OST
Ostschweizer
Fachhochschule

Bachelorarbeit:

Schnelle und patientengerechte Visualisierung von 24h Aktivitätsprotokollen

Studiengang Informatik
OST – Ostschweizer Fachhochschule
Campus St. Gallen

Frühjahrsemester 2025

Abstract

In der klinischen Praxis und Gesundheitsforschung gewinnt die objektive Erfassung von Bewegungs- und Schlafverhalten durch Actigraphie zunehmend an Bedeutung. Das am Kompetenzzentrum MOVE-IT der OST eingesetzte GENEActiv-System ermöglicht die kontinuierliche Dokumentation von 24-Stunden-Bewegungsmustern, jedoch erschwert der zeitaufwändige Analyseprozess und die komplexen Visualisierungen die praktische Anwendung.

Die vorliegende Bachelorarbeit adressiert diese Herausforderung durch die Entwicklung einer Desktop-Anwendung, welche den bestehenden Workflow grundlegend optimiert. Zum einen wurde der ursprüngliche R-Code zur Sensordatenanalyse durch Performance-Optimierungen beschleunigt. Zum anderen ersetzt ein intuitives User Interface den komplexen Prozess durch eine einheitliche Benutzerführung.

Die Implementierung erfolgte als plattformübergreifende .NET-Anwendung mit Avalonia UI, welche eine moderne MVVM-Architektur mit modularen Komponenten für verschiedene Sensortypen umsetzt. Der ursprüngliche R-Code wurde durch Performance-Optimierungen und Code-Bereinigung verbessert und in die .NET-Anwendung integriert. Die Datenvisualisierungen ermöglichen sowohl Fachkräften als auch Patienten eine verständliche Interpretation der Analyseergebnisse.

Management Summary

Ausgangslage und Problemstellung

Das Kompetenzzentrum MOVE-IT der OST nutzt GENEActiv-Sensoren zur objektiven Erfassung von Bewegungs- und Schlafverhalten in der klinischen Praxis. Der bestehende Workflow erfordert jedoch zeitaufwändige manuelle Prozesse: Die R-basierte Analyse ist ineffizient und benötigt erhebliche Verarbeitungszeit. Die Visualisierung erfolgt über komplexe R-Markdown-Dokumente, die in Word-Dateien konvertiert werden – ein umständlicher Prozess, der die praktische Anwendung im klinischen Alltag erheblich erschwert.

Zielsetzung und Lösungsansatz

Im Rahmen dieser Bachelorarbeit wurde eine benutzerfreundliche Desktop-Anwendung zur Optimierung des gesamten Analyseworkflows entwickelt. Die Hauptziele umfassten:

- **Performance-Optimierung**: Beschleunigung der R-basierten Datenanalyse durch gezielte Code-Optimierungen
- **Benutzerfreundlichkeit**: Entwicklung einer intuitiven Benutzeroberfläche zur Vereinfachung des Analyseprozesses
- **Moderne Visualisierung**: Ersatz der komplexen Word-Dokumente durch verständliche Datenvisualisierungen

Technische Umsetzung

Die Lösung wurde als plattformübergreifende .NET-Desktop-Anwendung mit Avalonia UI implementiert. Die Architektur folgt modernen Software-Engineering-Prinzipien:

- **MVVM-Pattern**: Saubere Trennung von UI-Logik und Geschäftslogik
- **Factory-Pattern**: Erweiterbare Architektur für zukünftige Sensortypen
- **R-Integration**: Einbindung der R-Analyse

Erzielte Verbesserungen

Performance-Optimierung

Durch systematische Code-Optimierungen konnte die Analysezeit reduziert werden:

- **Aktivitätsanalyse**: Reduktion von 13 Minuten 5 Sekunden auf 2 Minuten 46 Sekunden

- **Schlafanalyse**: Reduktion von 15 Minuten 40 Sekunden auf 2 Minuten 50 Sekunden

Die Optimierungen konzentrierten sich auf Code-Refactoring und die Entfernung redundanter Komponenten, die Vektorisierung der StepCounter-Funktion sowie die einmalige Berechnung wiederverwendbarer Datenstrukturen.

Benutzerfreundlichkeit

Die Applikation zeichnet sich durch eine intuitive, lineare Benutzerführung aus, die Analysen in wenigen Klicks ermöglicht. Realistische Zeitschätzungen und kontinuierliches Feedback während der Verarbeitung sorgen für Transparenz, während benutzerfreundliche Fehlermeldungen mit Lösungsvorschlägen die Bedienung erleichtern.

Die Visualisierungen bieten übersichtliche Darstellungen von Aktivitäts- und Schlafdaten mit Vergleichsmöglichkeiten mehrerer Datensätze. Barrierefreie Farbpaletten sowie Export-Funktionalitäten für PDF-Berichte und Rohdaten vervollständigen die Applikation.

Nachhaltigkeit und Ausblick

Die Anwendung wurde mit Fokus auf langfristige Wartbarkeit entwickelt:

- **Modulare Architektur**: Einfache Erweiterung um neue Sensortypen
- **Strukturierte Code-Organisation**: Klare Zerlegung der Verantwortlichkeiten in einer GitHub-Organisation mit separaten Repositories
- **Testabdeckung**: Hohe Codequalität durch systematische Unit- und Integrationstests

Inhaltsverzeichnis

Kapitel 1

Ausgangslage / Problemstellung / Stand der Technik

1.1 Einleitung

In der klinischen Praxis und Gesundheitsforschung gewinnt die objektive Erfassung und Analyse von Bewegungs- und Schlafverhalten zunehmend an Bedeutung. Die Actigraphie als nicht-invasives Verfahren ermöglicht die kontinuierliche Dokumentation dieser Verhaltensmuster über mehrere Tage oder Wochen im natürlichen Umfeld der Patienten. Diese Daten liefern wertvolle Einblicke in 24-Stunden-Bewegungsmuster, die als wichtiger Indikator für die physische und mentale Gesundheit anerkannt werden.

1.2 Das GENEActiv-System

Im Rahmen der vorliegenden Arbeit wird mit dem GENEActiv-System gearbeitet, das aus hochpräzisen Raw Data Accelerometer-Geräten besteht. Der Hersteller stellt Open-Source-Bibliotheken in der Programmiersprache R zur Verfügung, welche die Interpretation der erhobenen Beschleunigungswerte ermöglichen und diese in klinisch relevante Aktivitäts- und Schlafparameter transformieren.

1.3 Problemstellung

Trotz des hohen Potenzials der Actigraphie für personalisierte Interventionen in Bereichen wie Prävention, Rehabilitation und Behandlung chronischer Erkrankungen stehen die Fachkräfte am Kompetenzzentrum für Motor-Cognitive Learning and Sport (MOVE-IT) der OST vor Herausforderungen bei der Datenverarbeitung mit dem GENEActiv-System.

Der aktuelle Workflow vom Datenimport bis zur Visualisierung ist durch mehrstufige, zeitaufwändige Arbeitsschritte erschwert. Die Extraktion der .bin-Dateien vom GENEActiv-Sensor kann mit der bereitgestellten Software bis zu 15 Minuten pro Datei beanspruchen, während die anschliessende Analyse mittels des vom Hersteller Activinsights bereitgestellten R-Codes mehrere Stunden in Anspruch nimmt. Diese Ineffizienz stellt im klinischen Alltag der betroffenen Therapeuten und Mediziner eine bedeutsame Hürde zur alltäglichen Benützung dar.

Problematisch ist ebenfalls die gegenwärtige Visualisierungsmethodik des GENEActiv-Systems, bei der Analyseergebnisse ohne klinisch relevante Priorisierung oder intuitive Darstellung in

R-Markdown-Dateien erzeugt und anschliessend in Word-Dokumente konvertiert werden. Dies erschwert die effiziente Interpretation und eine patientengerechte Darstellung der Bewegungs- und Schlafdaten erheblich. Eine fokussierte, auf wesentliche Parameter der 24-Stunden-Aktivitätsprotokolle konzentrierte und verständliche Visualisierung fehlt im bestehenden System.

Kapitel 2

Aufgabenstellung / Ziel der Arbeit

2.1 Beschleunigung des Analyseprozesses

Ein zentrales Ziel dieser Bachelorarbeit besteht in der Optimierung des Analyseprozesses für die GENEActiv-Daten. Der vom Hersteller Activinsights bereitgestellte R-Code führt derzeit separate Analysen für Bewegungs- und Schlafdaten durch, was mehrere Stunden Verarbeitungszeit in Anspruch nimmt. Im Rahmen dieser Arbeit soll eine Lösung konzipiert werden, die die Effizienz des Analyseprozesses verbessert. Eine Reduktion der Analysezeit würde es dem Personal am MOVE-IT ermöglichen, Ergebnisse zeitnah zu generieren und effektiver in die Patientenbetreuung zu integrieren.

2.2 Entwicklung einer patienten- und therapeutengerechten Visualisierung

Das zweite Ziel besteht in der Entwicklung eines intuitiven und klinisch relevanten Visualisierungskonzepts. Die bisherige Darstellungsmethodik, in Form von Word-Dokumenten, erfüllt nicht die Anforderungen an eine effiziente Informationsvermittlung im klinischen Kontext.

Die entwickelten Visualisierungen sollen anhand realer Nutzungsszenarien mit Personal vom MOVE-IT evaluiert werden, um ihre Praxistauglichkeit sicherzustellen. Besonderes Augenmerk liegt dabei auf der Benutzerfreundlichkeit und einer verbesserten Darstellung.

2.3 Integrierter Gesamtansatz

Die beiden Hauptziele ergänzen sich zu einem integrierten Gesamtansatz, der den Analyseprozess optimiert und die Visualisierung der Daten vereinfacht. Durch die Kombination der verbesserten Analyse und nutzerorientierter Visualisierung soll ein System entstehen, das den praktischen Anforderungen gerecht wird und die Potenziale der Actigraphie besser ausschöpft.

Die entwickelte Lösung wird mit einer Open-Source-Lizenz versehen, um eine Nutzung und Weiterentwicklung im universitären und klinischen Umfeld zu ermöglichen.

Kapitel 3

Systemkontext

3.1 Domain Model

Das Domain Model bildet die konzeptionelle Grundlage, auf der unsere Software basiert, ab. Es zeigt alle Personen und Entitäten die mit der Software interagieren, sowie ihre Beziehungen innerhalb des Systems. Ausserdem wird der Arbeitsablauf vom Tragen des Sensors durch den Patienten bis zur Auswertung der Daten durch Fachpersonen dargestellt.

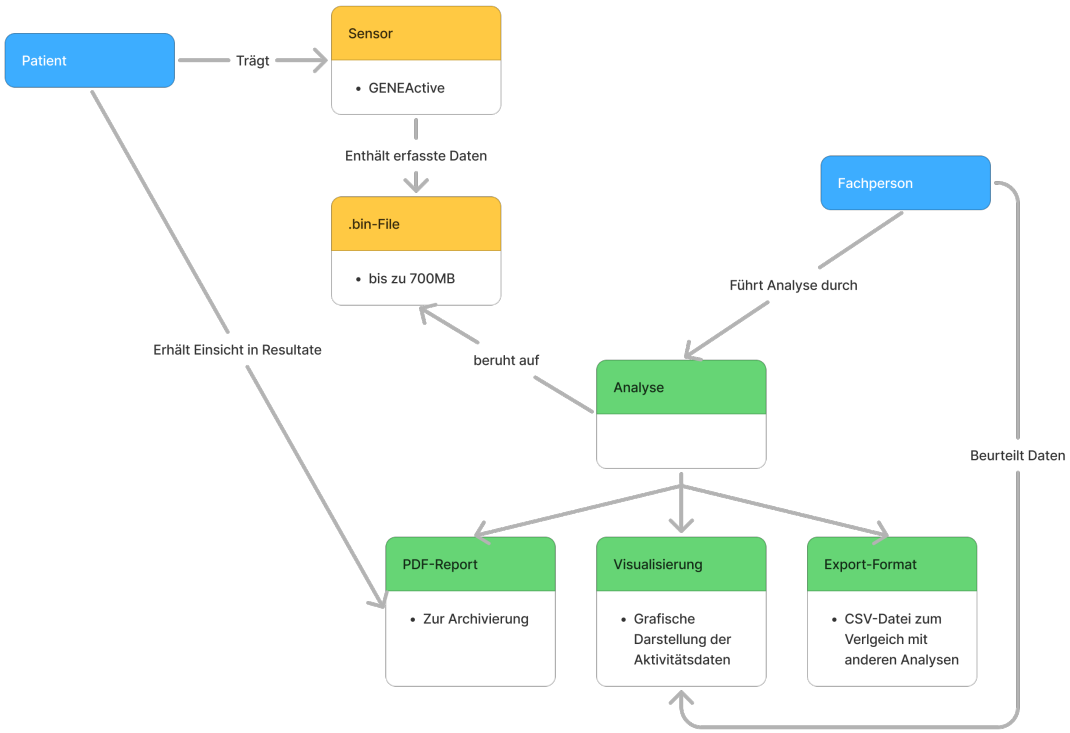


Abbildung 3.1: Domain Model

3.1.1 Erläuterung der Entitäten

Personen (Blau)

- **Patient**: Die Person, die den GENEActiv-Sensor trägt und deren Bewegungs- und Schlaf-

verhalten erfasst wird.

- **Fachperson**: Person die sich in der fachlichen Domäne auskennt und Schlüsse aus den analysierten Daten zieht.

Datenerfassung (Gelb)

- **Sensor (GENEActiv)**: Das Physische Gerät zur Erfassung von Beschleunigungsdaten. Der Patient trägt dieses am Körper.
- **.bin-File**: Die binäre Datei, die die vom Sensor erfassten Rohdaten enthält.

Analyse (Grün)

- **Analyse**: Der Prozess der Verarbeitung und Auswertung der Sensordaten, um relevante Informationen zu extrahieren.
- **PDF-Report**: Ein standardisiertes Dokument, das die Analyseergebnisse zusammenfasst und zur Archivierung dient.
- **Export-Format**: Standardisiertes Datenformat (CSV), das den Vergleich mit anderen Analysen ermöglicht.
- **Visualisierung**: Grafische Darstellung der Aktivitätsdaten zur besseren Verständlichkeit und Interpretation.

Kapitel 4

Anforderungen (Requirements)

4.1 Funktionale Anforderungen

Die funktionalen Anforderungen werden in Form von User Stories dokumentiert und in Epics zusammengefasst.

1. **Import**: Dieser Epic beschreibt alle Schritte, die vor der fertigen Analyse durchgeführt werden. Dies betrifft den Import der Datei und auch den Start des Analyseprozesses.
2. **Visualisierung**: In dieser Gruppe befinden sich Anforderungen, die nach der durchgeführten Analyse zum Tragen kommen. Dies betrifft die Darstellung der Daten sowie eventuelle Exportmöglichkeiten.
3. **Analyse**: In diesem Epic befinden sich die inhaltlichen Anforderungen an die Analyse. Konkret geht es darum, welche Informationen ausgewertet werden sollen.

Definitionen

Status E = Erfüllt, TE = Teilweise erfüllt, N = Nicht erfüllt

Prioritäten:

- **Priorität 1 - Ergänzende Funktionalität:** Diese Funktionen verbessern die Benutzerfreundlichkeit und Anwendungsqualität, sind jedoch für die Kernfunktionalität nicht relevant.
- **Priorität 2 - Erweiterte Funktionalität:** Diese Features sind wichtig zu implementieren, da sie die Nutzererfahrung erheblich verbessern. Sie bilden jedoch keine Voraussetzung für die Grundfunktionalität.
- **Priorität 3 - Grundlegende Funktionalität:** Ohne diese Features bietet das Produkt, im Vergleich zur vorherigen Lösung, keinen Mehrwert.

4.1.1 Import

Alle Anforderungen in diesem Abschnitt werden im Kapitel 8 behandelt.

Nr.	Beschreibung	Priorität	Status

1.1	Als Benutzer möchte ich die extrahierte .bin-Datei direkt über das **Benutzerinterface importieren** können, um den Importprozess zu vereinfachen.	3	E
1.2	Als Benutzer möchte ich die **Analyse in wenigen Schritten** starten, um Zeit zu sparen und den Prozess effizient zu halten.	3	E
1.3	Als Benutzer möchte ich, dass die **Schlaf- und Aktivitätsanalyse simultan** ausgeführt wird, um Zeit zu sparen.	3	E
1.4	Als Benutzer möchte ich auswählen können welche Analyse durchgeführt wird, um Zeit zu sparen falls eine bestimmte Analyse nicht gebraucht wird.	2	E
1.5	Als Benutzer möchte ich vor dem Start der Analyse zwischen verschiedenen **Cut-Off-Werten** wählen können, um die Analyse an die Aktivitätsniveaus der Patienten anzupassen.	3	E
1.6	Als Benutzer möchte ich während der laufenden Analyse den **Status des Fortschritts** sehen, damit ich abschätzen kann wie viel Zeit bis zur Fertigstellung noch benötigt wird.	2	E

4.1.2 Visualisierung

Die Implementierung und Designentscheidungen zu den Graphen werden im Kapitel 6.7 erläutert. Eine Übersicht zu den implementierten Graphen findet sich im Anhang 12.8.

Nr.	Beschreibung	Priorität	Status
2.1	Als Fachkraft möchte ich nach dem Analyseprozess eine **ansprechende und übersichtliche Visualisierung** der Daten sehen, um die Informationen leicht zu verstehen, zu analysieren und weiterzuverarbeiten. *Die Validierung erfolgte durch Usability-Tests, siehe Abschnitt 6.8.3.*	3	E
2.2	Als Patient möchte ich eine **leicht verständliche Visualisierung** der Daten sehen, um die Informationen auch als Laie zu verstehen.	2	E

2.3	Als Fachkraft möchte ich die Wahl zwischen der Patientenansicht und einer **detailreicheren Ansicht** mit mehr Informationen, um die Ergebnisse besser beurteilen zu können. *Der Wechsel zwischen den Ansichten wurde verworfen, da die einfach verständlichen Graphen laut dem Kunden ausreichend sind.*	1	N
2.4	Als farbenblinder Nutzer möchte ich Visualisierungen in einem **barrierefreien Design** sehen, um die Daten ohne Einschränkungen interpretieren zu können (siehe Abschnitt 6.9).	2	E
2.5	Als Fachkraft möchte ich verschiedene **Analysen miteinander vergleichen**, um Veränderungen nachvollziehen zu können (siehe Abschnitt 6.7).	1	E
2.6	Als Benutzer möchte ich die Analyseergebnisse **als PDF exportieren** können, um die Analyse zu teilen oder zu archivieren (siehe Abschnitt 8.7).	3	E

4.1.3 Analyse

Die Implementierung und Designentscheidungen zu den Graphen werden im Kapitel 6.7 erläutert. Eine Übersicht zu den implementierten Graphen findet sich im Anhang 12.8.

