. alle bestehenden Event-Listener und Funktionen erneut prüfen.
13. Tests ergänzen.
14. Tests und Build ausführen.
15. abschließenden Diff auf entfernte Funktionen prüfen.

Stoppe nicht nach der Analyse. Implementiere den Auftrag vollständig.

Falls während der Umsetzung eine Unklarheit entsteht, wähle die Variante, die:

- bestehende Funktionalität erhält,
  - bestehende APIs wiederverwendet,
  - Sicherheitsregeln nicht abschwächt,
  - keine Fake-Funktion erzeugt,
  - die geringste unnötige Architekturänderung verursacht.
- 

## 24. Abschlussbericht

Gib nach der Umsetzung einen kompakten Abschlussbericht aus mit:

## Geändert

- neue und geänderte Dateien,
- neue Agentenfunktionen,
- neue UI-Struktur,
- überführte bestehende Bereiche.

## Technische Entscheidungen

- wie die Agenten-API eingebunden wurde,
- wie der aktive Agent im Chat verwendet wird,
- wie die zentrale App-Shell aufgebaut wurde,
- welche CSS-Altteile entfernt wurden.

## Tests

- ausgeführte Befehle,
- Testergebnisse,
- Build-Ergebnis.

## Noch offen

Nur echte verbleibende Einschränkungen nennen. Keine Funktionen als fertig bezeichnen, die nur optisch dargestellt oder noch nicht an das Backend angebunden wurden.