

Terminal-Config-Manager
Informatik für Anwendungsentwicklung
CHECK24 Tech Hub und Services GmbH

Adrian Schurz

3. November 2022

1 Projektantrag

Diese Seite wird später entfernt.

1 Projektantrag

Der folgende Projektantrag wurde um die Auflagen, welche in der Terminbestätigung genannt wurden, erweitert. Weiterhin hat sich der Name der Firma, ohne jegliche Änderungen des Arbeitsverhältnisses, in der Zwischenzeit geändert und wurde hier aktualisiert.

Abschlussprüfung im Beruf Fachinformatiker für Anwendungsentwicklung

Winter 2022/23

Antragsformular

Azubi-Nr.: 480513

Name: Adrian Schurz

Ausbildungsbetrieb / Praktikumsbetrieb	CHECK24 Tech Hub und Services GmbH
Projektbezeichnung	terminal-config-manager
Projektbeschreibung	<p>Motivation: Bei der Arbeit an Software-Anwendungen, welche eng mit einer Vielzahl anderer solcher Anwendungen über mehrere Umgebungen hinweg interagieren, ergaben sich zwei häufige Anwendungsfälle beim Umgang mit Konfigurationsdateien.</p> <ol style="list-style-type: none">1. Das Einsehen und Kontrollieren von Einträgen innerhalb dieser Dateien. Dabei gibt es eine große Zahl an über verschiedene Verzeichnisse verstreuten Dateien. Innerhalb einer Datei gibt es jeweils eine große Zahl an Einträgen, von denen aber oft nur wenige relevant sind.2. Das Ändern eines solchen Eintrags, um das Verhalten der zugehörigen Anwendung anzupassen. <p>Sowohl das Kontrollieren als auch das Ändern von beliebigen Einträgen soll schnell und in einer Anwendung möglich sein.</p> <p>Beispielszenario</p> <p>In der Continuous-Integration-Pipeline (CI) einer zentralen Anwendung schlagen unvermittelt Akzeptanztests fehl. Um unverzüglich die Ursache feststellen zu können, muss das lokale Setup, das in diesem Moment an die Entwicklung einer anderen Anwendung und für die Ausführung einer anderen Menge an Tests konfiguriert ist, umkonfiguriert werden.</p> <p>IST-Zustand:</p> <p>Um die Zielanwendung zu untersuchen sind viele Teilaufgaben zu erledigen, beispielsweise das Aktivieren detaillierter</p>

	<p>Lognachrichten, das Ändern der auszuführenden Unit-, Integration- und Systemtestsuites, das Ändern von Cache-Verhalten, das anpassen von Ziel-IPs anderer, an der Testausführung beteiligter Anwendungen. Jeder dieser Schritte ist wiederum mit mehreren Einzelschritten verbunden. Meist muss zu diesem Zweck eine passende IDE gestartet, die relevante Konfigurationsdatei ermittelt und zum richtigen Unterordner navigiert werden. Danach muss die Datei geöffnet, ihr Inhalt untersucht und der Zielwert geprüft und gegebenenfalls angepasst werden. In Summe sind sehr viele, repetitive Einzelaufgaben händisch zu erledigen. Hier bietet sich daher Potential zur Verbesserung.</p> <p>Soll-Zustand:</p> <p>Es existiert ein Kommandozeilenprogramm, das übersichtlich eine Liste anzeigt. Die Liste enthält pro Zeile einen Beschreibungstext und den aktuellen Wert innerhalb der Zieldatei. Das Programm erlaubt es, per Tastendruck Zeilen auszuwählen und den zugehörigen Wert aus einer Menge möglicher Werte auszuwählen. Wird ein neuer Wert ausgewählt, so wird die dazugehörige Konfigurationsdatei umgeschrieben. Sowohl die Zieldateien als auch die jeweiligen möglichen Werte sind konfigurierbar. Zusätzlich existieren ausführliche Unit- und Akzeptanztests.</p>		
Projektumfeld	<p>Die CHECK24 Vergleichsportal GmbH, CHECK24 Vergleichsportal Reise GmbH und CHECK24 Tech Hub und Services GmbH sind Betreiber von check24.de, einer Website, auf der verschiedene Produkte zum Vergleich angeboten werden. Der Auftraggeber CHECK24 Tech Hub und Services GmbH betreibt auf dieser Online-Plattform eine Vergleichsmöglichkeit von Pauschalreisen.</p> <p>Die Menge und Komplexität interner Anwendungen, die am Produktionsbetrieb und der Qualitätssicherung beteiligt sind, wächst stetig. Damit einher gehen komplexere Interaktionen und Konfigurationsmöglichkeiten, die häufige Fehlerquellen im Betrieb und während der Entwicklung darstellen. Je einfacher diese Konfiguration möglich ist, desto schneller kann während der Fehlersuche und der Entwicklung gearbeitet werden.</p>		
Projektphasen (einschließlich Zeitplanung)	<table border="1"> <thead> <tr> <th>Phase</th><th>Dauer (h)</th></tr> </thead> </table>	Phase	Dauer (h)
Phase	Dauer (h)		