Nr.	Beschreibung	Priorität	Status
3.1	Als Benutzer möchte ich nach dem Analyseprozess die **Sitz-Zeit** des Patienten angezeigt bekommen, um Phasen der Inaktivität identifizieren zu können.	3	E
3.2	Als Benutzer möchte ich nach dem Analyseprozess die Zeit der **moderaten Aktivität** des Patienten sehen, um das Aktivitätsniveau besser bewerten zu können.	3	E
3.3	Als Benutzer möchte ich nach dem Analyseprozess die Zeit der **intensiven Aktivität** des Patienten sehen, um die körperliche Belastung zu analysieren.	3	E
3.4	Als Benutzer möchte ich nach dem Analyseprozess die **Schlafzeit** des Patienten sehen, um die Schlafgewohnheiten zu beurteilen.	3	E
3.5	Als Benutzer möchte ich nach dem Analyseprozess die **Zeit im Bett** des Patienten sehen, um die Gesamtdauer der Ruhephasen abschätzen zu können.	3	E

3.6	Als Benutzer möchte ich nach dem Analyseprozess die **Schlafeffizienz** des Patienten sehen, um die Qualität des Schlafes zu bewerten.	3	E

4.2 Nicht-funktionale Anforderungen

Um die nicht-funktionalen Anforderungen zu kategorisieren, verwenden wir das FURPS-Modell. Die NFAs werden direkt in diesem Abschnitt erläutert und auf Abschnitte verwiesen, welche die Anforderungen weiter erläutern oder die Implementierung erklären.

4.2.1 Functionality

Nr.	Beschreibung
NFA 1	**Die Applikation importiert extrahierte .bin-Dateien vom GENEActive Sensor.** .bin-Dateien werden für den Import unterstützt. Ebenfalls können vom Programm exportierte PDF-Reports importiert werden (siehe Abschnitt 8.5 und 8.7).
NFA 2	**Sämtliche Verarbeitung sensibler Daten erfolgt lokal auf dem Gerät des Benutzers, ohne Übertragung an externe Server oder Dienste.** Die Handhabung der zu verarbeitenden Daten erfolgt lokal auf dem jeweiligen Gerät (siehe Kapitel zur Implementierung 8).
NFA 3	**Die Applikation speichert keine personenbezogenen Daten permanent.** Um die Verarbeitung zu ermöglichen, speichert die Applikation Analysen temporär in einem Benutzerverzeichnis. Beim Beenden des Programms werden automatisch alle Dateien aus diesem Verzeichnis entfernt (siehe Abschnitt 8.2.2).

4.2.2 Usability

Nr.	Beschreibung
NFA 4	**Die Benutzeroberfläche soll eine intuitive Navigation und konsistente UI-Elemente enthalten.** Für die Benutzeroberfläche wurden vorgefertigte Komponenten vom Package Semi.Avalonia verwendet. Eine intuitive Bedienbarkeit wurde mit den Usability-Tests validiert (siehe Abschnitt 6.8.1).
NFA 5	**4 von 5 Benutzern können ohne vorherige Erklärung innerhalb von maximal 2 Minuten, abzüglich der Analysezeit, eine vollständige Datenanalyse durchführen.** Voraussetzung hierfür ist, dass Nutzer wissen, wie die Daten vom Sensor extrahiert werden.

	Aufgrund der einfachen Funktionalität waren bei den durchgeführten Tests alle Teilnehmer schneller als die geforderten zwei Minuten. Diese Tests konnten aufgrund zeitlicher Beschränkungen nur mit zwei Testpersonen durchgeführt werden (siehe Abschnitt 6.8.2).
NFA 6	**Die Benutzeroberfläche hat bei Benutzerinteraktionen eine maximale Reaktionszeit von 100ms.** Aufgrund der Projektstruktur und zeitlichen Beschränkungen wurden keine automatisierten Performance-Messungen implementiert. Die Anforderung wird jedoch durch die Architektur der Anwendung erfüllt: UI-Operationen werden über asynchrone Befehle und das MVVM-Pattern abgewickelt, wodurch die Benutzeroberfläche stets responsiv bleibt. Während der manuellen Tests zeigte sich eine unmittelbare visuelle Rückmeldung bei allen Interaktionen ohne wahrnehmbare Verzögerungen.
NFA 7	**Die Bedienelemente entsprechen dem Punkt 1.4.1 der WCAG 2.1 Richtlinien zur Barrierefreiheit.** Es wurden ausreichende Farbkontraste für Text, Hintergrundelemente und Visualisierungen gewählt, sodass Inhalte auch für Benutzer mit Sehbeeinträchtigungen gut erkennbar sind (siehe Abschnitt 6.9).

4.2.3 Reliability

Nr.	Beschreibung
NFA 8	**Die Anwendung darf nicht mehr als einmal pro 20 durchgeführten Analysen abstürzen.** Die Stabilität der Anwendung wird durch umfassende Fehlerbehandlung und Try-Catch-Blöcke in kritischen Bereichen gewährleistet (siehe Abschnitt 8.8.2). Die R-Integration ist isoliert implementiert, sodass Fehler in der Datenverarbeitung die Hauptanwendung nicht beeinträchtigen. Während der finalen Tests wurden keine Abstürze dokumentiert.
NFA 9	**Die Anwendung hält bei Datensätzen von bis zu 7 Tagen kontinuierlicher Sensoraufzeichnung bei 90% aller Analysevorgänge die weiter unten angegebenen Performancewerte ein.** Die Anwendung wurde mit verschiedenen Datensatzgrössen getestet, einschliesslich 7-Tage-Aufzeichnungen. Die Performancewerte befanden sich bei den Tests mit den Geräten der Entwickler bei jeder Analyse im Bereich *übertroffen* in den Landingzones.

4.2.4 Performance

Nr.	Beschreibung
NFA 10	**Die Geschwindigkeit der Schlafanalyse vom Start bis zur Ausgabe der Werte soll schneller als zuvor sein.** Ziele sind in Form von Landing Zones definiert (siehe Tabelle 4.9). Die Geschwindigkeitssteigerung erfolgte durch Optimierung der GENEActive-Bibliotheken. Siehe 7.3
NFA 11	**Die Geschwindigkeit der Aktivitätsanalyse vom Start bis zur Ausgabe der Werte soll schneller als zuvor sein.** Ziele sind in Form von Landing Zones definiert (siehe Tabelle 4.9). Die Geschwindigkeitssteigerung erfolgte durch Optimierung der GENEActive-Bibliotheken. Siehe 7.3
NFA 12	**Die Startzeit der Anwendung bis zur vollen Funktionsbereitschaft beträgt maximal 3 Sekunden.** Das System erfüllt hierbei folgende Voraussetzungen: mindestens 8GB RAM, Prozessorleistung von mindestens 1GHz und zwei Prozessorkerne. Alle gemessenen Startzeiten lagen unter der maximalen Startzeit von 3 Sekunden. Der Durchschnitt über alle Systeme beträgt 1.78 Sekunden (siehe Anhang 12.7).

4.2.5 Supportability

Nr.	Beschreibung
NFA 13	**Die Anwendung ist vollständig kompatibel mit Windows 11 (ab Version 23H2) und macOS (ab Version Sonoma 14.0) ohne Einschränkungen in Funktionalität oder Performance.** Die Kompatibilität wird durch die Verwendung von .NET 9 und Avalonia UI gewährleistet, welche native Cross-Platform-Unterstützung bieten. Tests wurden auf Windows 11, macOS (Intel-basiert) und Linux (Pop!_OS) durchgeführt und bestätigen die vollständige Funktionalität ohne plattformspezifische Einschränkungen. Tests auf Apple Silicon konnten aufgrund fehlender Testgeräte nicht durchgeführt werden.
NFA 14	**Die Architektur verwendet modulare Komponenten mit definierten Schnittstellen, die unabhängig aktualisiert werden können. Somit wird sichergestellt, dass die Unterstützung weiterer Sensoren einfach implementiert werden kann.** Das Factory-Pattern und Dependency Injection ermöglichen eine lose Kopplung der Komponenten. Neue Sensoren können durch Implementierung der definierten Interfaces hinzugefügt werden (siehe Abschnitt 8.9).
NFA 15	**Die Software wird als ausführbare Datei bereitgestellt.**

	Die Anwendung wird als eigenständige ausführbare Datei mit .NET 9 Single-File-Deployment generiert. Ein Installer wird verwendet, um die Anwendung im System zu registrieren und verfügbar zu machen, jedoch ohne Installation zusätzlicher Runtime-Komponenten. Die R-Runtime muss separat vom Benutzer installiert werden (siehe Abschnitt 8.10).
NFA 16	**Die Qualität des Codes wird durch eine Code-Coverage von mindestens 80% sichergestellt. Die Abdeckung wird mit der Implementierung automatisierter Tests ermöglicht. Bestehender R-Code wird nicht in die Coverage miteinbezogen.** Unit Tests mit NUnit decken alle kritischen Komponenten ab und erreichen eine Code-Coverage von 81% (siehe Abschnitt 9.4.5). UI-Komponenten werden nicht durch Unit Tests abgedeckt, sondern durch manuelle Tests validiert. R-Scripts werden separat durch Regressionstests (siehe Abschnitt 9.5) abgedeckt.
NFA 17	**Die Konsistenz der Analyseergebnisse muss durch automatisierte Regressionstests sichergestellt werden. Bei Änderungen am bestehenden R-Code dürfen die Analyseergebnisse nicht von den bereits validierten Resultaten der Vorgängerversion abweichen, es sei denn, die Abweichung ist explizit beabsichtigt, dokumentiert und begründet.** Die Implementierung der Regressionstests erfolgte durch den Einsatz von Referenzdateien. Details hierzu finden sich in Abschnitt 9.5.3.

4.3 Landing Zones

Zur Zeit der Definition dieser Anforderungen ist eine konkrete Angabe zur Verbesserung der Analysezeit nur schwer möglich. Aus diesem Grund verwenden wir Landing Zones, um akzeptable Zeitrahmen zu definieren and denen wir uns orientieren wollen.

	Minimum	Ziel	Übertroffen
Schlafanalyse	10min	6min	4min
Aktivitätsanalyse	10min	6min	4min

Tabelle 4.9: Landing Zones

Kapitel 5

Architektur

5.1 Einleitung

Da die Applikation hochsensible Gesundheitsdaten verarbeitet, sind Sicherheitsanforderungen von zentraler Relevanz und haben zur Entscheidung für eine lokalen Desktop-Anwendung beigetragen. Diese Architekturentscheidung stellt sicher, dass die Verarbeitung der sensiblen Patientendaten ausschliesslich auf dem jeweiligen Endgerät erfolgt, ohne externe Netzwerkkommunikation.

Eine Client-Server-Architektur wurde im Rahmen unserer Evaluation geprüft, jedoch angesichts der Projektanforderungen verworfen. Diese Entscheidung basiert primär auf drei Faktoren:

1. Eine solche Architektur würde zusätzliche Sicherheitsrisiken bei der Verarbeitung sensibler Patientendaten verursachen.
2. Die hohen Performanceanforderungen wären für die komplexe Datenverarbeitung in einer verteilten Umgebung schwieriger zu erfüllen.
3. Die lokale Verarbeitung bietet in diesem speziellen Anwendungsfall keine signifikanten Nachteile bezüglich der Nutzererfahrung, die den erhöhten Implementierungsaufwand rechtfertigen würden.

Für die Verarbeitung der Sensordaten existieren bereits etablierte Bibliotheken in der Programmiersprache R. Nach eingehender Prüfung haben wir uns gegen eine Portierung dieser Komponenten entschieden, da die betreffenden Bibliotheken eine hohe Komplexität aufweisen und eine Neuentwicklung den Projektumfang überschreiten würde. Stattdessen integrieren wir die bestehenden R-Bibliotheken in unsere Anwendungsarchitektur.

Die detaillierte Bewertung der evaluierten Technologien wird in den folgenden Abschnitten dargelegt.

5.2 Kriterien

In unserer Evaluation und mit Bezug auf die Anforderungen haben wir folgende Liste mit den für uns ausschlaggebenden Kriterien für die Technologieauswahl erstellt:

- **Autonome Ausführbarkeit**: Die Anwendung muss als eigenständige Executable (.exe) ohne zusätzliche Installationen oder Abhängigkeiten lauffähig sein

- **R-Integration**: Möglichkeit zur Einbindung des bestehenden R-Codes als Service oder Komponente
- **Performance**: Technologie sollte effiziente Algorithmen und parallele Datenverarbeitung unterstützen
- **Plattformunabhängigkeit**: Lauffähigkeit auf allen gängigen Betriebssystemen (Windows, Linux, MacOS)
- **Clean-Code-Prinzipien**: Unterstützung moderner Software-Engineering-Praktiken wie SOLID-Prinzipien, Dependency Injection und Testbarkeit zur Förderung von wartbarem und lesbarem Code
- **Entwicklerfreundlichkeit**: Gute Dokumentation, aktive Community und intuitive Entwicklungsumgebung
- **Moderne UI/UX-Möglichkeiten**: Unterstützung für zeitgemässe Benutzeroberflächen
- **Geringe Herstellerabhängigkeit**: Möglichkeit zur klaren Trennung zwischen Geschäftslogik und Technologie-spezifischem Code
- **Langfristige Wartbarkeit**: Etablierte Technologie mit kontinuierlicher Weiterentwicklung und Support
- **Visualisierungsbibliotheken**: Verfügbarkeit leistungsfähiger Bibliotheken für wissenschaftliche Datenvisualisierung
- **Testbarkeit**: Gute Unterstützung für automatisierte Tests zur Qualitätssicherung
- **Deployment-Freundlichkeit**: Einfache Paketierung und Verteilung der Anwendung
- **Datenschutzkonformität**: Möglichkeit zur Implementierung notwendiger Datenschutzmassnahmen für sensible Patientendaten

5.3 Evaluation

Auf Basis der vorgängigen Kriterien wurden nun verschiedene Technologien geprüft und nach einem Punktesystem beurteilt.

Erklärung zur Punkteverteilung:
1 = Ungeeignet, 2 = Durchschnittlich, 3 = Gut, 4 = Sehr gut, 5 = Hervorragend

5.3.1 Technologie 1: Python mit Tkinter

Einfaches, in Python integriertes UI-Framework mit plattformübergreifender Funktionalität und minimalistischem Design-Ansatz.

Kriterium	Beurteilung	Punkte
Autonome Ausführbarkeit	Bereitstellung als eigenständige Executable erfordert zusätzliche Tools wie PyInstaller, was den Prozess verkomplizieren kann.	3
R-Integration	Integration mit R über rpy2 ist möglich, jedoch mit Einschränkungen bei der nahtlosen Einbindung komplexer R-Funktionalitäten.	3
Performance	Python als interpretierte Sprache weist Performancedefizite bei rechenintensiven Operationen auf.* *Performance-kritische Logik ist dennoch primär im R-Code implementiert.	2
Clean-Code-Prinzipien	Die Sprache ermöglicht sauberen Code, jedoch fehlen strenge Typisierung und strukturierte Interfaces für robuste Architekturen.	3
Entwicklerfreundlichkeit	Sehr zugängliche Syntax ermöglicht schnellen Einstieg und hohe Produktivität, besonders durch Pythons intuitive und lesbare Programmiersprache.	5
Moderne UI/UX-Möglichkeiten	Tkinter bietet grundlegende UI-Elemente, wirkt jedoch optisch veraltet und bietet begrenzte Anpassungsmöglichkeiten für moderne Interfaces.	2
Geringe Herstellerabhängigkeit	Als Teil der Python-Standardbibliothek besteht keine direkte Herstellerabhängigkeit.	5
Langfristige Wartbarkeit	Tkinter wird zwar weiterhin unterstützt, erhält aber wenig aktive Weiterentwicklung für moderne Anforderungen.	3
Visualisierungsbibliotheken	Umfangreicher Zugang zu leistungsstarken Bibliotheken wie Matplotlib, Seaborn und Plotly mit guten Integrationsmöglichkeiten durch Canvas-Widgets in Tkinter.	4
Testbarkeit	Grundlegende Testmöglichkeiten existieren, aber die Simulation von UI-Interaktionen gestaltet sich aufwendiger als in moderneren Frameworks.	3
Deployment-Freundlichkeit	Python-Abhängigkeiten und Umgebungsvariablen können zu Deployment-Herausforderungen führen, besonders bei komplexeren Anwendungen.	2
Datenschutzkonformität	Als vollständig lokale Anwendung bietet Datenschutzkonformität, da sämtliche Daten lokal verarbeitet werden und keine externe Kommunikation notwendig ist.	5

Tabelle 5.1: Bewertung von Python mit Tkinter

5.3.2 Technologie 2: C#/.NET (Avalonia UI)

Plattformübergreifendes UI-Framework mit nativer Performance-Optimierung und .NET-Kernfunktionalität.

