File Explorer

Ein Projekt für den Informaticup

Autoren:
Brandt Marco (Matrikel-Nr. 13779)
Lauritz Wiebusch (Matrikel-Nr. 13771)
Sören Panten(Matrikel-Nr. 13766)
Schatz Daniel (Matrikel-Nr. 13796)

Projekt: Informaticup
Abgabedatum: keine Ahnung

Inhaltsverzeichnis

1 Intro

Software für den Eigengebrauch zu entwickeln – mit genau den Funktionen, die man sich über Jahre hinweg im Alltag gewünscht hat – stellt nicht nur eine kreative, sondern auch eine methodisch interessante Herangehensweise dar. Gerade im Bereich der Dateiverwaltung, der tagtäglich durch den Einsatz verschiedenster File Explorer geprägt ist, zeigt sich ein eklatantes Missverhältnis zwischen praktischer Nutzung und funktionaler Erfüllung. Während bestehende Systeme primär darauf ausgerichtet sind, Dateien anzuzeigen und ein minimales Interface zur Verfügung zu stellen, bleiben tiefere Anforderungen moderner Nutzerinnen und Nutzer häufig unberücksichtigt.

Die Möglichkeit, einen File Explorer von Grund auf neu zu denken, eröffnet daher nicht nur den Freiraum zur Integration lang ersehnter Features, sondern stellt auch ein erkenntnisreiches Experiment in der Human-Computer-Interaktion dar. Welche Strukturen, Navigationsprinzipien und Automatisierungsansätze fördern tatsächliche Effizienz? Wie kann ein Interface gestaltet sein, das nicht nur funktioniert, sondern den Nutzer aktiv unterstützt, informiert und entlastet? Eine solche Neuentwicklung ist nicht lediglich als technischer Fortschritt zu verstehen, sondern als reflektierter Beitrag zur Evolution alltäglicher Softwarewerkzeuge.

Auf das Projekt sind wir aufmerksam geworden durch ein YouTube Video [1]. In diesem erklärt er seine Idee von einem schnellen File Explorer

Wir haben dieses Projekt spezifisch ausgewählt, da wir den Ersteller selbst kontaktiert haben und er sich an unserem Interesse erfreut hat und der Explorer optisch nicht ansprechend gestaltet war, wenige Funktionen vorhanden waren und vieles auch nicht cross Plattform kompatibel war. Außerdem hatte der Ersteller tiefgehende Probleme bezüglich der Suchfunktion

2 Daten vom Backend zum Frontend

Datenverarbeitung findet hauptsächlich im Backend statt. Dieses besteht im Framework Tauri [3] aus Rust Code. Das Frontend kann Methoden aufrufen und über die asynchrone runntime können Daten an das Frontend geschickt werden. Dies ist recht einfach über Json oder sonst auch normalen primitiven Datentypen. Wenn selbst erstellte Objekte durchgereicht werden sollen, müssen Felder usw. im Frontend den selben Namen tragen und auch gleich Strukturiert sein. Für das Projekt haben wir uns allerdings entschieden Daten ausschließlich über Json bereitzustellen. Dies bildet eine einheitliche Lösung und es werden keine Probleme auftreten, welche aufgrund von verschiedenen Datentypen auftauchen und fehlerhafter Konvertierung.

Die Methoden, welche aus dem Frontend aufgerufen werden können, werden als `#[tauri::command]` markiert und dem Builder für die Applikation hinzugefügt. Somit muss jeder Endpoint registriert werden. Dann jedoch können Methoden über ihren Namen aus dem Frontend aus erreicht werden. Wichtig ist, dass jede Methode automatisch *asynchron* ist.

```
#[tauri::command]
pub async fn open_file(path: &str) -> Result<String, String> {
    let path_obj = Path::new(path);
    // Check if path exists
    if !path_obj.exists() {
        return Err(format!(//TODO));
    }
    // Check if path is a file
    if !path_obj.is_file() {
        return Err(format!(//TODO));
    }
    // Read the file pfs
    fs::read_to_string(path).map_err(|err| format!("Failed to read file: {}", err))
}
```

Es gibt noch weitere Optionen um Daten an das Frontend zu übergeben. Der sogenannte *State* [2]. Hierbei ist der Vorteil zentralisierte Datensammlung für das Frontend, welche über die gesamte Laufzeit der Applikation bestehen bleibt. Dieser State kann jederzeit vom Frontend aus aufgerufen werden. Ein Beispiel für State kann im Anhang gefunden werden 3.1.

Bei dem File Explorer wird unter anderem folgende Datenstruktur über so einen State bereitgestellt.

```
  pub struct MetaData {
    version: String,
    abs_file_path_buf: PathBuf,
    abs_file_path_for_settings_json: PathBuf,
    pub abs_folder_path_buf_for_templates: PathBuf,
    pub template_paths: Vec<PathBuf>,
    all_volumes_with_information: Vec<VolumeInformation>,
    current_running_os: String,
    current_cpu_architecture: String,
    user_home_dir: String
}
```

Das oben gezeigt `struct` ist zuständig für Basisinformationen, welche über die Applikation gespeichert werden oder über das System, auf welchem das Programm läuft. Dies wird zum Startup des Programmes initialisiert und kann mithilfe von Tauri für das Frontend bereit gestellt werden. Das oben gezeigte `struct` besitzt Typen, welche aus einem `PathBuf` hervorgehen. Diese können in `Typescript` nicht ohne weiteres realisiert werden. Ein Ansatz hierfür wäre das folgende Konstrukt, welches aber auch nicht automatisch generiert werden kann.

3 Anhang

3.1 State Beispiel

Rust Code:

```
use tauri::State;
struct MyState(String);
#[tauri::command]
fn get_value(state: State<'_, MyState>) -> String {
    state.0.clone()
}
fn main() {
    tauri::Builder::default()
        .manage(MyState("Hello from backend".to_string())) // Initial value
        .invoke_handler(tauri::generate_handler![get_value])
        .run(tauri::generate_context!())
        .expect("error while running tauri application");
}
```

Frontend in Typescript

```
'use client';
import { useEffect, useState } from 'react';
import { invoke } from '@tauri-apps/api/tauri';
export default function GetStateExample() {
  const [value, setValue] = useState<string>('');
  useEffect(() => {
    async function fetchValue() {
      const result = await invoke<string>('get_value');
      setValue(result);
    }
    fetchValue();
  }, []);
  return (
    <div>
      <h1>Backend State</h1>
      <p>{value}</p>
    </div>
  );
}
```

Literaturverzeichnis

[1] conaticus01. *I Made a FAST File Explorer (in Rust)*. https://www.youtube.com/watch?v=Z60f2g-COJY. YouTube video. Juni 2023.

[2] Tauri Project. *State Management*. Accessed: 2025-05-10. 2025. URL: https://v2.tauri.app/develop/state-management/.

[3] Tauri Project. *Tauri - Build Smaller, Faster, and More Secure Desktop Applications with a Web Frontend*. Accessed: 2025-05-10. 2025. URL: https://v2.tauri.app/.

Abbildungsverzeichnis

Eidesstaatliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst und dabei keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht. Die Arbeit wurde bisher weder gesamt noch in Teilen einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

.....	.....	.....	.....
Marco Brandt	Daniel Schatz	Sören Panten	Lauritz Wiebusch

Ort, den 11. Mai 2025