	Konzeption	10
	Wahl des Techstacks	3
	Einrichtung Entwicklungsumgebung	2
	Implementierung	30
	Qualitätssicherung	15
	Dokumentation	10
	Gesamt	70
Dokumentation zur Projektarbeit (nicht selbstständig erstellte Dokumente sind zu kennzeichnen)	<p>Das Projekt wird ausführlich dokumentiert. Dazu gehören:</p> <ul style="list-style-type: none"> • Eine ausführliche, aus Quellcodekommentaren generierte Dokumentation der einzelnen Programmmodule im HTML-Format • Hilfetexte, die vom Programm selbst bei Bedarf ausgegeben werden • README-Dateien, welche das Aufsetzen des Projekts, die Kompilierung und Ausführung der Unit- und Akzeptanztests beschreiben • Eine Projektdokumentation im PDF-Format, welches eine IST-Analyse, die Anforderungen an die Software, die Projektziele, die Zeitplanung, den Projektverlauf und das Pflichtenheft beinhaltet. • Die Planung des Projekts • Die Umsetzung des Projekts • Eine Ergebnisdiskussion 	
Bearbeitungsdauer von	1.10.2022	
Bearbeitungsdauer bis	14.11.2022	
Präsentationsmittel	Laptop	
Overheadprojektor	vorhanden	
Projektionsbildschirm (als Beamer nutzbar)	vorhanden	
Andere Präsentationsmittel (sind vom Prüfling mitzubringen)	Laptop	

Themenbetreuer	Jessica Parth Falk Döring
----------------	------------------------------

Themenbestätigung

Folgendes wurde aus Ihrem Schreiben vom 23.09.2022, der Zustimmung des betrieblichen Auftrages mit Auflagen, übernommen:

Thema bestätigt	
Mit Auflagen bestätigt	X Projektdokumentation umfasst auch: Planung, Umsetzung, Ergebnisdokumentation -> Planung entsprechend anpassen
Grund Ablehnung	

2 Nachweisblatt

Inhaltsverzeichnis

1 Projektantrag

2 Nachweisblatt

3 Textteil - vorläufiger Titel 1

3.1	Problemstellung	I
3.2	Technologie	2
3.3	Projektplanung	3
3.4	Ressourcenplanung	3
3.5	Projektorganisation	4
3.6	Design	5
3.7	Ergebnisdiskussion	5

4 Anlagen 6

5 Kundendokumentation 11

5.1	Beschreibung	II
5.2	Installation	II
5.3	Konfiguration	12
5.4	Benutzung	15
5.5	Problembehandlung	16

Literatur 17

Glossar 18

3 Textteil - vorläufiger Titel

3.1 Problemstellung

Der Bedarf für die im Rahmen dieser Projektarbeit erstellte Softwarelösung ergab sich bei der Arbeit an Software-Anwendungen, welche eng mit einer Vielzahl anderer solcher Anwendungen über mehrere Umgebungen hinweg interagieren.

Der Großteil dieser Anwendungen besitzt weitläufige Konfigurationsmöglichkeiten welche ihren Betrieb in verschiedensten Szenarien steuern. Beispiele für Konfigurationsmöglichkeiten und deren Ausprägungen sind:

- Das Loggingverhalten der Anwendung
 - Logging gegen die Logverarbeitungssoftware der Produktionsumgebung
 - Logging gegen eine lokale Instanz der Logverarbeitungssoftware
 - Logging auf das Dateisystem
 - Logging mit verschiedenen Logleveln
- Die Zieldatenbank der Anwendung
 - Datenbank der Produktionsumgebung
 - Datenbank der Testumgebung
 - lokale Datenbank
- Die Ausführung von Softwaretests
 - Ausführen von ausschließlich Unittests
 - Ausführen von Akzeptanztests
 - Ausführen der Gesamtheit der Tests
 - Anpassung der Ziel-IP einer weiteren, für die Testausführung notwendigen Anwendung
- uvm.

Der Kontext der Arbeit an der Software wechselt regelmäßig zwischen Entwicklung und dem Suchen bzw. Nachvollziehen von potentiellen Softwarefehlern, welche im Produktions- oder Testbetrieb auftreten. Um dabei das beobachtete Verhalten der Anwendung korrekt zu interpretieren sind u.a. zwei Arbeitsschritte häufig zu erledigen:

- **Prüfen** der aktuellen Konfiguration

- **Anpassen** der aktuellen Konfiguration

Die Anzahl der an jedem Einzelfall beteiligten Anwendungen und die Anzahl der Konfigurationsdateien pro Anwendungen führen dazu, dass jeweils viele verschiedene und weit über das Dateisystem verstreute Dateien relevant sind. Pro Datei und Einzelfall sind folgende Arbeitsschritte zu erledigen:

- Starten der zur Anwendung gehörigen IDE
- Ermitteln der Konfigurationsdatei
- Navigation im Verzeichnisbaum
- Öffnen der Datei
- Finden des relevanten Eintrags in einer mitunter sehr langen Textdatei
- Ermitteln der möglichen Zielwerte
- Ändern des Eintrags
- Speichern der Datei

Diese Einzelschritte, multipliziert mit der Anzahl an Dateien, stellt eine Menge an repetitiven Handlungen dar die, wenn sie erleichtert würden, Zeit und Konzentrationsvermögen einsparen können.