Kriterium	Beurteilung	Punkte
Autonome Ausführbarkeit	Bereitstellung als Single-File-Executable ist ohne Komplikationen realisierbar.	5
R-Integration	Mehrere etablierte Bibliotheken ermöglichen die nahtlose Einbettung von R-Code in .NET-Anwendungen.	4
Performance	C# bietet dank JIT-Compiler exzellente Laufzeitperformance.* *Performance-kritische Logik ist dennoch primär im R-Code implementiert.	4
Clean-Code-Prinzipien	Die Sprache und das Framework unterstützen moderne Clean-Code-Paradigmen in vollem Umfang.	5
Entwicklerfreundlichkeit	Intuitive Syntax und robuste Features steigern die Produktivität, erfordern jedoch eine gewisse Einarbeitungszeit in die Konzepte.	4
Moderne UI/UX-Möglichkeiten	Ermöglicht die Entwicklung visuell ansprechender, nativer Anwendungen mit aktuellen UI-Patterns.	4
Geringe Herstellerabhängigkeit	Die architektonische Trennung durch das MVVM-Pattern gewährleistet hohe Flexibilität bei technologischen Änderungen.	4
Langfristige Wartbarkeit	.NET profitiert von kontinuierlicher Weiterentwicklung und langfristiger Unterstützung durch Microsoft und die Open-Source-Community.	5
Visualisierungsbibliotheken	Solides Angebot an Visualisierungsbibliotheken, wenn auch nicht so umfangreich wie bei reinen WPF-Anwendungen.	3
Testbarkeit	Umfassende Testmöglichkeiten mit etablierten Frameworks wie NUnit, die leicht zu implementieren und zu verstehen sind.	4
Deployment-Freundlichkeit	Der Deployment-Prozess ist unkompliziert durch integrierte Tooling-Unterstützung und Self-Contained-Deployment-Optionen.	4
Datenschutzkonformität	Als vollständig lokale Anwendung werden Datenschutzanforderungen inhärent erfüllt, ohne externe Datenübertragung.	5

Tabelle 5.2: Bewertung von C#/.NET (Avalonia UI)

5.3.3 Technologie 3: Electron.js + React.js

Webbasiertes Framework für Desktop-Anwendungen, das Webtechnologien wie JavaScript, HTML und CSS mit plattformübergreifender Funktionalität kombiniert.

Kriterium	Beurteilung	Punkte
Autonome Ausführbarkeit	Bereitstellung als eigenständige Anwendung problemlos möglich mit integriertem Packaging-System, jedoch resultieren grössere Installationspakete durch die eingebettete Chromium-Engine.	4
R-Integration	Integration mit R erfordert zusätzliche Bridge-Lösungen wie R-Serve oder HTTP-basierte APIs für die Kommunikation zwischen JavaScript und R.	2
Performance	Als browserbasierte Technologie mit gewissem Overhead verbunden, was bei rechenintensiven Operationen zu Einschränkungen führen kann.* *Performance-kritische Logik ist dennoch primär im R-Code implementiert.	2
Clean-Code-Prinzipien	React bietet strukturierte Komponenten-Architektur, folgt jedoch keinen standardisierten Konventionen, was bei komplexen Anwendungen schnell zu unsauberem Code führen kann.	4
Entwicklerfreundlichkeit	Breite Verfügbarkeit von JavaScript-Entwicklern und umfangreiches Ökosystem.	5
Moderne UI/UX-Möglichkeiten	Hervorragende Möglichkeiten zur Erstellung moderner, ansprechender Benutzeroberflächen mit zahlreichen verfügbaren UI-Bibliotheken.	5
Geringe Herstellerabhängigkeit	Starke Abhängigkeit von Electron.js und React.js. Es gibt dennoch Design Pattern, die Business-Logik und Framework-Logik trennbar machen.	3
Langfristige Wartbarkeit	Aktive Weiterentwicklung durch grosse Community, jedoch mit Risiken durch schnelle Entwicklungszyklen und potenziellen API-Änderungen.	3
Visualisierungsbibliotheken	Ausserordentlich reiches Angebot an hochwertigen JavaScript-Visualisierungsbibliotheken wie D3.js, Chart.js und zahlreichen React-spezifischen Komponenten.	5
Testbarkeit	Umfangreiche Testmöglichkeiten mit etablierten JavaScript-Frameworks wie Jest und React Testing Library.	4
Deployment-Freundlichkeit	Automatisierte Build-Prozesse mit electron-builder möglich, jedoch komplexer bei plattformübergreifenden Deployments und Update-Mechanismen.	3
Datenschutzkonformität	Grundsätzlich als lokale Anwendung datenschutzkonform, jedoch erhöhtes Risiko unbeabsichtigter Netzwerkkommunikation durch Webtechnologie-Basis.	3

Tabelle 5.3: Bewertung von Electron.js + React.js

5.4 Auswertung und Architekturentscheid

5.4.1 Vergleich

	Python mit Tkinter	C#/.NET (Avalonia UI)	Electron.js + React.js
Autonome Ausführbarkeit	3	5	4
R-Integration	3	4	2
Performance	2	4	2
Clean-Code-Prinzipien	3	5	4
Entwicklerfreundlichkeit	5	4	5
Moderne UI/UX-Möglichkeiten	2	4	5
Herstellerabhängigkeit	5	4	3
Langfristige Wartbarkeit	3	5	3
Visualisierungsbibliotheken	4	3	5
Testbarkeit	3	4	4
Deployment-Freundlichkeit	2	4	3
Datenschutzkonformität	5	5	3
Summe	**40**	**51**	**43**

Tabelle 5.4: Vergleich der Technologien

C#/.NET (Avalonia UI) hat mit 51 Punkten die höchste Bewertung erreicht und geht somit als klarer Sieger aus diesem Vergleich hervor.

5.4.2 Erläuterung der Architekturentscheidung

Die Wahl von C#/.NET in Verbindung mit Avalonia UI als Technologiegrundlage für dieses Projekt resultiert aus einer sorgfältigen Analyse der Anforderungen. Ausschlaggebend waren:

Robuste Performance und moderne User Experience: C#/.NET ermöglicht die Entwicklung performanter, nativer Anwendungen, die eine moderne Nutzererfahrung bieten.

.NET Ökosystem: Ein umfangreiches, gut gewartetes Ökosystem mit breiter Unterstützung durch Microsoft und eine aktive Community.

Plattformübergreifende Entwicklung: Avalonia UI bietet die Möglichkeit, Anwendungen für Windows, macOS und Linux mit einer einzigen Codebase zu entwickeln. Dies ist nicht nur effizient, sondern auch notwendig, da beide Entwickler auf Unix-Systemen arbeiten.

Einbindung von R-Code .NET bietet Unterstützung für die Einbindung von R-Code, was für unser Projekt von zentraler Bedeutung ist.

Wir sind zuversichtlich, dass diese Technologieauswahl uns eine solide Basis bietet, um ein erfolgreiches und zukunftssicheres Projekt zu realisieren.

5.5 R-Code als Analysebasis

Die zentrale Datenanalyse des Projekts basiert auf bestehendem R-Code. Nachfolgend wird die Entscheidung begründet, diesen Code weiterzuverwenden anstatt eine vollständige Neuentwicklung anzustreben.

5.5.1 Bestehendes Framework

Die Firma ActiveInsights stellt umfangreiche R-Bibliotheken zur Verfügung, die speziell für das Auslesen und Klassifizieren von Accelerometer-Rohdaten entwickelt wurden. Diese Bibliotheken bieten ein etabliertes und validiertes Analysesystem.

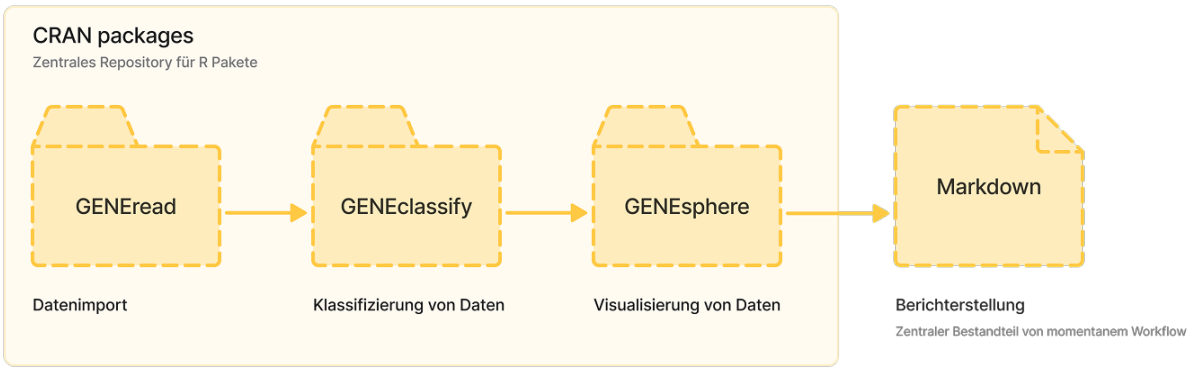


Abbildung 5.1: Workflow für die Datenverarbeitung und Berichtserstellung mit CRAN-Paketen

5.5.2 Bewertung einer Codeportierung

Die Portierung des R-Codes in eine andere Programmiersprache wurde nach der Analyse verworfen. Das bestehende System stellt kein isoliertes Skript, sondern ein komplexes Ökosystem von, von einander abhängigen, Bibliotheken dar, dessen Neuimplementierung den vorgesehenen Projektrahmen deutlich überschreiten würde.

Zudem besteht bei einer vollständigen Neuimplementierung ein erhebliches Risiko für Analyseabweichungen durch subtile Implementierungsunterschiede, was die Datenqualität und Vergleichbarkeit mit bestehenden Forschungsergebnissen gefährden könnte.

5.5.3 Performanceoptimierung des R-Codes

Die Optimierung der Ausführungsgeschwindigkeit ist für die Nutzererfahrung essenziell, da die aktuellen Analyselaufzeiten zu inakzeptablen Wartezeiten führen. Zur Verbesserung der Performance bestehen folgende Optionen:

- **Parallelisierung:** Mehrere Analysen können gleichzeitig auf verschiedenen Prozessorkernen ausgeführt werden.
- **C++-Integration:** Rechenintensive Teilfunktionen können mit Rcpp in C++ implementiert werden.

- **Vektorisierung:** Nutzung von vektorisierten Operationen anstelle von Schleifen.
- **Speicheroptimierung:** Gezielte Freigabe von Arbeitsspeicher und effizienter Umgang mit grossen Objekten.

Hierbei handelt es sich um Mutmassungen und eine Konkretisierung folgt erst im Rahmen der Implementation.

5.5.4 Integration in .NET

Für die Einbindung des R-Codes in eine .NET-Umgebung stehen drei Ansätze zur Verfügung:

- **Prozessbasiert:** Ausführung des R-Codes als externer Prozess mit Parameterübergabe und Ergebnisrückgabe über Dateien oder Standardausgabe. Einfach zu implementieren, jedoch mit Einschränkungen bei der Interaktivität.
- **R.NET-Integration:** Diese Bibliothek ermöglicht direkten Zugriff auf R-Funktionen aus C#-Code.
- **Native Integration:** Bei teilweiser Implementierung in C++ können über P/Invoke C++-Bibliotheken als DLLs eingebunden werden. Diese Methode bietet höchste Performance, erfordert jedoch erheblichen Entwicklungsaufwand und tiefgreifendes Verständnis der R-C++-Schnittstellen.

Die endgültige Entscheidung über die optimale Integrationsstrategie wird während der Implementierungsphase auf Basis konkreter Leistungsanforderungen getroffen (siehe Implementation 8.5).

5.5.5 Mögliche Alternativen zum GENEActiv-Ökosystem

Es stellte sich auch die Frage, ob es Alternativen zum GENEActiv-System gibt. Der von ActivInsights verfolgte Ansatz, spezialisierte Hardware mit Open-Source-Software zu kombinieren, ist jedoch relativ einzigartig am Markt.

Als Open-Source-Alternative existiert das GGIR R-Paket [vHSA+19], welches unter anderem auch ActiveInsights-Sensoren unterstützt. Auf kommerzieller Seite bietet ActiGraph mit ActiLife eine proprietäre Software-Lösung [Act24].

Da am MOVE-IT bereits etablierte Workflows und Datenbestände auf Basis der GENEActiv-Algorithmen existieren, wurde die Optimierung des bestehenden Systems einer Migration vorgezogen.

5.6 C4-Modell

Zur Visualisierung der Systemarchitektur und der Komponenten-Interaktion verwenden wir das C4-Modell, das einen strukturierten Überblick von der Kontext- bis zur Detailebene ermöglicht.

5.6.1 Level 1: Systemkontext

Das Kontextdiagramm zeigt die Interaktion zwischen Benutzer, Eingabedaten und der Desktop-Applikation zur Analyse von Aktivitäts- und Schlafdaten.

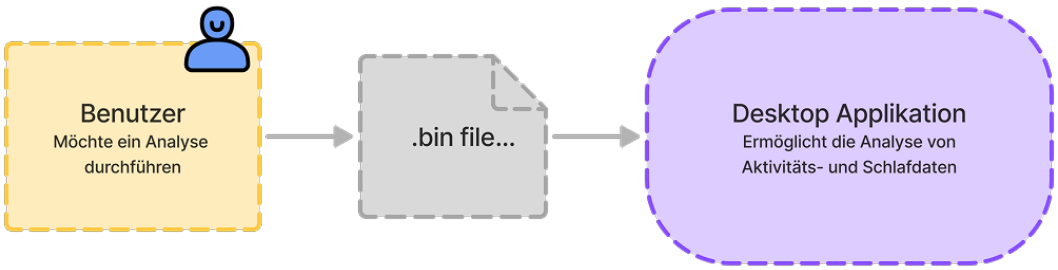


Abbildung 5.2: Level 1 des C4 Models: System Context Diagram

5.6.2 Level 2: Architekturebene

Das Container-Diagramm visualisiert den technischen Aufbau der Applikation mit Fokus auf die Interaktion zwischen C#/.NET-Frontend und R-Code.

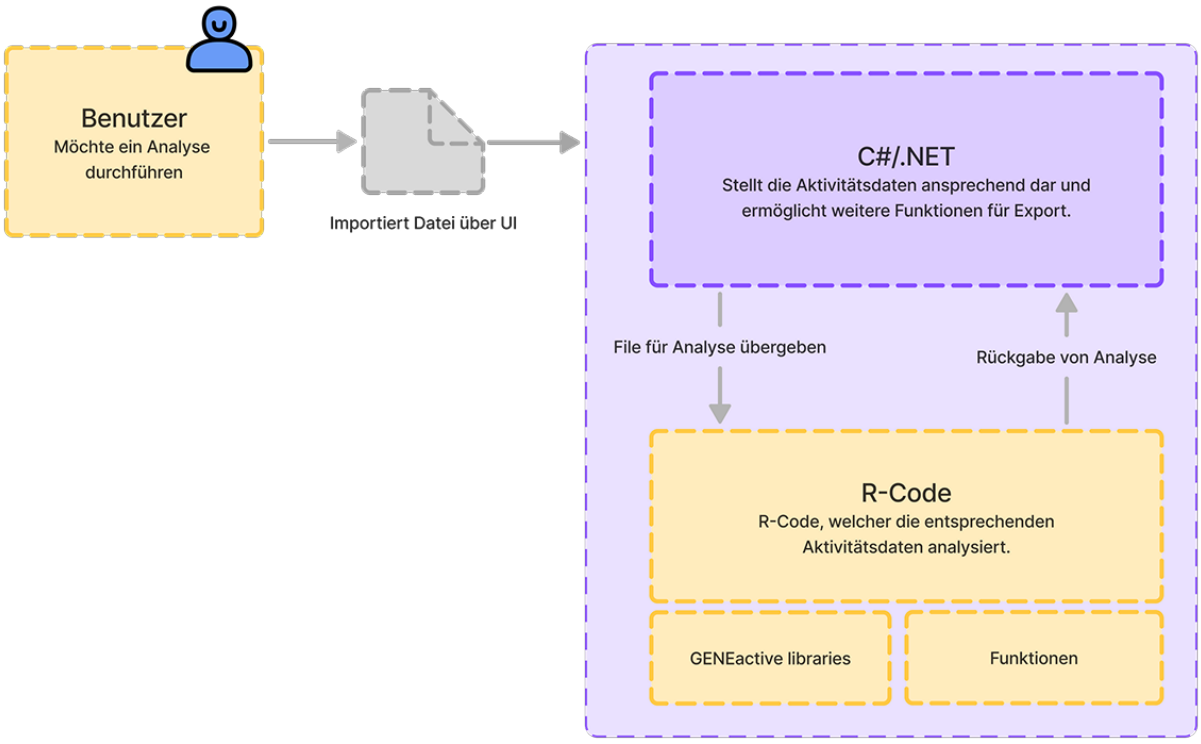


Abbildung 5.3: Level 2 des C4 Models: Container Diagram

5.7 Datenverarbeitungspipeline

Der Aufbau der Datenverarbeitungspipeline ist im Kapitel 7.1.2 zu finden.

5.8 Struktur Github-Repository

Das ActiveSense-Projekt ist in einer GitHub-Organisation namens **ActiveSense** strukturiert, die vier separate Repositories umfasst. Diese Aufteilung ermöglicht eine klare Trennung zwischen der Hauptanwendung und den modifizierten R-Bibliotheken und bietet eine zentrale Verwaltung aller projektbezogenen Repositories.

5.8.1 Repository-Übersicht

activesense
Das Hauptrepository enthält die vollständige .NET Desktop-Anwendung sowie das zentrale R-Analyseskript. Das Repository umfasst sowohl den C#-Code der Avalonia-Anwendung als auch die refaktorierten R-Skripte für die Sensordatenanalyse.

GENEAread
Dieses Repository enthält eine angepasste Version der ursprünglichen GENEAread-Bibliothek mit spezifischen Performance-Optimierungen. Die Modifikationen umfassen Verbesserungen in der Datenlesegeschwindigkeit und Speichernutzung, die für die Anforderungen von ActiveSense optimiert wurden.

GENEAclassify
Eine modifizierte Version der GENEAclassify-Bibliothek. Die Optimierungen betreffen insbesondere die StepCounter-Funktionalität.

activesense.r-universe.dev
Dieses Repository enthält die notwendigen Konfigurationsdateien für das R-Universe-Hosting der angepassten Bibliotheken. R-Universe ermöglicht es, die modifizierten GENEAread- und GENEAclassify-Pakete als Custom-Repository zu hosten, da diese aufgrund der Änderungen nicht über die Standard-CRAN-Repositories verfügbar sind.