Ziel dieser Projektarbeit ist es ein Programm zu entwerfen und zu implementieren welches die beschriebenen Arbeitsschritte bestmöglich vereinfacht und beschleunigt.

3.2 Technologie

Für dieses Projekt bieten sich grundsätzlich alle gängigen Programmiersprachen an. Während der Umsetzung soll ausgewählten Aspekten der Softwareentwicklung gesonderte Aufmerksamkeit zukommen.

Korrektheit und Laufzeitstabilität Es soll auf technischem Weg zum Einen sichergestellt werden, dass sich das Programm zu jedem Zeitpunkt erwartungsgemäß und korrekt verhält und zum Anderen, dass Fehlerzustände zur Laufzeit so weit wie möglich ausgeschlossen werden.

Als hauptsächliche Wege dies zu erreichen werden folgende Ansätze gewählt:

- Wahl einer Programmiersprache mit strenger Typisierung
- Wahl einer kompilierten Programmiersprache mit vergleichsweise starken Garantien zum Laufzeitverhalten
- Einbeziehung von Property-Based-Testing [5] in das Konzept der Softwaretests

Ausführliche, vom Quellcode abgeleitete Moduldokumentation Neben der Projektdokumentation soll eine Dokumentation der einzelnen Softwaremodule entstehen. Um dem Problem zu begegnen, dass Dokumentation und Quellcode im Laufe der Entwicklung auseinanderlaufen soll die Moduldokumentation direkt aus den Quellcodekommentaren generierbar sein.

Auswahl Als Programmiersprache und Build-System wurden auf Basis der obengenannten Ziele Haskell [4] und Stack [7] gewählt.

Stack bietet neben seiner Hauptaufgabe die Software-Abhängigkeiten des Projekts zu verwalten und den Buildvorgang zu steuern die Möglichkeit Moduldokumentation in einer Vielzahl von Formaten zu generieren während Haskell eine typsichere, kompilierte Programmiersprache mit Unterstützung für Property-Based-Testing darstellt.

3.3 Projektplanung

Die geplante Umsetzungsdauer beträgt ca. 70 Stunden und ist folgendermaßen gegliedert.

Projektphase	geplante Zeit (h)
Konzeption	10
Technologiewahl	3
Einrichtung	2
Implementierung	30
Qualitätssicherung	15
Dokumentation	10
Gesamt	70

3.4 Ressourcenplanung

Hier werden alle zur Fertigstellung des Projekts verwendeten Ressourcen, sowohl Hardware- als auch Software- und Personalressourcen, aufgelistet.

- Hardware
 - Lenovo Thinkpad P52
- Software
 - Arch Linux (Betriebssystem)
 - Stack (Buildsystem)
 - GHC (Compiler)
 - Git (Versionskontrolle)
 - Haddock (Generierung der Moduldokumentation)
 - Graphmod (Graphgenerierung, Modulabhängigkeiten)
 - Visual Studio Code (Codeeditor)
 - T_EX-Live (Projektdokumentation)
- Personal
 - Softwareentwickler zur Umsetzung
 - Softwareentwickler zur Qualitätskontrolle

3.5 Projektorganisation

Die Hauptelemente der Verzeichnisstruktur des Projekts sind wie folgt organisiert.

```
\src
\test
\doc
\distribution
\bin
\tool
package.yaml
README.md
```

Die Verzeichnisse enthalten dabei in der angezeigten Reihenfolge

- Quellcodes des Programms
- Quellcode der Unit- und Integrationstests
- generierte Moduldokumentation und Projektdokumentation

- Skripte um Softwarepakete für verschiedene Betriebssysteme zu erstellen
- kompilierte, ausführbare Dateien
- Hilfsskripte, z.B. zur Generierung des Modulabhängigkeitsgraph

Die Datei `package.yaml` enthält die für Stack notwendigen Metadaten, beispielsweise Deklarationen der verwendeten Softwarebibliotheken und Konfiguration des Buildprozesses.

Die Datei `README.md` enthält eine kurze Beschreibung des Programms und eine Nutzungsanleitung zusammen mit offenen Aufgaben und Zusatzinformationen.

3.6 Design

3.7 Ergebnisdiskussion

Funktionalität Die Hauptfunktionalität des Programms wurde erfolgreich umgesetzt und es wird von mir selbst bereits genutzt. Das Programm arbeitet seither wie erwartet, ist konfigurierbar und informiert mit lesbaren Nachrichten im Falle eines Fehlers. Ein weiteres Feature wäre jedoch wünschenswert, die Fähigkeit des Programms mit schreibgeschützten Zieldateien umgehen zu können und in diesem Fall eine sudo-Passwortabfrage auszulösen. Letzteres ist nicht Teil der ursprünglichen Anforderungen und wird daher erst zukünftig umgesetzt.