Kapitel 6

UX/UI Design

6.1 Einleitung

Die Optimierung der Nutzererfahrung stellt einen zentralen Aspekt dieser Arbeit dar. Der gegenwärtige Workflow basiert auf der manuellen Ausführung eines R-Skripts, welches die Daten verarbeitet und Visualisierungen erzeugt. Dieser Prozess erweist sich als komplex in der Einarbeitung und birgt ein hohes Fehlerrisiko.

Im Rahmen der vorliegenden Arbeit soll dieser Workflow grundlegend verbessert werden. Unser Ziel ist die Entwicklung einer intuitiven Benutzeroberfläche, die es den Anwendern ermöglicht, in wenigen, klar definierten Schritten eine Analyse durchzuführen und die Ergebnisse effizient auszuwerten.

6.2 Analyse des bestehenden Workflows

Um die Grundlage für gezielte Verbesserungen zu schaffen, analysieren wir zunächst den aktuellen Workflow im Detail. Diese Bestandsaufnahme ermöglicht es, die kritischen Schwachstellen und Optimierungspotenziale zu identifizieren.

1. **RStudio starten:** Der Prozess beginnt mit dem Öffnen von RStudio und dem Laden der entsprechenden Projektdatei (.proj). Dieser Schritt setzt bereits Kenntnisse in der Bedienung von RStudio voraus.

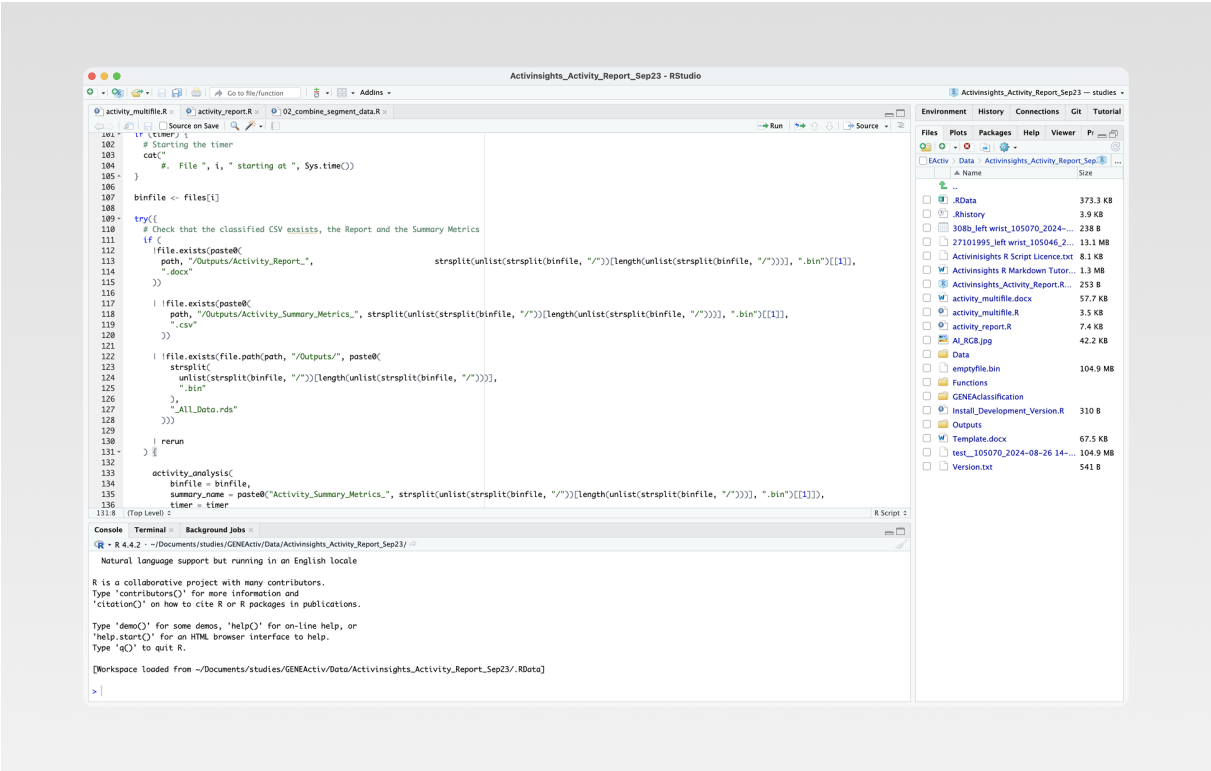


Abbildung 6.1: RStudio mit geöffnetem Projekt

2. **Daten bereitstellen:** Die zu analysierenden Dateien müssen manuell im korrekten Verzeichnis abgelegt werden. Dieser Schritt erfordert Kenntnis der Projektstruktur und kann bei falscher Ausführung zu Fehlern führen.

3. **Analyse initiieren:** Die Datenanalyse wird über die Knit-Funktion von RStudio gestartet. Hierbei werden die R-Skripte ausgeführt, die für die Verarbeitung der Rohdaten zuständig sind.

4. **Ergebnisdokumentation öffnen:** Nach Abschluss der Analyse wird automatisch eine Word-Datei mit den Ergebnissen erstellt, die manuell geöffnet werden muss.

5. **Ergebnisse interpretieren:** Die erzeugten Visualisierungen und Analyseergebnisse müssen vom Anwender interpretiert werden, was durch die Fülle an Informationen in der .docx-Datei zusätzlich erschwert wird. Tatsächlich relevant ist nur eine Teilmenge der Informationen.

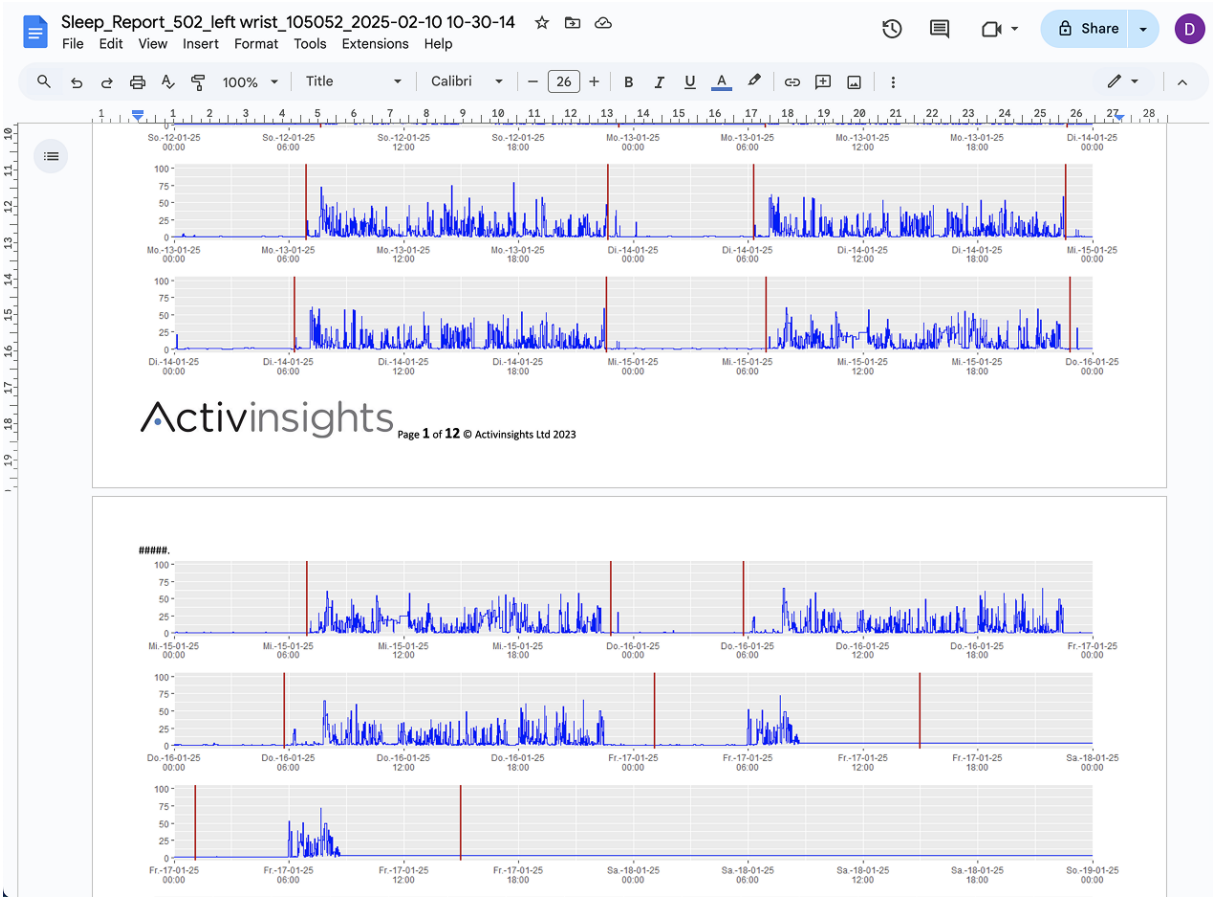


Abbildung 6.2: Generiertes .docx-Dokument // Seite 1 von 12

6.3 Identifizierte Probleme des aktuellen Workflows

Die Analyse des bestehenden Prozesses offenbart mehrere kritische Schwachstellen:

- **Schulungsaufwand:** Der Workflow erfordert eine dedizierte Einarbeitung der Anwender in den spezifischen Prozessablauf. Jeder neue Nutzer muss in die korrekte Abfolge der Arbeitsschritte, die Bedienung der Oberfläche und die Interpretation der Ergebnisse eingewiesen werden.
- **Fehleranfälligkeit:** Die manuelle Ausführung mehrerer sequenzieller Schritte erhöht das Risiko von Bedienungsfehlern.
- **Zeitaufwand:** Der gesamte Prozess ist zeitintensiv und umständlich, was die Effizienz im klinischen Alltag beeinträchtigt.
- **Mangelnde Benutzerführung:** Es fehlt an einer klaren Anleitung und visuellen Rückmeldung während des Prozesses.
- **Komplexe Ergebnisdarstellung:** Die generierten Visualisierungen sind nicht optimal auf die Bedürfnisse von Klinikern und Patienten zugeschnitten.

6.4 Analyse und Anforderungsdefinition

6.4.1 Zielsetzung

Das Hauptziel besteht in der deutlichen Vereinfachung der Nutzererfahrung durch Reduzierung der Komplexität bei der Durchführung von Aktivitätsanalysen. Die Überführung vom derzeitigen codebasierten Ausführungsmodell zu einer intuitiven Desktop-Applikation wird die Benutzerfreundlichkeit erheblich steigern.

6.4.2 Zielgruppenanalyse

Unsere Zielgruppe umfasst gemäss Aufgabenstellung Ärzte, Fachexperten und Patienten. Die Herausforderung liegt in der Entwicklung einer universellen Darstellungsform, die sowohl den Bedürfnissen der Experten als auch dem Verständnisniveau der Patienten gerecht wird.

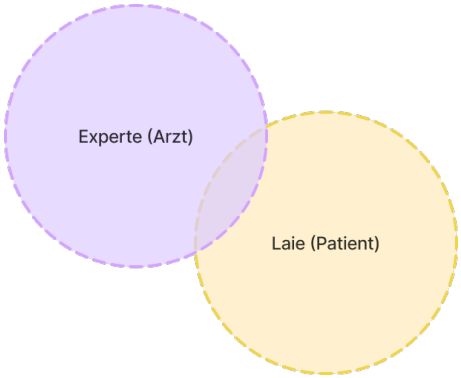


Abbildung 6.3: Zielgruppe

6.4.3 Nutzerbefragungen

In mehreren Gesprächen mit Mitarbeiter der MOVE-IT an der OST und Anreger des Projekts zur Prozessoptimierung, wurden die zentralen Probleme des bestehenden Workflows identifiziert. Diese bestätigen die bereits dokumentierten Schwachstellen: ein verwirrender, vielschrittiger und ineffizienter Prozess. Während die Performanceoptimierung Teil der technischen Entwicklung ist, fokussiert sich die UX/UI-Verbesserung auf die Schaffung eines prägnanten und effizienten Arbeitsprozesses.

6.4.4 Personas

Experte/Arzt

Die primäre Persona repräsentiert medizinische Fachkräfte oder MOVE-OST-Mitarbeiter mit spezifischem Fachwissen, die als Hauptanwender der Applikation fungieren. Sie führen Datenanalysen durch, interpretieren Ergebnisse und vermitteln diese an Patienten, indem sie die visualisierten Daten erklären und medizinisch einordnen.

Laie/Patient

Als sekundäre Persona fungiert der Patient, an dem die Daten erhoben wurden. Obwohl er die Applikation in der Regel nicht selbst bedient, muss er die präsentierten Visualisierungen verstehen können, wenn sie ihm vom Experten erklärt werden.

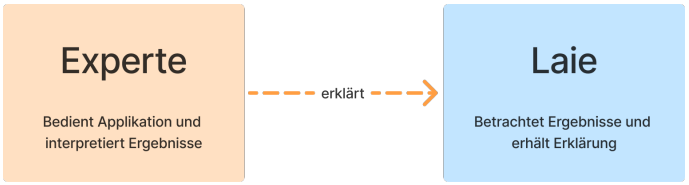


Abbildung 6.4: Beziehung zwischen Experte und Laie

6.5 Informationsarchitektur

Die Informationsarchitektur einer Anwendung definiert, wie Informationen organisiert, strukturiert und präsentiert werden, um eine intuitive Navigation und Bedienung zu ermöglichen.

6.5.1 Grundlegende Struktur

Die Anwendung folgt einem linearen Prozessmodell, das den Arbeitsablauf von Gesundheitsexperten widerspiegelt. Dieser Workflow umfasst den Import von Rohdaten, deren Analyse, die Konfiguration der Darstellung und schliesslich den Export der Ergebnisse. Diese Hauptphasen bilden die Grundpfeiler der Informationsarchitektur und werden in der Benutzeroberfläche klar voneinander abgegrenzt.

6.5.2 Hierarchische Organisation

Die Funktionalitäten der Anwendung sind in einem hierarchischen System organisiert:

- **Primäre Funktionsbereiche:** Import, Dateimanagement, Analyse, Konfiguration und Export
- **Sekundäre Funktionsbereiche:** Der Bereich, in dem die Analyseergebnisse visualisiert werden.

Diese zweistufige Hierarchie gewährleistet, dass Benutzer nicht mit zu vielen Optionen auf einmal konfrontiert werden, sondern gezielt auf die benötigten Funktionen zugreifen können.

Abbildung 6.5: Funktionsbereiche bildet visuelle Grundstruktur für Applikation

6.5.3 Informationsfluss

Der Informationsfluss innerhalb der Anwendung wurde so gestaltet, dass er den natürlichen Arbeitsablauf bei der Analyse von Aktivitätsdaten unterstützt:

- **Datenerfassung:** Import von .bin-Rohdateien oder bereits analysierten .csv-Dateien
- **Datenverwaltung:** Übersicht über verfügbare Dateien, Statusanzeige der Analyse
- **Datenverarbeitung:** Durchführung und Überwachung der Analyse
- **Datenvisualisierung:** Betrachtung der Analyseergebnisse mit verschiedenen Ansichten
- **Konfiguration:** Anpassung der Darstellung und Hervorhebung relevanter Informationen
- **Ergebnisexport:** Speicherung der Ergebnisse in verschiedenen Formaten

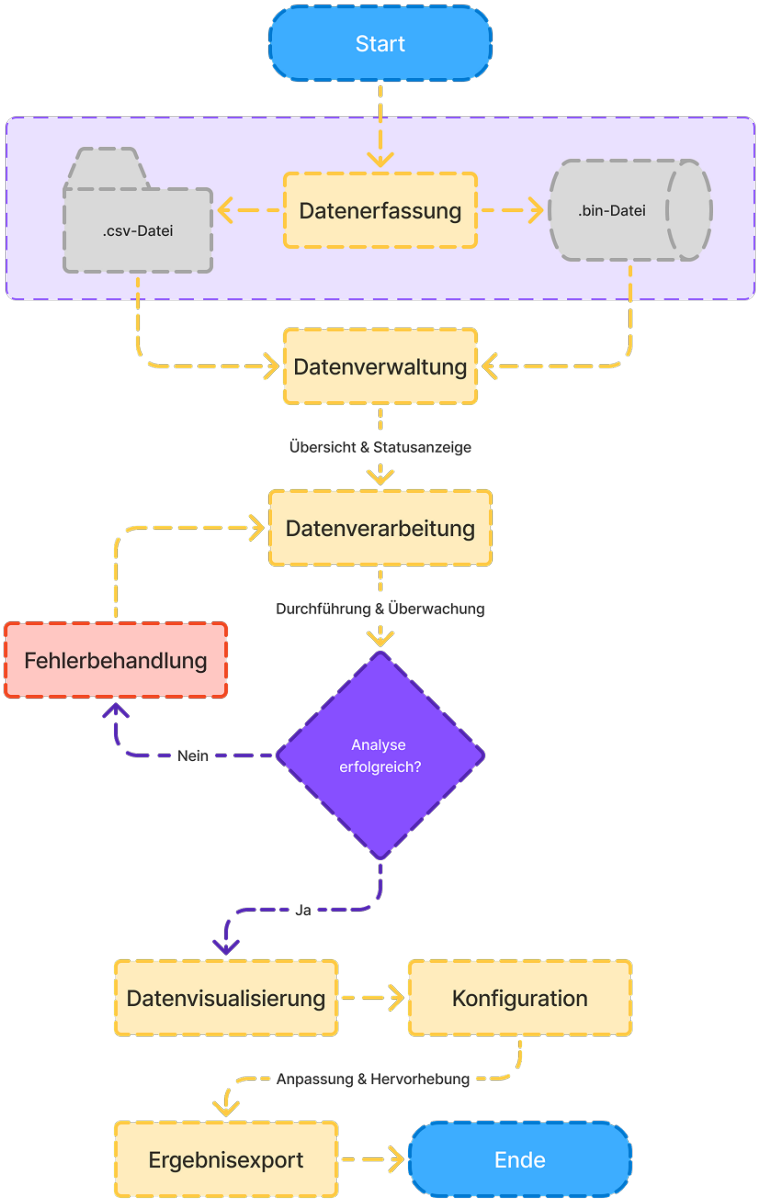


Abbildung 6.6: Flussdiagramm - Informationsfluss

6.5.4 Feedback-Mechanismen

Um Benutzer über den Status ihrer Aktionen zu informieren, werden verschiedene Feedback-Mechanismen implementiert:

- **Fortschrittsanzeige:** Visuelle Darstellung des Analyseprozesses durch Ladebalken mit Prozentangabe, die den aktuellen Stand der Verarbeitung transparent kommuniziert
- **Fehlermeldungen:** Informative Benachrichtigungen bei auftretenden Problemen mit konkreten Handlungsempfehlungen zur Behebung

Diese Mechanismen tragen zur Transparenz bei und vermitteln den Benutzern ein Gefühl der Kontrolle über den Analyseprozess.