Domain Driven Design Das Ziel die Abhängigkeiten der einzelnen Softwaremodule untereinander einem mit dem Domain-Driven-Design kompatiblen Schema folgend zu organisieren ist mit einer Ausnahme geglückt. Im Anhang, Abb. 3, sind Module der Applikationsebene abhängig von Modulen des User-Interface. Das verwendete GUI-Framework, Brick, führt durch sein Interface allerdings zu einem Muster bei dem diese Abhängigkeit umgekehrt ist. Letzteres wird deutlich beim Vergleich des Schemas mit den tatsächlichen Modulabhängigkeiten (siehe Abb. 4 im Anhang). Dieser Umstand stellt für den Moment eine vernachlässigbare Designschwäche dar die ohne weitere Konsequenz ist. Aus diesem Grund erhielt die Aufgabe dies zu beheben eine niedrige Priorität und steht vorerst noch aus. Alle sonstigen Module halten das angestrebte Schema ein.

Moduldokumentation Das Ziel die Moduldokumentation ständig, während des Buildprozesses, aktuell zu halten ist teilweise geglückt. Im Projektverzeichnis liegt die ausführliche Moduldokumentation einschließlich jener der eingebundenen Softwarebibliotheken im HTML-Format vor. Auszüge davon sind in den Abbildungen 1 und 2 dargestellt. Allerdings gibt es seit wenigen Tagen bei neueren Versionen des Buildsystems Stack [7] und des Generierungstools Haddock [3] eine Inkompatibilität.

Bis diese in diesen Projekten behoben und veröffentlicht ist, ist die in diesem Projekt hinterlegte Moduldokumentation nicht aktuell. Es ist zu erwarten, dass dieses Problem in naher Zukunft seitens der Entwickler der beiden Tools behoben wird.

Testabdeckung Die Abdeckung der Funktionalität des Programms auf Unittest-Ebene ist, besonders dank der Verwendung von Property-Based-Testing [5] zufriedenstellend.

Eines der angestrebten Ziele war jedoch neben ausführlichen Unittests ebenso ausführliche Akzeptanztests bereitzustellen. Dies ist trotz erheblichem Aufwand gescheitert. Das Programm ist eine Konsolenanwendung, welche auf Tasteneingaben reagiert. Das bedeutet, dass automatisierte Tests in einer kontrollierten Umgebung Terminal-Emulatoren starten und ihnen Tasteneingaben simulieren müssen um das Programm zu testen. Je nach der verwendeten grafischen Benutzeroberfläche des Betriebssystems unterscheiden sich die Ansätze dies zu erreichen jedoch stark. Es existieren mehr oder weniger gut gepflegte Open-Source-Softwaretools für die Teilaufgabe der Eingabeemulation. Verschiedene Kombinationen dieser Tools (xdotool vs ydotool) wurden mit verschiedenen Displayservern (xorg vs. wayland), Betriebssystemen (Ubuntu vs Arch), und Virtualisierungslösungen (virtualbox vs docker) evaluiert. Keine der Varianten führte zum Erfolg.

Bekannte Fehler Es sind zwei Bugs in Software-Abhängigkeiten des verwendeten GUI-Frameworks bekannt, welche mit geringer Häufigkeit das Rendering bzw. den Start des Programms beeinträchtigen. Diese Bugs sind dokumentiert (siehe [2] und [1]). Einer dieser Fehler führt dazu, dass der aktuelle Wert eines Eintrags weiß statt blau gerendert wird und ist schwer reproduzierbar (ix ca. pro 50-100 Programmstarts). Der andere Fehler führt zu einem Crash des Programms beim Start (ix ca. pro 30-50 Programmstarts).

4 Anlagen

Abbildung 1: Generierte Moduldokumentation, Hauptseite

terminal-config-manager-0.1.0.0
Quick Jump · Instances · Contents · Index

terminal-config-manager-0.1.0.0
Please see the README on GitHub at <https://github.com/githubuser/terminal-config-manager#readme>

Modules

- ▼ Application
 - [Application.App](#) Build and run a Brick app after loading the config file.
- ▼ Domain
 - [Domain.State](#) Type definitions concerning the application state.
 - [Domain.StateTransition](#) Expose functions to change the application state. This covers item and value selection.
- ▼ Infrastructure
 - [Infrastructure.Config](#) Expose a function which parses the contents of the config file and returns a list of items specified in that file.
 - [Infrastructure.FileModification](#) Expose a type representing target file content and a function to modify the content of a file.
 - [Infrastructure.Util](#) Expose a range of helper functions.
- ▼ UserInterface
 - [UserInterface.Input](#) Handle the keyboard user input.
 - [UserInterface.Render](#) Expose a function to draw an application state to the screen and two different styling, one for the description text and one for the value

Produced by [Haddock](#) version 2.26.0

Abbildung 2: Generierte Moduldokumentation, Domain.State

Domain.State

Documentation

data AppState# Source

The state of the application. It will guarantee that the list of items contains at least one element and keep track of a cursor position.