6.6 Konzeption und Designentwicklung

In diesem Kapitel wird der iterative Prozess der Designentwicklung von den ersten Ideenskizzen bis zum finalen User Interface (UI) dargestellt. Ziel war es, eine benutzerfreundliche und effiziente Lösung zu erarbeiten.

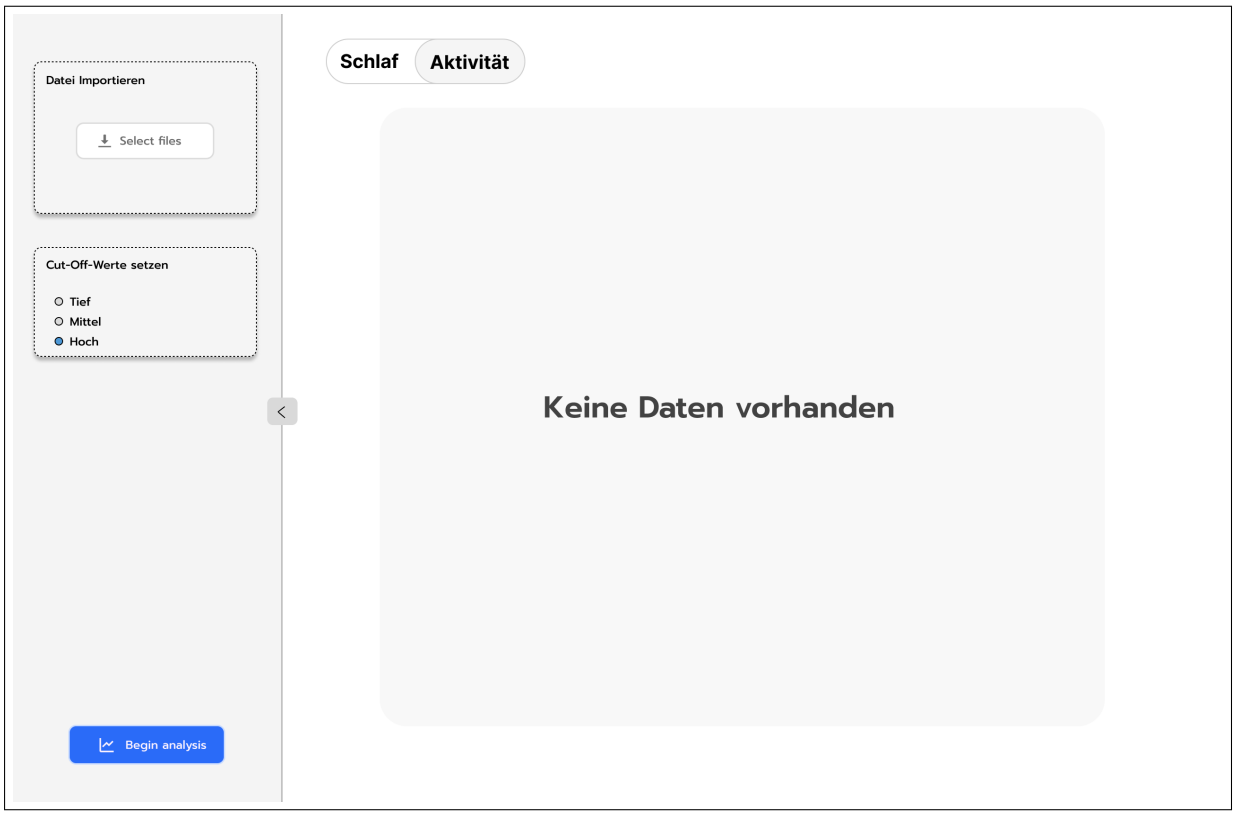
6.6.1 Wireframing und initiale Konzepte

Am Beginn des Designprozesses wurden verschiedene UI-Konzepte durch Wireframes untersucht. Diese Herangehensweise ermöglichte die Evaluation unterschiedlicher Layout-Varianten und UX-Ansätze zur Optimierung der Anwendungsarchitektur.

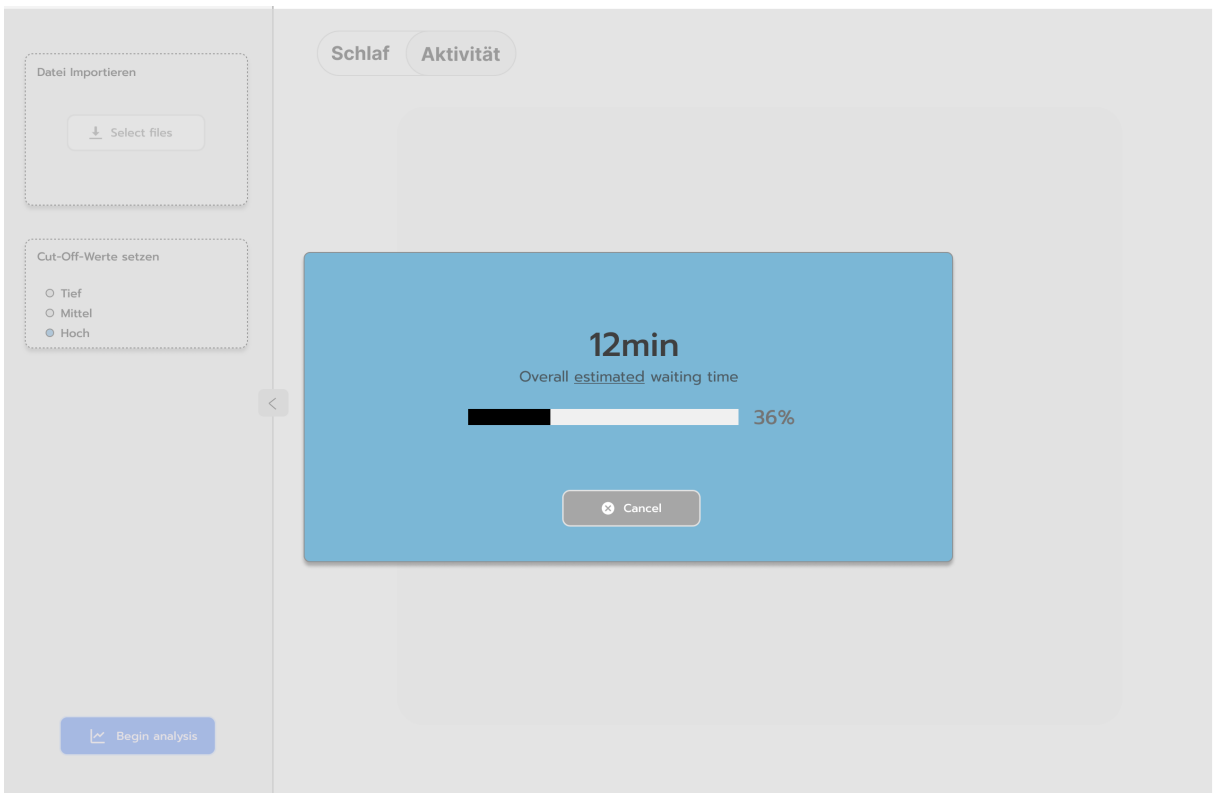
Konzept 1: Navigation mittels Sidebar

Der erste Ansatz zog die Integration einer Sidebar in Betracht. Die Idee dahinter war, dem Nutzer eine Übersicht über alle Kernfunktionen zu bieten und eine zentrale Benutzeroberfläche zu schaffen, die sämtliche relevanten Funktionalitäten und Einstellungsmöglichkeiten auf einem einzigen Bildschirm zusammenführt.

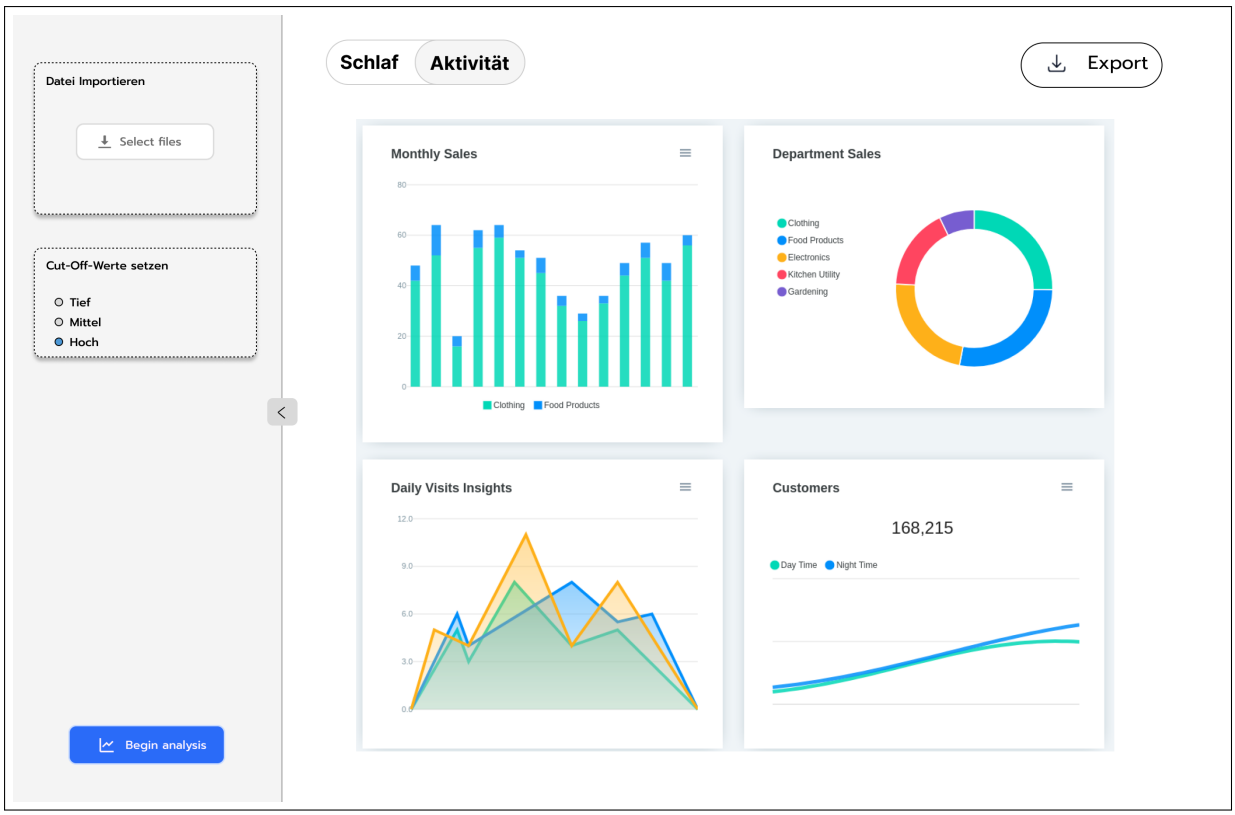
Schritt 1: Datei auswählen



Schritt 2: Datei hochladen



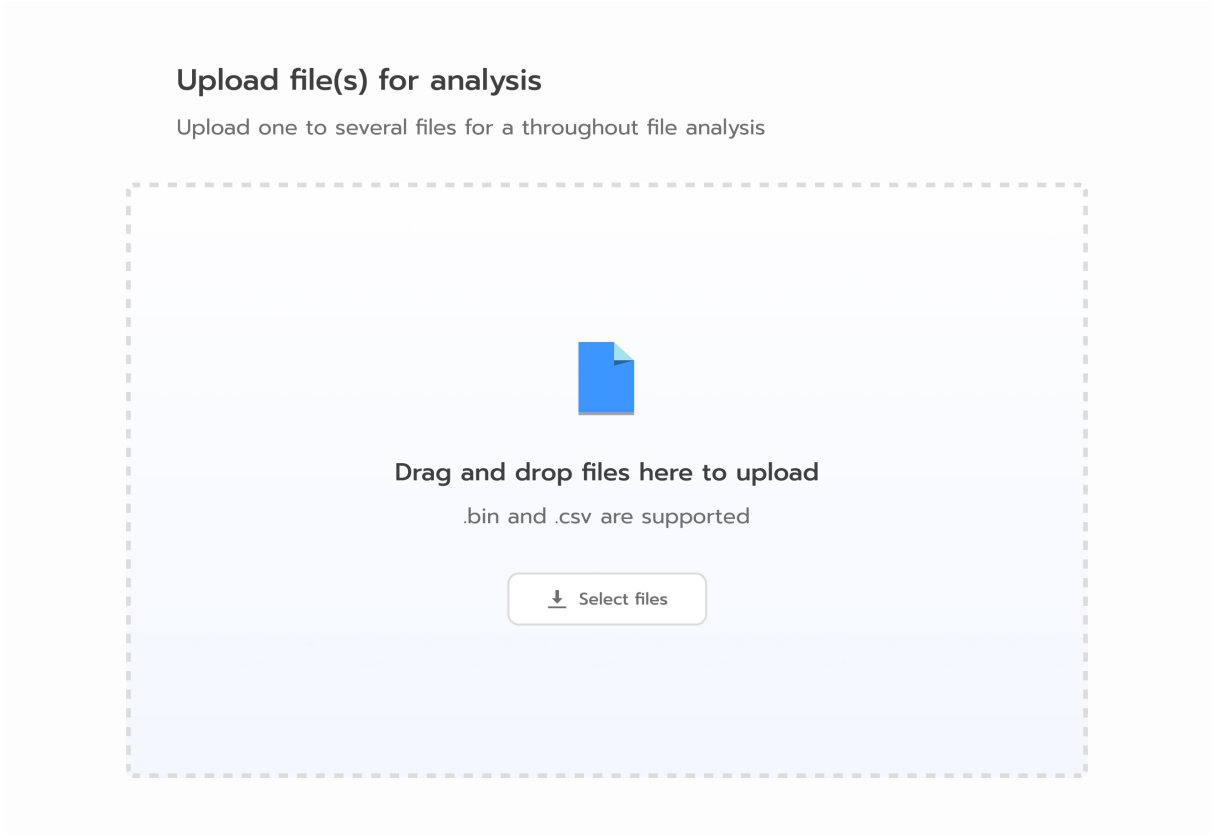
Schritt 3: Daten analysieren



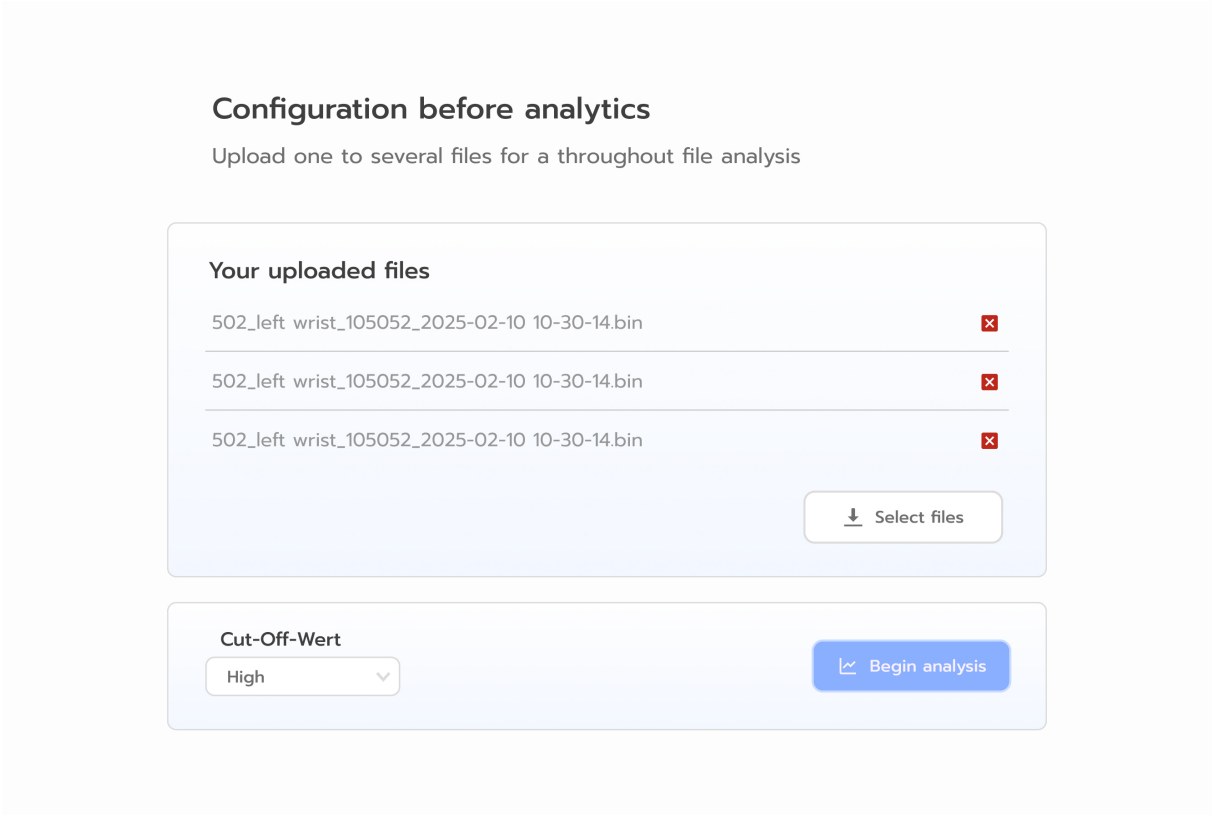
Konzept 2: Linearer Navigationsfluss

Ein alternativer Ansatz basierte auf einem strukturierten, linearen Aufbau. Die Intention hierbei war, den Nutzer schrittweise und geführt durch die verschiedenen Sektionen und Aufgaben der Applikation zu leiten. Dies sollte die Komplexität für den Endanwender reduzieren und den Fokus auf die jeweils aktuelle Interaktion lenken.