Constructors

MkAppState (NonEmptyCursor ConfigItem)

Instances

▷ Show AppState# Source

▷ Eq AppState# Source

type NextAppState = EventM ResourceName (Next AppState)# Source

When reacting to an event the result needs to be of type EventM. Though not entirely accurate the synonym here is trying to communicate that, in essence, a new state of the application is created.

data ResourceName# Source

An dummy type used only as a parameter of EventM

Instances

▷ Show ResourceName# Source

▷ Eq ResourceName# Source

▷ Ord ResourceName# Source

Synopsis

8

Abbildung 3: Beispielschema für Modulabhängigkeiten beim Domain-Driven-Design

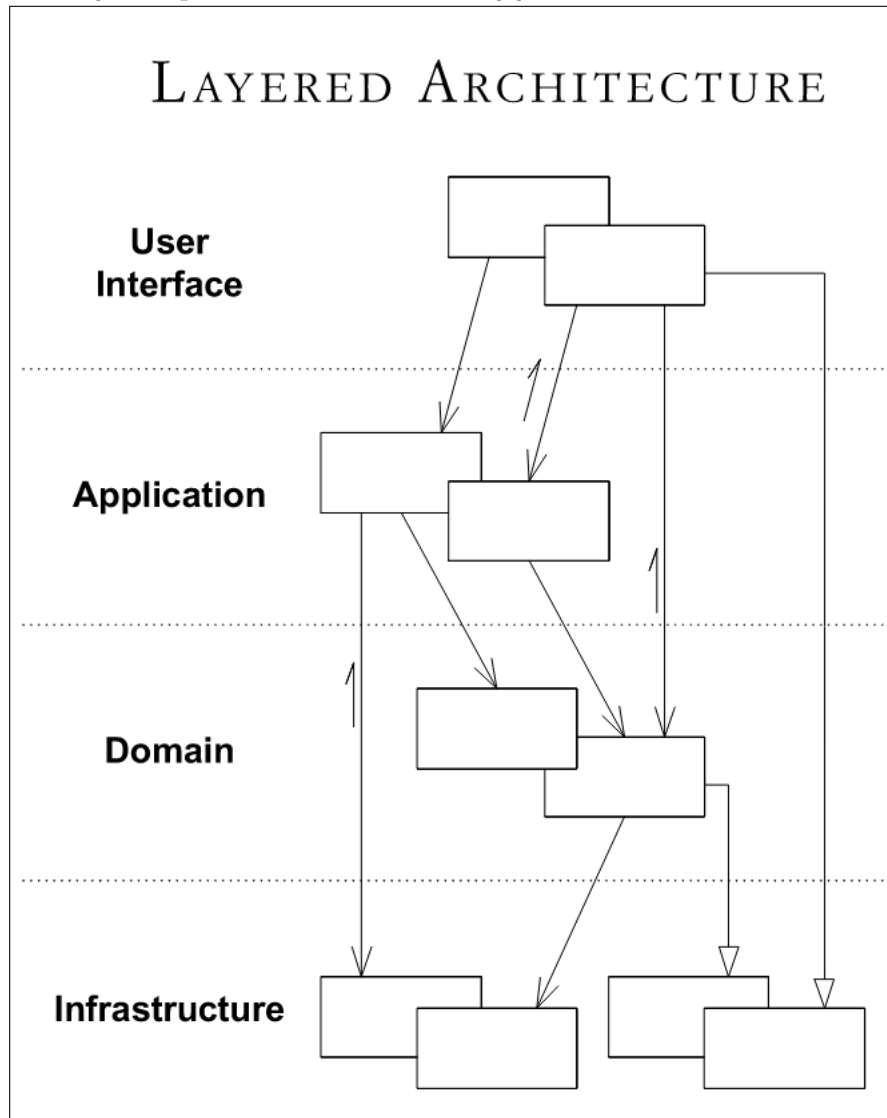
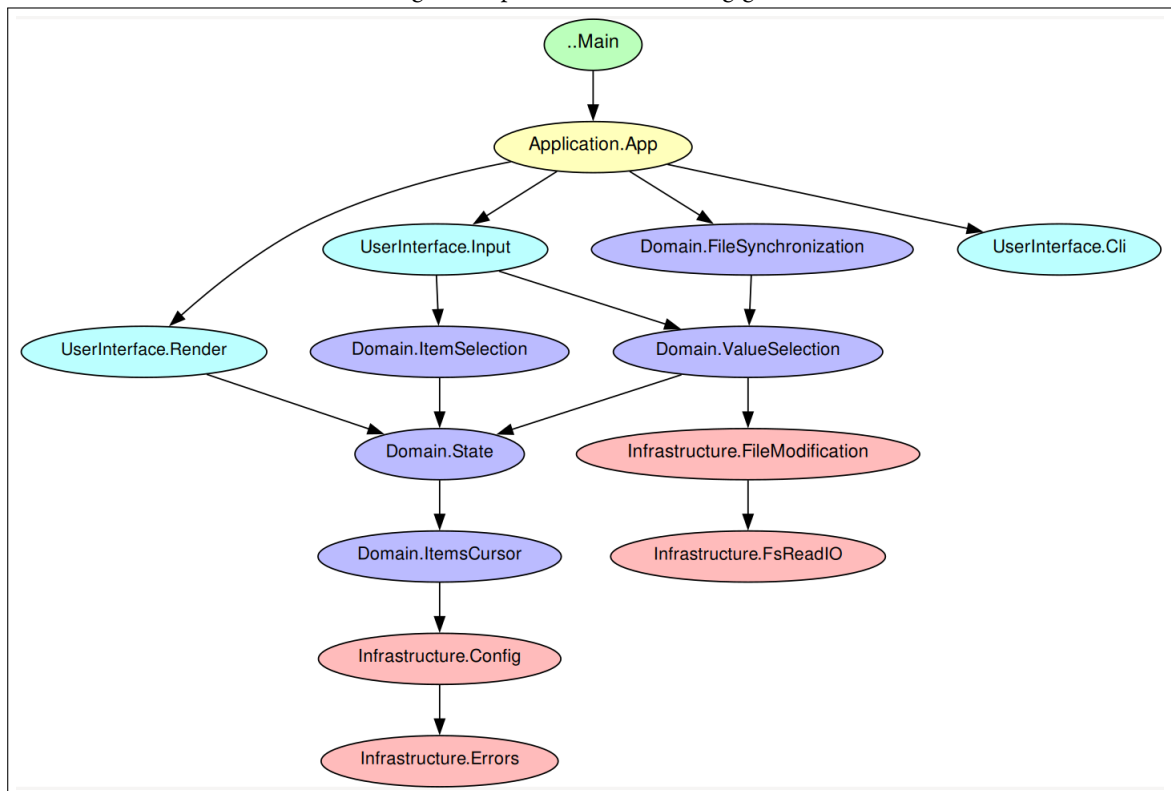