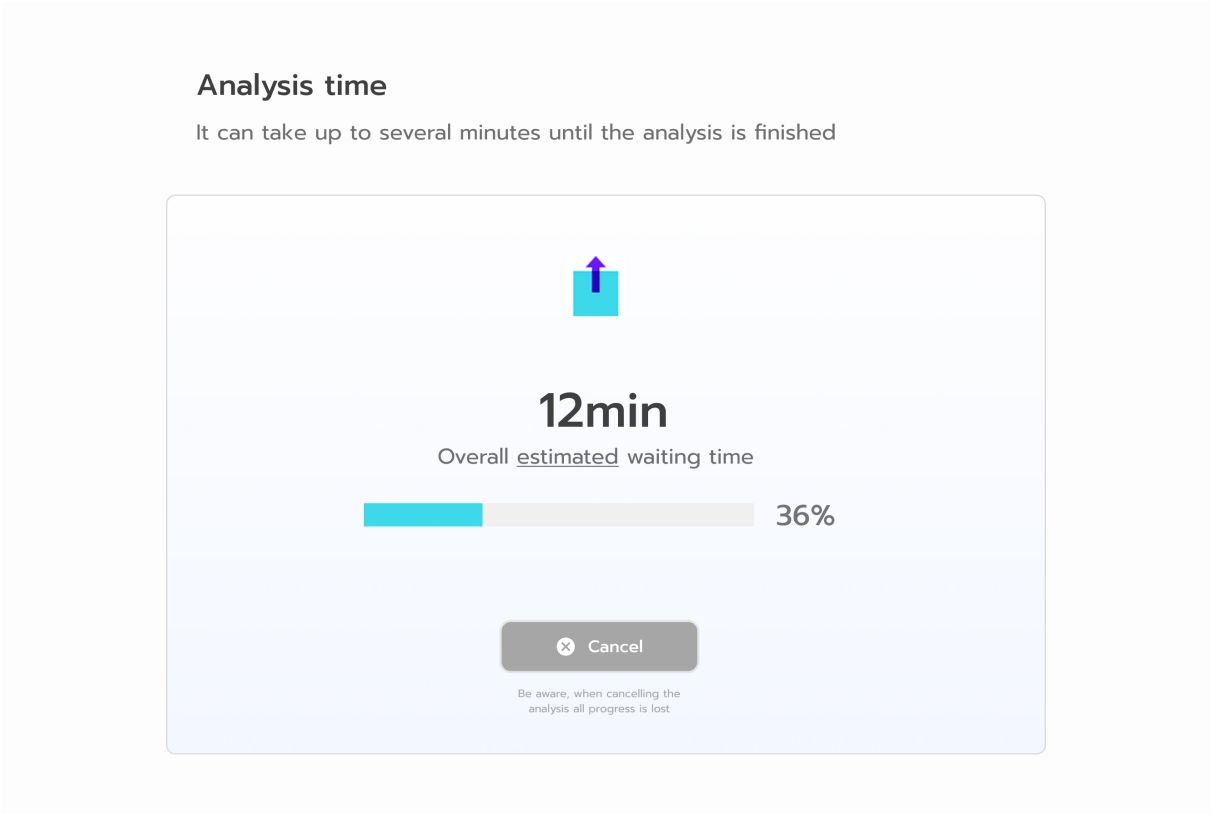
Schritt 1: Datei auswählen



Schritt 2: Datei prüfen und Upload starten



Schritt 3: Datei hochladen

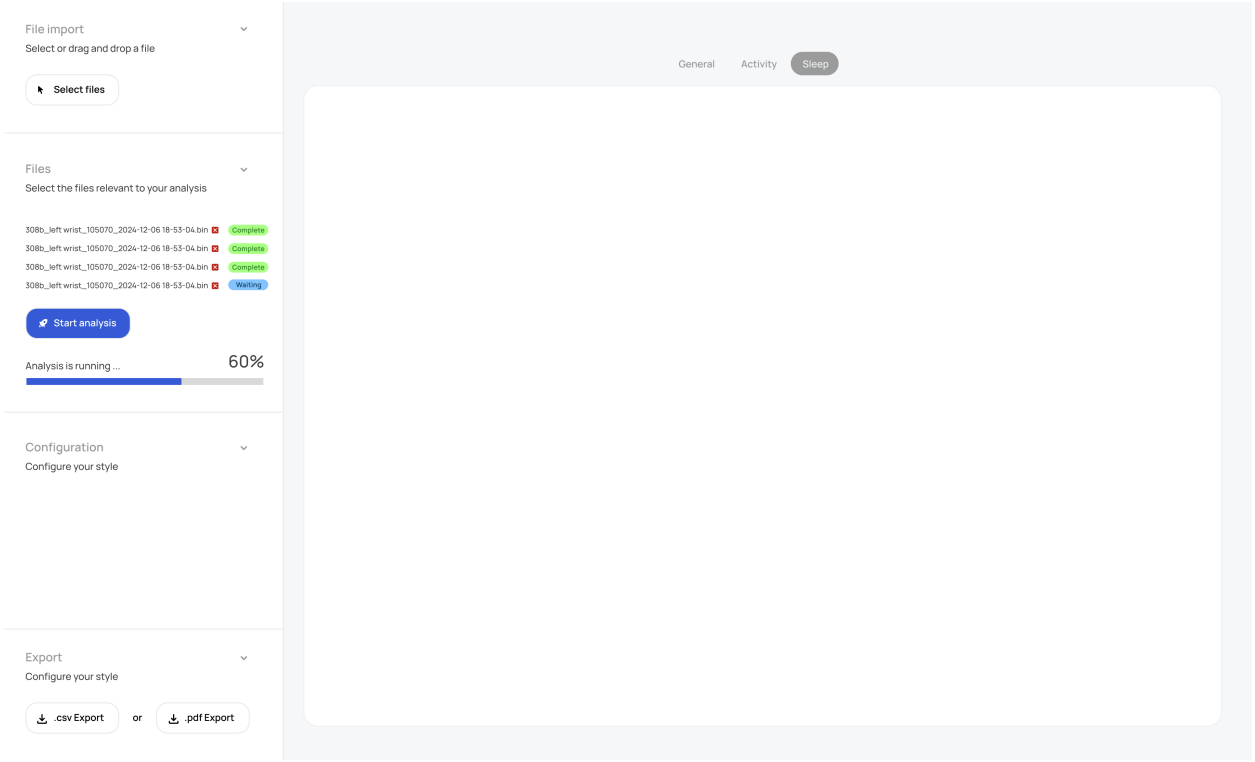


6.6.2 Prototypenentwicklung

Um die Eignung der beiden zuvor skizzierten Konzepte bewerten zu können, wurden für beide Ansätze Prototypen in Figma ausgearbeitet. Diese Ausarbeitungen ermöglichten eine Entscheidungsfindung zwischen den verschiedenen Designalternativen.

Prototyp A: Sidebar-basierte Navigation

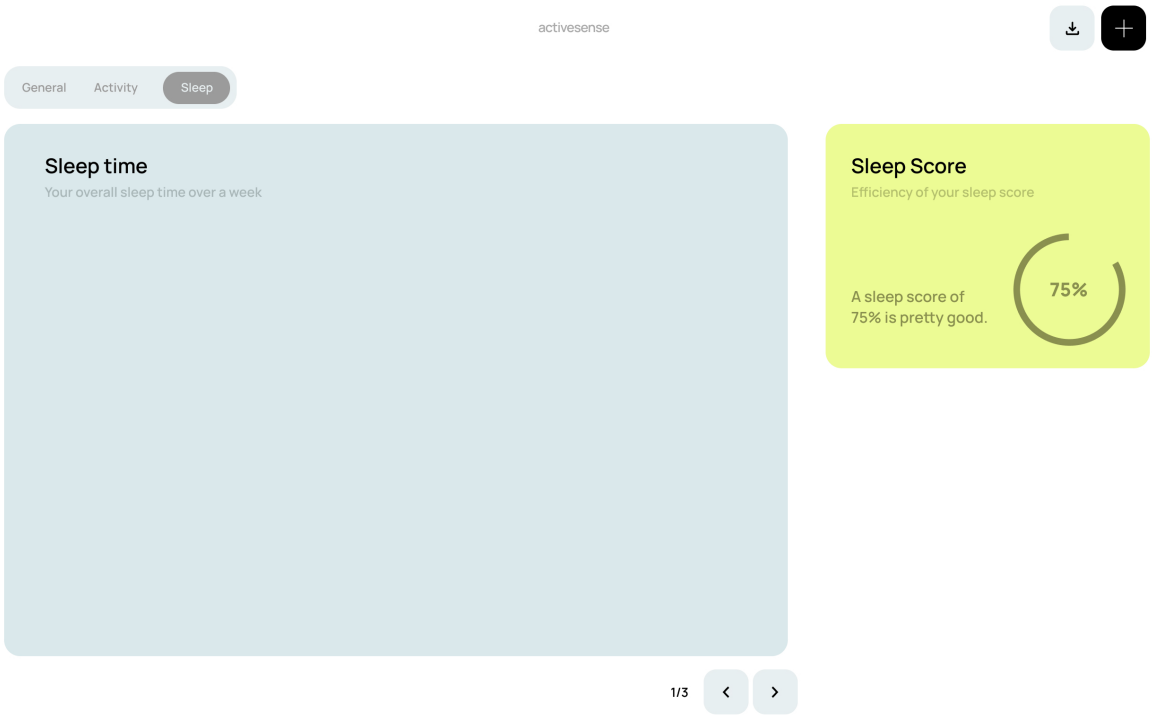
Der Prototyp mit Sidebar demonstrierte die Navigation über ein feststehendes Menü, welches die Hauptbereiche der Applikation zugänglich machte.



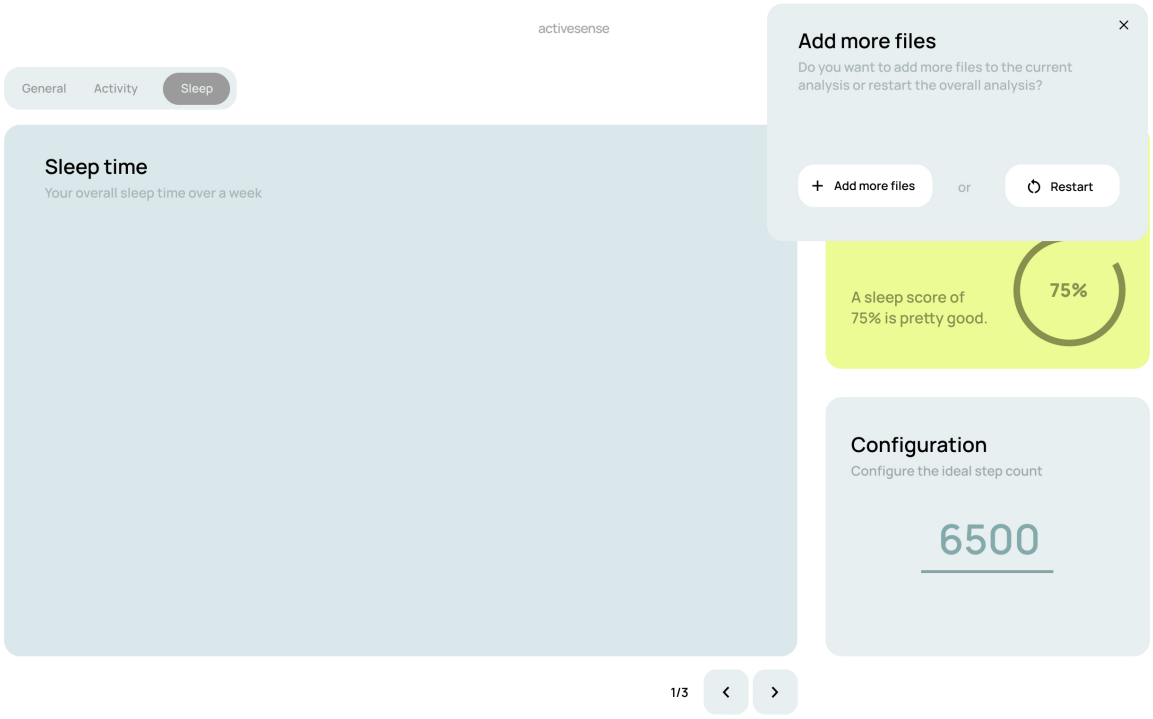
Prototyp B: Linearer Aufbau

Der Prototyp für den linearen Aufbau visualisierte den schrittweisen Durchlauf durch die Applikation. Hier wurde besonderer Wert auf klare Handlungsaufforderungen und eine reduzierte Informationsdarstellung pro Bildschirmansicht gelegt.