Abbildung 4: Graph der Modulabhängigkeiten



5 Kundendokumentation

5.1 Beschreibung

Terminal-Config-Manager ist ein Linux-Programm mit welchem Textpassagen innerhalb mehrerer Dateien schnell zwischen einer Reihe von vorkonfigurierten Textpassagen umgeschalten werden können. Der Hauptanwendungsfall ist die effiziente Manipulation von Konfigurationsdateien von Softwareanwendungen die häufig angepasst werden müssen.

5.2 Installation

Es wurden vorkonfigurierte Pakete für sowohl ArchLinux-basierte als auch Debian-basierte Betriebssysteme bereitgestellt. Alternativ kann das Programm auch manuell installiert werden.

Arch-Linux, via PKGBUILD Datei und pacman

Im Projektverzeichnis unter

```
/distribution/arch/PKGBUILD
```

befindet sich eine Spezifikationsdatei anhand derer das Softwarepaket erstellt und anschließend installiert werden kann:

```
cd distribution/arch
makepkg
pacman -U terminal-config-manager-1.0.0-1-x86_64.pkg.tar.zst
```

Die Deinstallation erfolgt mittels

```
pacman -R terminal-config-manager
```

Debian, via .deb Datei und dpkg bzw. apt

Im Projektverzeichnis unter

```
/distribution/debian/terminal-config-manager.deb
```

befindet sich ein Softwarepaket, das mittels dpkg oder apt direkt installiert werden kann.

```
cd distribution/debian
dpkg --install ./terminal-config-manager.deb
# apt install ./terminal-config-manager.deb
```

Die Deinstallation erfolgt mittels

```
dpkg --remove terminal-config-manager  
# apt remove terminal-config-manager
```

Alternative, ohne Paketmanager

Wenn das Programm nicht vom systemeigenen Paketmanager verwaltet werden soll, dann kann es manuell kompiliert und in einem passenden Verzeichnis abgelegt werden.

Voraussetzung hierfür ist, dass das Programm `stack` auf dem System installiert ist.

Im Projektverzeichnis wird mit

```
stack build --test --copy-bins
```

das Programm kompiliert, die Testsuite ausgeführt und die ausführbare Datei im Projektverzeichnis unter

```
bin/terminal-config-manager
```

abgelegt. Anschließend kann das Programm in ein Verzeichnis kopiert werden, das in die Systempfadliste eingetragen ist, beispielsweise

```
cp bin/terminal-config-manager ~/.local/bin
```

Die Deinstallation erfolgt mittels

```
rm ~/.local/bin/terminal-config-manager  
rm <Konfigurationsdateipfad>
```

5.3 Konfiguration

Die Zielfile und -textpassagen müssen vor Ausführung des Programms über eine Datei im YAML-Format [8] konfiguriert werden.

Verzeichnis Das Programm erwartet, dass sich eine solche Datei in einem der folgenden Verzeichnisse befindet. Die Reihenfolge entspricht der absteigenden Priorität beim Vorhandensein mehrerer Konfigurationsdateien:

1. `./config.yaml`

Abbildung 5: Beispielaufbau der Konfigurationsdatei

```
config_lines_to_manage:
- title: Beispieltitel 1
  path: /home/alice/zieldatei.conf
  pattern: "'statspush_enabled' => {{value}},"
  targetValue: "true"
  possibleValues:
    - "true"
    - "false"

- title: Beispieltitel 2
  path: /home/alice/verzeichnis/weitere-zieldatei.txt
  pattern: "SOFTWARE_ENV={{value}}"
  targetValue: production
  possibleValues:
    - testing
    - staging
    - production
    - local

- ...
```

2. `${HOME}/.config/terminal-config-manager/config.yaml` (**empfohlen**)
3. `${HOME}/.terminal-config-manager.yaml`

Der Dateipfad 1 bezeichnet den Ort der ausführbaren Datei selbst und sollte nur zu Debugging- oder Entwicklungszecken genutzt werden. Die Pfade 2 und 3 sind gängige Ablageorte für nutzerspezifische Konfigurationsdateien unter Linux und sind für die normale Nutzung geeignet.

Aufbau In Abb. 5 ist der Aufbau der Konfigurationsdatei illustriert.

- `config_lines_to_manage:`

Dies ist das äußere Element der Konfigurationsdatei und **muss** vorhanden sein. Innerhalb dessen befindet sich eine Liste von Einträgen. Jeder Eintrag gehört zu genau einer Textpassage, die mithilfe des Programms gezielt verändert werden soll.

Jeder Eintrag hat folgende Unterelemente:

- `title: Beispieltitel 1`

Dies ist ein Titel der frei gewählt werden kann und vom Programm angezeigt wird. Idealerweise wird dafür eine sehr kurze Beschreibung des von der Ziel-Textpassage gesteuerten Anwendungsverhaltens benutzt.

- `path: /home/alice/zieldatei.conf`

Hier wird der absolute Dateisystempfad der Zieldatei angegeben innerhalb derer Textpassagen verändert werden sollen.

- `pattern: "'statspush_enabled' => {{value}},"`

Dieses Feld enthält ein Zielmuster. Mithilfe dieses Musters das einen Platzhalter enthält wird vom Programm der genaue Ort der Ziel-Textpassage identifiziert. Das Zielmuster sollte sowohl den Platzhalter als auch an die Ziel-Textpassage angrenzenden Text enthalten. **Wenn die Kombination aus Platzhalter und angrenzendem Text die Ziel-Textpassage nicht eindeutig eingrenzt, dann wird vom Programm ausschließlich die erste übereinstimmende Textpassage modifiziert.**

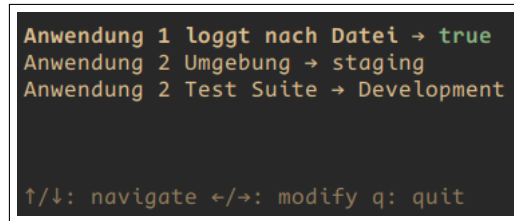
- `targetValue: "true"`

Der aktuelle Wert der Ziel-Textpassage. **Da das Programm in neueren Versionen beim Start den aktuellen Wert selbstständig ausliest, wird dieser Konfigurationswert in einer der folgenden Versionen entfernt werden. Aktuell muss er jedoch weiterhin in der Konfigurationsdatei angegeben werden.**

- `possibleValues:`
 - `"true"`
 - `"false"`

Dies ist die Liste der Werte die mithilfe des Programms ausgewählt und anstelle der Ziel-Textpassage in die Zieldatei geschrieben werden können. **Wenn sich der aktuelle Wert nicht in dieser Liste befindet, dann kann mit dem Programm auch nicht auf den ursprünglichen Wert zurückgewechselt werden.**

Abbildung 6: Ansicht nach Programmstart



```
Anwendung 1 loggt nach Datei → true
Anwendung 2 Umgebung → staging
Anwendung 2 Test Suite → Development

↑/↓: navigate ←/→: modify q: quit
```

5.4 Benutzung

Start Das Programm wird nach erfolgreicher Installation mit dem Befehl

```
terminal-config-manager
```

von der Kommandozeile aus gestartet. Für jeden Eintrag in der Konfigurationsdatei zeigt das Programm eine Zeile an.

Ansicht Abb. 6 zeigt eine typische Ansicht direkt nach dem Start des Programms. Die ersten drei Zeilen repräsentieren jeweils einen Eintrag in der Konfigurationsdatei und somit eine Ziel-Textpassage mit ihrem aktuellen Wert. Jede dieser Zeilen besteht aus dem in der Konfigurationsdatei vergebenen Titel des Eintrags, einem Pfeil und dem aktuellen Wert der Ziel-Textpassage. Die **aktuell ausgewählte Zeile** ist fett gedruckt während der **aktuelle Wert** der ausgewählten Zeile blau dargestellt wird. Die genaue Darstellung ist dabei vom verwendeten Terminal-Emulator und dessen Einstellungen bezüglich Schriftart und Farbwerten abhängig.

Die ausgegraute Zeile am unteren Rand zeigt zu jeder Zeit in Kurzform die verfügbaren Kommandos und die davon ausgelösten Aktionen.

Aktionen Die Hauptfunktionen werden über die vier Pfeiltasten gesteuert. Die Pfeiltasten hoch bzw. runter bewegen die Zeilenmarkierung nach oben bzw. unten. Die Pfeiltasten links bzw. rechts schalten den zur markierten Zeile gehörigen Wert weiter zum nächsten Wert aus der konfigurierten Liste der möglichen Werte (siehe Kapitel 5.3 - Konfiguration). Beim Umschalten eines Werts wird die zugehörige Zielfeile entsprechend modifiziert.

Mit einem Tastendruck auf q kann das Programm jederzeit beendet werden und zur Kommandozeile zurückgekehrt werden.

5.5 Problembehandlung

Fehlende Konfigurationsdatei Wenn beim Programmstart der Fehler

```
Error: No config file found at any of the search paths: ...
```

auftritt, dann bedeutet das, dass bei der Suche nach Konfigurationsdateien an keinem der angegebenen Pfade eine Datei gefunden wurde.