Main Dashboard

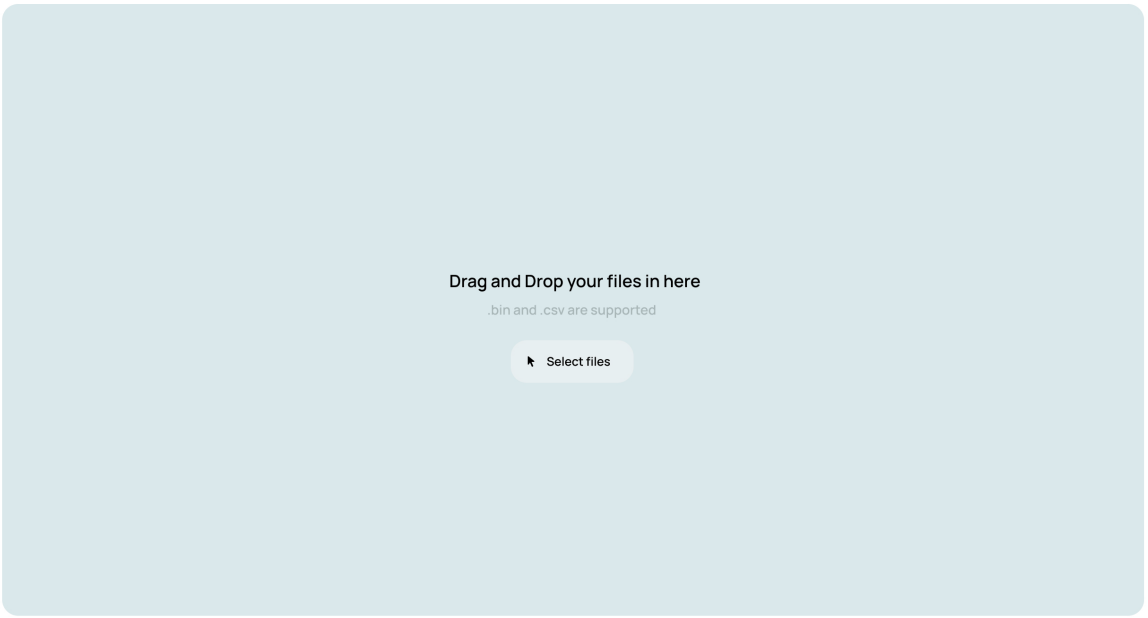


Import Schritt 1: Import starten



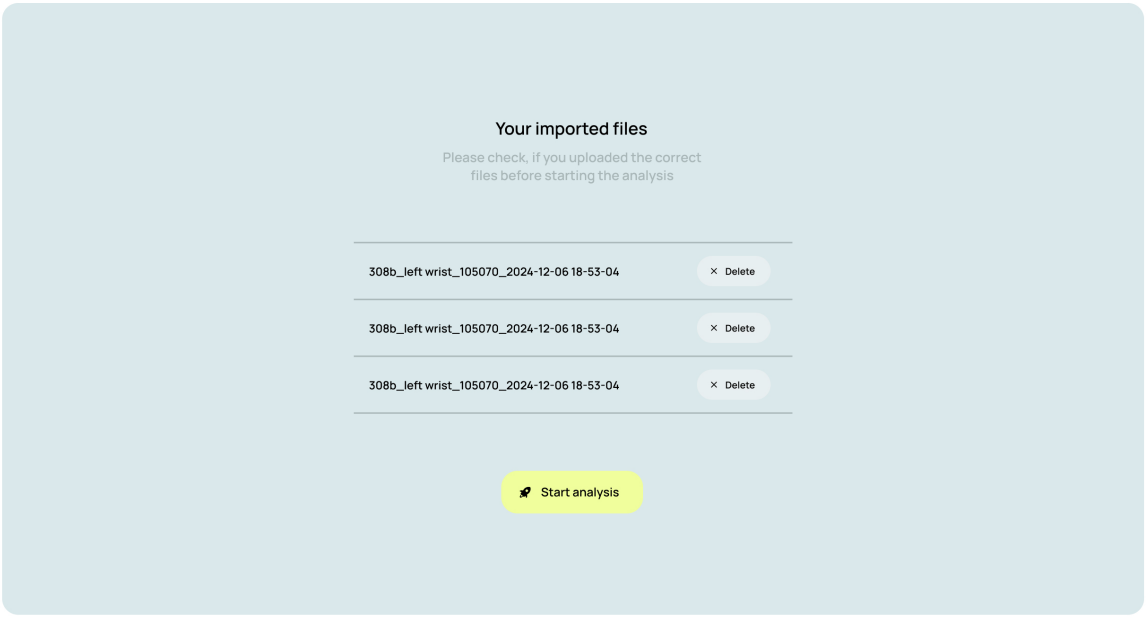
Import Schritt 2: Datei auswählen

activesense

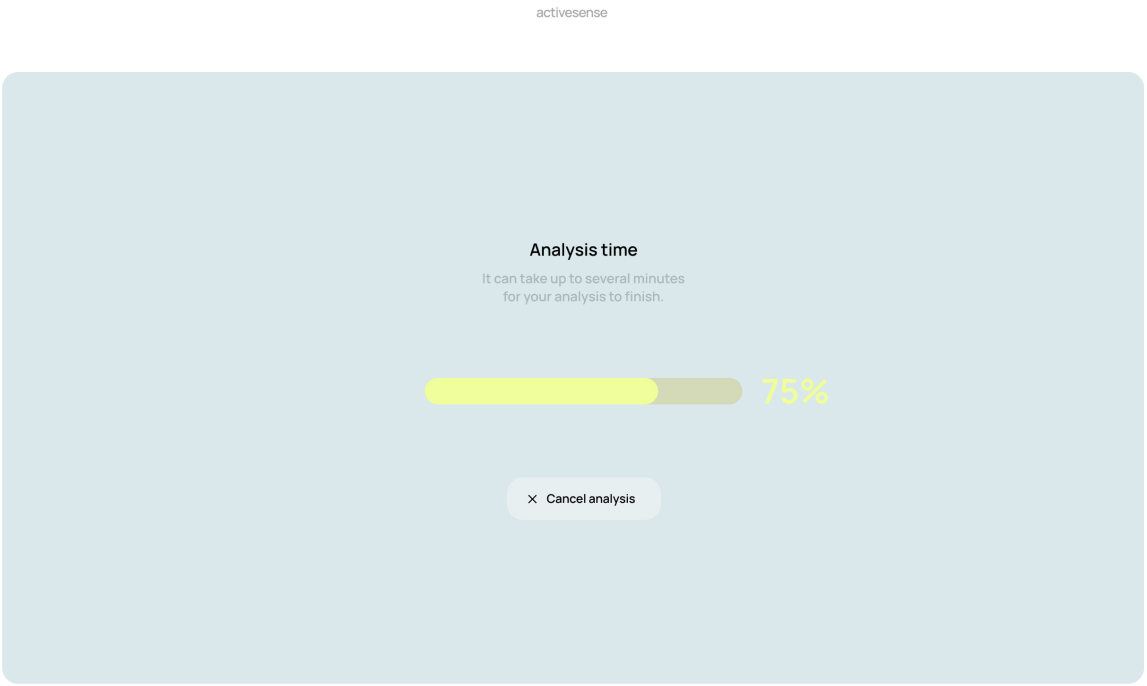


Import Schritt 3: Datei prüfen und Upload starten

activesense



Import Schritt 4: Datei hochladen

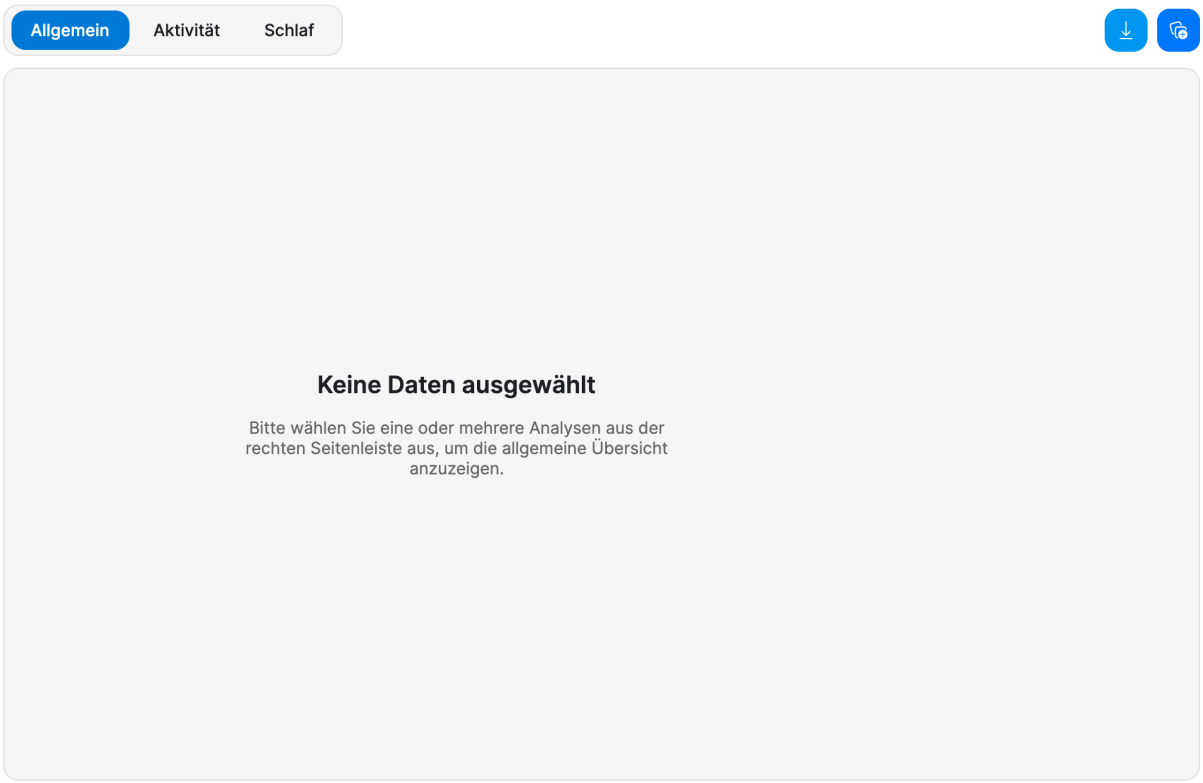


Das finale User Interface

Basierend auf der Evaluation der Prototypen fiel die Entscheidung für das finale User Interface auf ein lineares Navigationskonzept. Dieser Ansatz leitet den Nutzer fokussiert und schrittweise durch die gesamte Applikation.

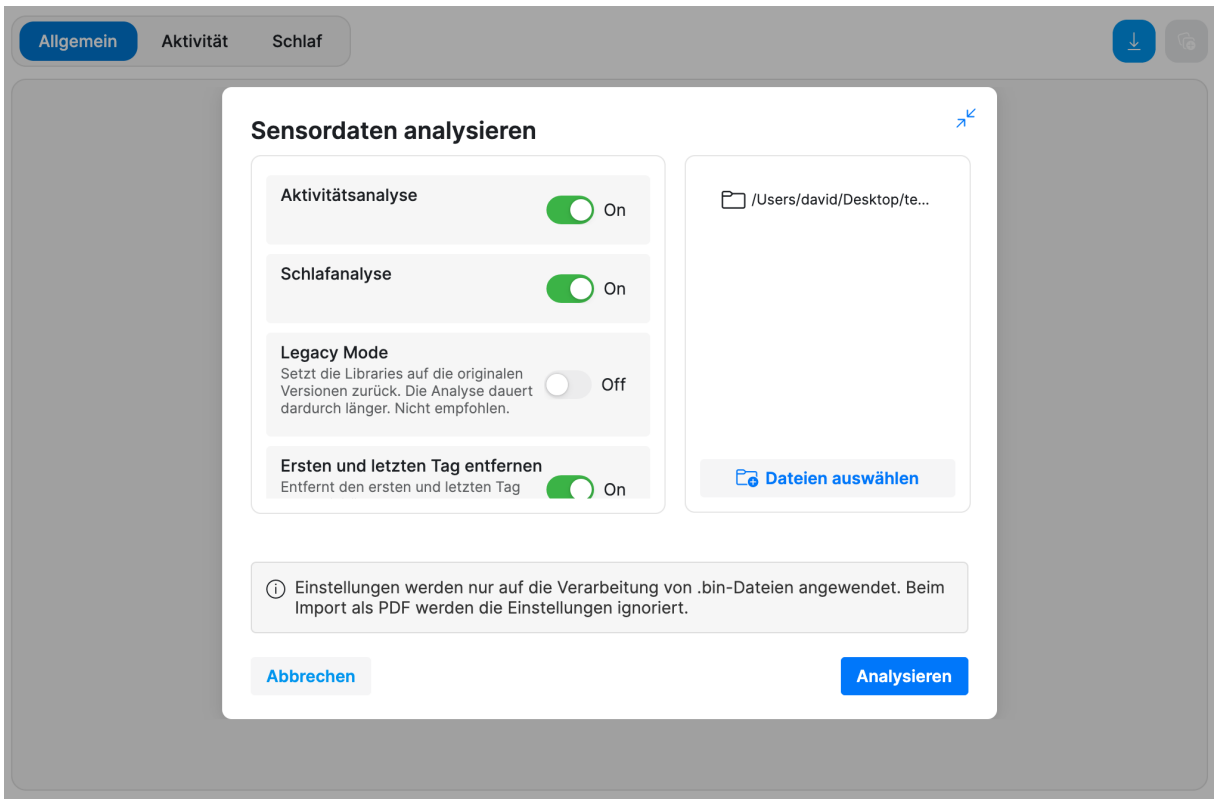
Das nachfolgende User-Interface zeigt die finale Implementierung der Anwendung, wie sie in der Avalonia realisiert wurde.

Dashboard



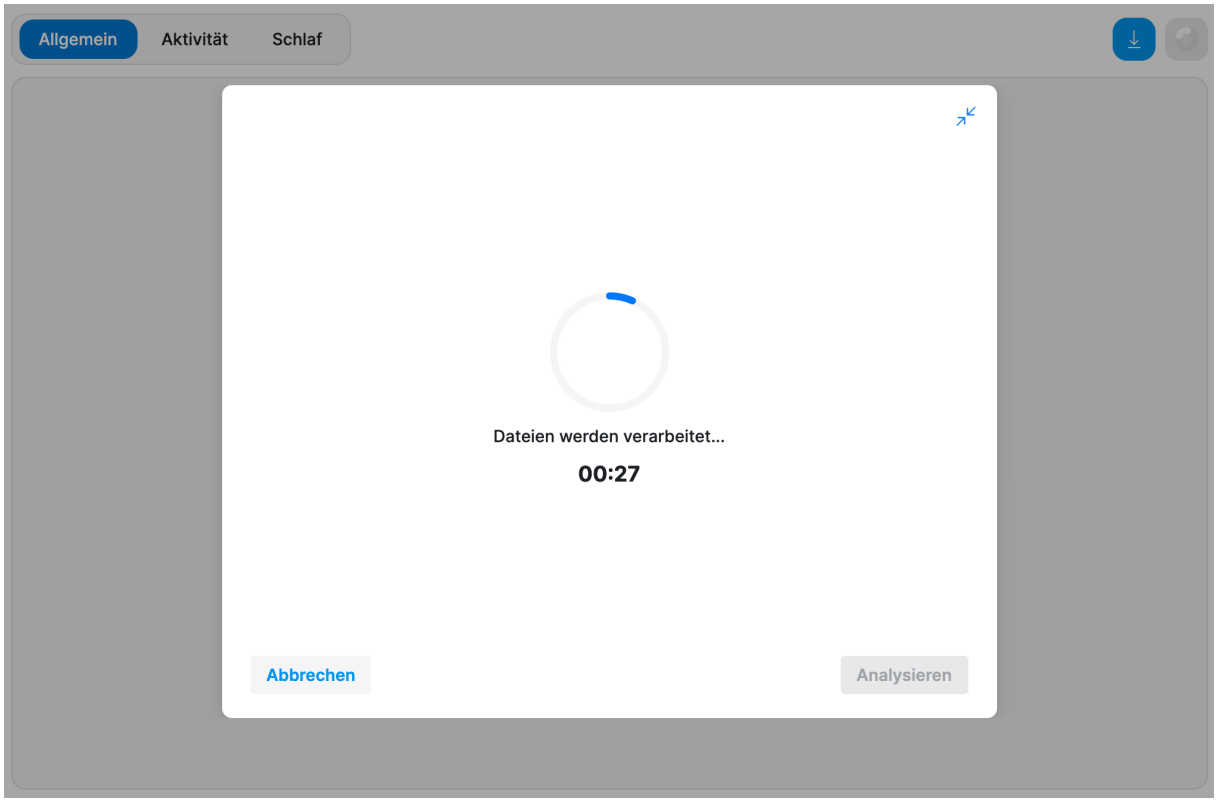
- **Navigation:** Die Tab-Navigation strukturiert die Anwendung in drei klar abgegrenzte Analysebereiche: Allgemein, Aktivität und Schlaf.
- **Upload + Export:** Im oberen rechten Bereich der Anwendung sind zwei Buttons für den Datenimport und -export positioniert.

Datei Upload



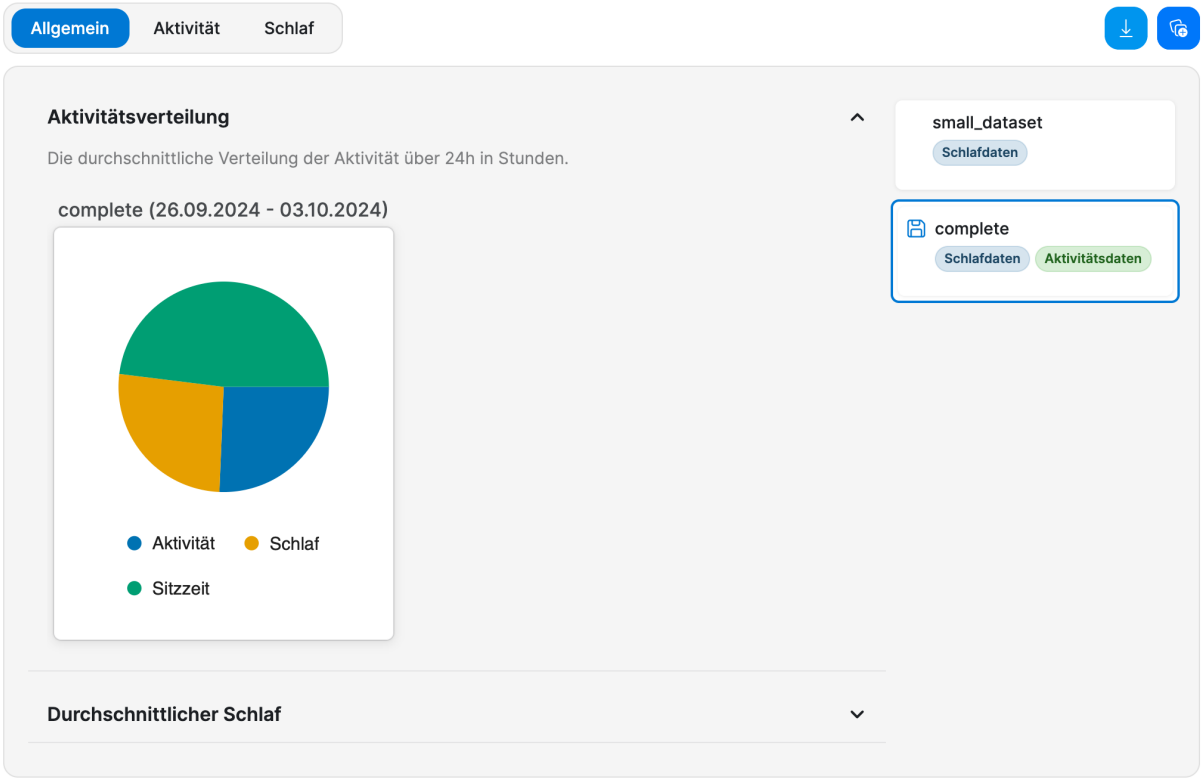
- **Einstellungen:** Das System bietet Konfigurationsmöglichkeiten, die es Nutzern ermöglichen, die Analyseparameter präzise an ihre spezifischen Anforderungen und Zielsetzungen anzupassen.

Analyse Vorgang



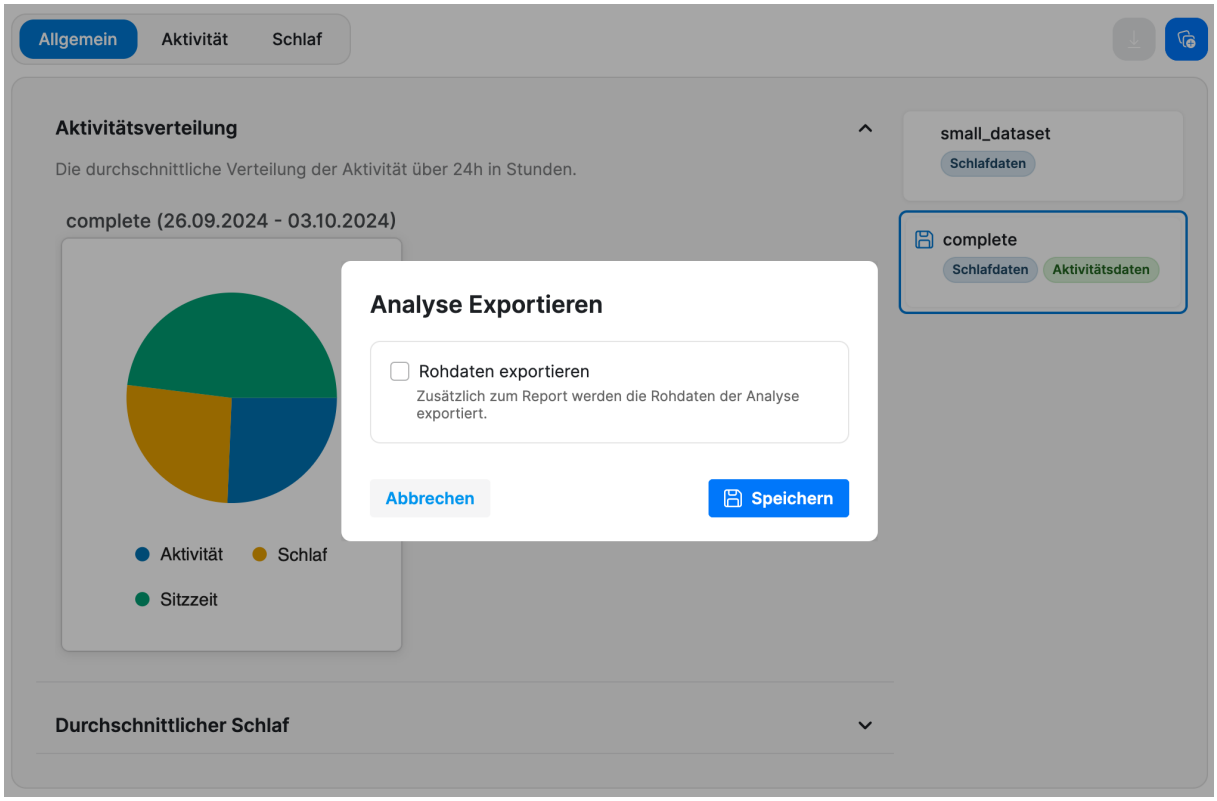
- **Feedback:** Während des Analyseprozesses erhält der Nutzer kontinuierliche Rückmeldungen über den Fortschritt sowie eine präzise Zeitschätzung für die verbleibende Bearbeitungsdauer.
- **Abbruch:** Das System ermöglicht es dem Nutzer, laufende Analysevorgänge jederzeit durch Betätigung der Abbruch-Funktion zu unterbrechen und zu beenden.