Lösung Es wird wie in Kapitel 5.3 (Konfiguration) beschrieben eine Konfigurationsdatei an einem der validen Dateipfade angelegt. Im Dateisystempfad unter

```
/usr/share/terminal-config-manager/config.yaml
```

befindet sich eine Beispielskonfigurationsdatei, welche als Vorlage genutzt werden kann:

```
mkdir ~/.config/terminal-config-manager  
cp /usr/share/terminal-config-manager/config.yaml \\  
~/.config/terminal-config-manager
```

Falsches Konfigurationsdateiformat Wenn beim Programmstart ein Fehler ähnlich

```
An error occurred while parsing the configuration file.  
The details are: ...
```

auftritt, dann bedeutet das, dass die erste vom Programm gefundene Konfigurationsdatei entweder nicht dem YAML-Format [8] entspricht und/oder fehlende Elemente aufweist.

Lösung Die Fehlermeldung wird weitere Detailinformationen enthalten wie beispielsweise:

```
The top level of the config file  
should be an object named 'config_lines_to_manage'
```

anhand derer sich das Problem identifizieren lässt. Im Zweifelsfall muss dem Kapitel 5.3 (Konfiguration) bzw. der Problembehandlung für fehlende Konfigurationsdateien in Kapitel 5.5 folgend eine valide Konfigurationsdatei per Hand angelegt werden.

Fehlende Dateizugriffsrechte Wenn bei der Auswahl eines neuen Werts der Fehler


```
terminal-config-manager: <Zielfeldpfad>: withFile:
permission denied
```

auftritt, dann bedeutet das, dass dem aktuellen Linux-Nutzer die nötigen Zugriffsrechte fehlen um die Zielfeld zu modifizieren.

Lösung Es muss sichergestellt werden, dass alle in der Konfigurationsdatei definierten Zielfeldern vom aktuellen Nutzer modifiziert werden können. Im häufigsten Fall ist der aktuelle Nutzer nicht als owner der Datei eingetragen:

- Wenn dies angebracht ist, dann kann der Nutzer der Datei geändert werden:

```
chown <Nutzer> <Zielfeldpfad>
```

- Wenn dies angebracht ist, dann können die Schreibrechte der Zielfeld angepasst werden:

```
chmod o+w <Zielfeldpfad>
```

Wenn keine der beiden oben genannten Optionen anwendbar ist, dann ist diese Zielfeld nicht für die Modifizierung durch das Programm geeignet.

Literatur

- [1] *Github Issue eines Bugs der selten zu einem Rendering-Problem führt*. URL: <https://github.com/judah/terminfo/issues/47>.
- [2] *Github Issue eines Bugs der zum Crash zum Programmstart führen kann*. URL: <https://github.com/jtclougherty/vty/issues>.
- [3] *Haddock Project Homepage*. URL: <https://haskell-haddock.readthedocs.io/en/latest/>.
- [4] *Haskell Project Homepage*. URL: <https://www.haskell.org/>.
- [5] Fred Herbert. *The Pragmatic Programmers: Property-Based Testing with PropEr, Erlang, and Elixir*. 1. Aufl. Pragmatic Bookshelf, 2019.
- [6] *Jinja2 Project Homepage*. URL: <https://jinja.palletsprojects.com/en/2.11.x>.
- [7] *Stack Project Homepage*. URL: <https://docs.haskellstack.org/en/stable/>.
- [8] *YAML Project Homepage*. URL: <https://yaml.org>.

Glossar

GUI Abkürzung für den Begriff graphical user interface. Zu deutsch: grafische Benutzeroberfläche. 1, 5, 6

Platzhalter Ein vordefinierter Text: das englische Wort `value` umgeben von doppelten geschweiften Klammern: `{{value}}` Das Format des Platzhalters ist an die Template-Engine `Template-Engine Jinja2` [6] angelehnt. Er ist Teil des Zielmusters das zusätzlich Text vor und nach der Ziel-Textpassage enthält. Er markiert den Ort der Ziel-Textpassage relativ zum Zielmuster innerhalb der Zielfeile. 1, 14, 18

Property-Based-Testing Property-Based-Testing bezeichnet eine spezielle Strategie Softwaretests zu formulieren. Statt einer Reihe explizit ausformulierter Tests wird eine Regel formuliert, welche das zu testende Programm für automatisch generierte Eingabewerte einhalten muss. Die in großer Zahl automatisch generierten Testfälle decken mit höherer Wahrscheinlichkeit Grenzfälle durch z.B. besonders große oder abwegige Eingabewerte ab. 1, 3, 6

Rendering Im Kontext dieses Programms beschreibt der Begriff Rendering den Prozess der Wandlung eines Zustands des Programms in eine auf dem Bildschirm darstellbare Form, in diesem Fall Text. 1, 6

Textpassage Ein Stück Text aus einer Textfeile. Dabei wird sich meist auf eine in der Konfigurationsfeile spezifizierte Zielfeile des Programms bezogen. 1, 11, 13–15, 18

Zielmuster Ein Element innerhalb der Konfigurationsfeile. Es bezeichnet ein Stück Text das einen speziellen Platzhalter, den Text `{{value}}`, enthält. Es wird vom Programm dazu genutzt den genauen Ort der Ziel-Textpassage innerhalb der Zielfeile zu identifizieren. Das Zielmuster sollte 1, 14, 18