Dashboard (mit Daten)



- **Visualisierung:** Nach erfolgreichem Datenimport werden die Informationen mittels grafischer Darstellungen und Diagramme übersichtlich visualisiert.
- **Tab-Verwaltung:** Zur Optimierung der Benutzerübersicht können einzelne Analysen gezielt aktiviert oder deaktiviert werden, wodurch eine Darstellung der relevanten Datenbereiche ermöglicht wird.

Export



- **Exportformate:** Das System unterstützt den Export der durchgeführten Analysen im PDF-Format, wodurch Nutzer bereits abgeschlossene Analysen zu einem späteren Zeitpunkt wieder importieren können, ohne den gesamten Analyseprozess erneut durchlaufen zu müssen. Zusätzlich besteht die Möglichkeit, die zugrundeliegenden Rohdaten separat zu exportieren.

Begründung der Designentscheidung

Die Entscheidung für eine lineare Navigation erfolgte aufgrund der spezifischen Zielgruppe, die überwiegend aus nicht-technischen Nutzern besteht. Dieser Ansatz führt Anwender strukturiert durch den Analyseprozess, ohne sie durch eine Vielfalt an Optionen zu überlasten. Dieses Designkonzept erfüllt die identifizierten Nutzerbedürfnisse optimal durch folgende Aspekte:

- **Zielgruppengerechte Gestaltung:** Da die primären Nutzer keine technisch versierten Personen sind, wurde besonderer Wert auf eine einfache und intuitive Benutzerführung gelegt.
- **Funktionale Reduktion:** Die Konzentration auf wesentliche Funktionalitäten verhindert kognitive Überlastung und ermöglicht eine fokussierte Aufgabenerledigung.
- **Inhaltliche Priorisierung:** Die Analyseergebnisse als wichtigste Information erhalten den grössten Darstellungsraum, während unterstützende Elemente zurückhaltend positioniert

sind.

- **Vertraute Bedienkonzepte:** Das Interface orientiert sich an modernen Webstandards, wodurch Nutzer auf bekannte Interaktionsmuster zurückgreifen können.

Obwohl das Sidebar-Konzept mehr Flexibilität bietet, erwies sich für die definierte Zielgruppe die geführte Benutzerführung als geeigneter, da sie einen effizienten und fehlerfreien Prozessablauf unterstützt.

6.7 Visualisierungen

Die Entwicklung der Graphen und Diagramme war eine zentrale Aufgabe, um die komplexen Analysedaten für die Nutzer verständlich darzustellen.

6.7.1 Anforderungen und Umsetzung

Ein Teil der Visualisierungen war bereits in den funktionalen Anforderungen definiert und wurde vollständig implementiert. Darüber hinaus entwickelten wir zusätzliche Graphen, um den Nutzen für die Anwender zu erhöhen:

- Kombinierte Darstellung von Schlafzeit und Effizienz
- Kuchendiagramm für Zeit im Schlaf- und Wachzustand
- Gestapeltes Balkendiagramm für Aktivitätsverteilung
- Integrierte Analyse von Aktivität, Sitzzeit und Schlaf

Eine vollständige Übersicht findet sich im Anhang (siehe 12.8).

6.7.2 Herausforderungen

Die grösste Schwierigkeit lag darin, dass wir als Entwicklerteam nicht wussten, welche Visualisierungen für die Fachexperten wirklich relevant sind. Die Gespräche mit dem Kunden lieferten leider keine konkreten Vorgaben, sodass wir eigenständig Graphen entwickeln mussten, die aus technischer Sicht sinnvoll erschienen.

6.7.3 Anpassungen durch Feedback

Erst gegen Ende des Projekts erhielten wir Rückmeldungen zu den Visualisierungen. Ein Graph zur Korrelation zwischen Schlafeffizienz und Aktivität wurde wieder entfernt, da diese Parameter in der Praxis selten zusammenhängen.

Die Usability-Tests zeigten, dass der Vergleich von Durchschnittswerten besonders wichtig ist. Daraufhin implementierten wir zwei weitere Graphen speziell für diese Funktion.

6.7.4 Design

Bei der Gestaltung achteten wir auf eine barrierefreie Farbpalette (siehe Abschnitt 6.9). Bei mehr als acht gleichzeitigen Datensätzen kann die Barrierefreiheit jedoch nicht mehr vollständig gewährleistet werden.

Die Diagrammtypen wählten wir je nach Datenart: Balken- und Liniendiagramme für Vergleiche, Kuchendiagramme für Proportionen und gestapelte Balkendiagramme für zeitliche Verläufe von Proportionen.

6.8 Evaluation und Optimierung

6.8.1 Testplanung und -vorbereitung

Für die Evaluation der Desktop-Applikation wurden Usability-Tests geplant. Ziel war es zu überprüfen, ob die implementierten Funktionen den Anforderungen entsprechen und die Anwendung für die Endnutzer intuitiv bedienbar ist.

Zielgruppendefinition

Als Testzielgruppe wurden Mitarbeiter des MOVE-IT (OST) definiert. Diese Nutzergruppe verfügt über grundlegendes Verständnis für Datenanalyse im medizinischen Kontext.

Ursprünglich war geplant, die Tests auch mit medizinischen Fachexperten in einer Klinik durchzuführen, jedoch reichte die verfügbare Zeit nicht aus.

Testszenarien-Entwicklung

Bei der Entwicklung der Testszenarien stand im Vordergrund, dass jedes Szenario eine bestimmte Funktionalität der Anwendung prüft und dabei Rückschlüsse auf die Benutzerfreundlichkeit ermöglicht. Dabei wurde bewusst darauf geachtet, den Testern minimale Unterstützung zu bieten, um die intuitive Bedienbarkeit der Applikation objektiv bewerten zu können.

Die Testszenarien wurden so konzipiert, dass sie den typischen Arbeitsablauf eines medizinischen Fachexperten bei der Analyse von Aktivitätsdaten widerspiegeln.

Testszenarien

Für die Usability-Tests wurden fünf zentrale Szenarien entwickelt:

- **Szenario 1:** Test der grundlegenden Import-Funktionalität von Sensordaten. Der Tester startet die Applikation, importiert eine .bin-Datei, initiiert die Analyse und verschafft sich einen Überblick über die Ergebnisse.
- **Szenario 2:** Überprüfung der Möglichkeit, bereits existierende Analysen im PDF-Format zu importieren und für Vergleiche zu nutzen.

- **Szenario 3:** Test der Analysefunktionen durch Ermittlung spezifischer Daten wie Schlafzeit und Aktivitätsstunden sowie Durchführung von Vergleichen zwischen verschiedenen Datensätzen.
- **Szenario 4:** Überprüfung der Export-Funktionalität durch Speicherung einer Analyse als PDF-Dokument und Kontrolle der entsprechenden Systemmarkierungen.
- **Szenario 5:** Test des Exports von Rohdaten für die Weiterverarbeitung und Überprüfung der Dateistruktur im Dateisystem.

6.8.2 Testdurchführung

Testumgebung und Voraussetzungen

Für die Testdurchführung wurde eine kontrollierte Umgebung mit installierter Testversion der Desktop-Applikation, verfügbaren Testdateien (.bin und .pdf) sowie einem definierten Verzeichnis für Dateiexporte geschaffen.

Methodik und Ablauf

Die Usability-Tests wurden nach der Think-Aloud-Methode durchgeführt. Die Teilnehmer bearbeiteten die fünf Testszenarien in vorgegebener Reihenfolge und äusserten dabei ihre Gedanken und Reaktionen. Die beiden Autoren nahmen eine passive Beobachterrolle ein und dokumentierten die Interaktionen, ohne den Testern Hilfestellung zu geben.

Durchführungsort und -bedingungen

Die Tests fanden vor Ort in den Räumlichkeiten des MOVE-IT an der OST statt. Die vertraute Arbeitsumgebung der Testteilnehmer schuf eine natürliche Testsituation. Die Autoren positionierten sich diskret neben den Testern und führten in ruhiger Atmosphäre Protokoll über die Nutzerinteraktionen.

6.8.3 Ergebnisauswertung

Die Usability-Tests zeigten insgesamt ein positives Ergebnis. Die Applikation erwies sich als weitgehend intuitiv bedienbar, und beide Tester konnten die gestellten Aufgaben erfolgreich lösen. Die Navigationswege waren für die Nutzer verständlich und die grundlegenden Funktionen wurden ohne grössere Schwierigkeiten gefunden. Darüber hinaus fanden die Tester Gefallen an den implementierten Visualisierungen und bewerteten diese als hilfreich für das Verständnis der Daten.

Dennoch offenbarten die Tests drei spezifische Bereiche, in denen Optimierungsbedarf bestand:

Aktivitätsdaten:

- **Problem:** Ein Tester äusserte den Wunsch nach einer Option zum automatischen Entfernen des ersten und letzten Messtages, da diese typischerweise unvollständige Datensätze

enthalten.

- **Massnahme:** Eine entsprechende Clipping-Funktion wurde in die Anwendung integriert, die es Nutzern ermöglicht, diese Randdaten bei Bedarf auszuschliessen.

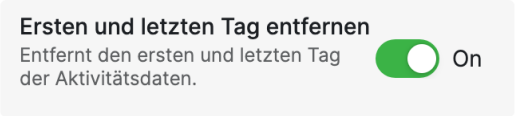


Abbildung 6.7: Option für (De)Aktivierung von Erstem und Letzten Tag

Visualisierungen (Durchschnitte):

- **Problem:** Bei der Betrachtung der Graphen stellte sich heraus, dass für die Benutzer die Durchschnittswerte der Analysen von besonderem Interesse sind. Die ursprüngliche Darstellung fokussierte sich primär auf Einzelwerte und tägliche Verläufe.
- **Massnahme:** Die Unterseite Allgemein wurde entsprechend überarbeitet und zeigt nun die Durchschnittswerte für Schlaf- und Aktivitätsdaten auch über mehrere Analysen hinweg in einer verständlichen und kompakten Form.

PDF-Report Optimierung:

- **Problem:** Es zeigte sich, dass in den exportierten PDF-Reports die Graphen nur wenig Bedeutung für die praktische Nutzung haben. Die Tester fokussierten sich mehrheitlich auf die Durchschnittswerte.
- **Massnahme:** Aufgrund dieser Beobachtung wurde der PDF-Report überarbeitet. Die Visualisierungen wurden entfernt und durch eine kompakte, tabellarische Darstellung der wichtigsten Durchschnittswerte ersetzt. Der neue Report fokussiert sich auf Kennzahlen wie durchschnittliche Schlafzeit, Schlafeffizienz, tägliche Schrittanzahl und Aktivitätsverteilung in einem übersichtlichen Format.

Export:

- **Problem:** Beim Datenexport entstand Verwirrung bezüglich der verfügbaren Ausgabeformate. Ein Nutzer suchte nach einer Möglichkeit, ausschliesslich Rohdaten zu exportieren, ohne das automatisch generierte PDF-Dokument.
- **Massnahme:** Die Export-Funktionalität wurde in der GitHub-Dokumentation beschrieben, sodass zukünftige Nutzer über die verfügbaren Optionen informiert sind.

Anwählen mehrere Dateien:

- **Problem:** Die Bedienung der Mehrfachauswahl stellte sich als nicht selbsterklärend heraus. Ein Tester benötigte Zeit, um zu verstehen, wie mehrere Analysen gleichzeitig

selektiert werden können.

- **Massnahme:** Diese Funktionalität wurde in der GitHub-Dokumentation ergänzt, um Nutzern eine klare Anleitung für die Multi-Select-Bedienung zu bieten.

Die identifizierten Problembereiche waren durchweg lösbar und führten zu konkreten Verbesserungen in der Anwendung.

Das vollständige Testkonzept mit detaillierten Anweisungen und Bewertungsraster ist im Anhang unter 12.4 dokumentiert.

6.9 Accessibility

Da in der Anwendung mit Diagrammen gearbeitet wird, die möglicherweise unterschiedliche Datensätze enthalten, welche durch Farben differenziert werden, wird darauf geachtet, eine möglichst barrierefreie Farbpalette für die Diagramme zu verwenden. Die Farbpalette beschränkt sich auf acht Farben. Werden mehr benötigt, kann die Barrierefreiheit nicht mehr gewährleistet werden. Die Erfahrung zeigt jedoch, dass meist nicht mehr als maximal vier Farben benötigt werden.

6.9.1 Farbenblindfreundliche Farbpalette

Die implementierte Farbpalette basiert auf der wissenschaftlich validierten *Okabe-Ito-Palette* [Won11], die speziell für die häufigsten Formen der Farbenblindheit (Protanopie, Deuteranopie und Tritanopie) entwickelt wurde. Diese Formen betreffen etwa 8% der männlichen und 0,5% der weiblichen Bevölkerung [Bir12] und beeinträchtigen hauptsächlich die Unterscheidung von Rot-Grün-Kombinationen.

Die acht verwendeten Farben (0072B2, E69F00, 009E73, CC79A7, D55E00, 56B4E9, F0E442, 999999) wurden nach perzeptuellen Prinzipien ausgewählt: Rot und Blau bleiben für alle Arten der Farbenblindheit unterscheidbar, während problematische Rot-Grün-Kombinationen vermieden werden.

6.9.2 Compliance mit WCAG 2.1

Die Bedienelemente entsprechen dem Kriterium 1.4.1 (Verwendung von Farbe): UI-Elemente werden nicht ausschliesslich durch Farben unterschieden [W3C18]. Zusätzlich wurde durch die Verwendung der farbenblindfreundlichen Farbpalette sichergestellt, dass auch Diagramme für die meisten Nutzer zugänglich sind.

Kapitel 7

Refactoring R

7.1 Ausgangssituation

Das Kompetenzzentrum MOVE-IT der OST nutzt zur Analyse von Aktivitätsdaten das von der Firma ActiveInsights bereitgestellte R-Ökosystem GENEActiv. Dieses System basiert auf zwei zentralen CRAN-Paketen: **GENEAread** für das Einlesen und die Vorverarbeitung der Sensordaten sowie **GENEAclassify** für die Klassifizierung der Bewegungsmuster.

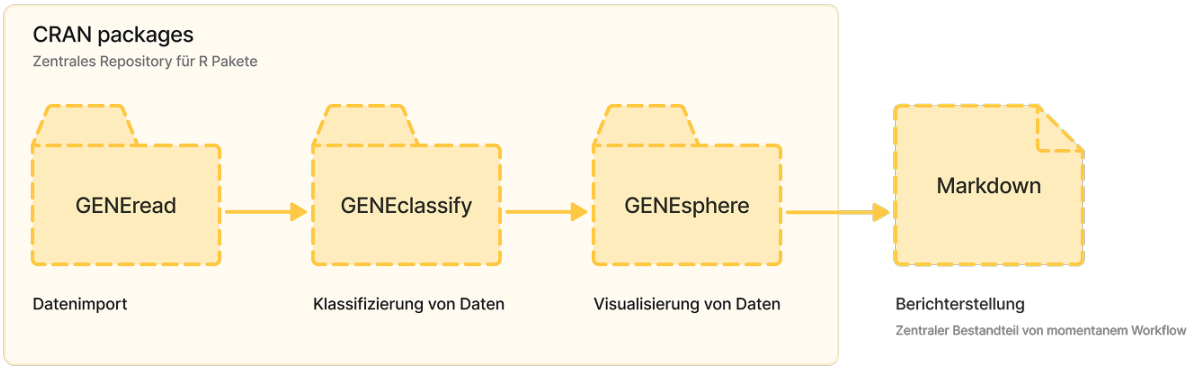


Abbildung 7.1: Workflow für die Datenverarbeitung und Berichtserstellung mit CRAN-Paketen

Das bestehende Analysesystem umfasst zwei separate R-Markdown-Skripte (umgesetzt via `knitr`), die jeweils spezialisierte Auswertungen durchführen.

> **Hinweis:**
> Mittels des `knitr`-Pakets werden die R-Analyseergebnisse direkt in finale Berichtsdokumente (Word-Format) konvertiert und für die Endnutzer aufbereitet.

Das erste Skript konzentriert sich auf die Schlafanalyse, während das zweite die Aktivitätsanalyse der Patienten durchführt. Beide Analyseskripte greifen auf die identischen Kernbibliotheken zurück (GENEAread und GENEAclassify), verarbeiten jedoch die Daten mit unterschiedlichen Parametern.

7.1.1 Daten

Die Sensoren erfassen kontinuierlich verschiedene Messwerte, die zusammen ein umfassendes Bild der körperlichen Aktivität und des Verhaltens liefern.

Erfasste Sensordaten:

- **Tri-axiale Beschleunigungsdaten:** Bewegungsmessungen in drei Raumachsen (x, y, z) zur Erfassung von Körperbewegungen und Lageveränderungen
- **Umgebungslicht:** Kontinuierliche Lichtmessung zur Analyse von Tag-Nacht-Zyklen
- **Temperatur:** Hauttemperaturmessungen zur Unterstützung der Schlafanalyse und Aktivitätserkennung

Technische Eigenschaften:

- **Abtastrate:** 100 Hz ermöglicht präzise Erfassung auch schneller Bewegungen
- **Aufzeichnungsdauer:** Bis zu 7 Tage kontinuierliche Datenerfassung
- **Datenformat:** Strukturierte Binärdateien (.bin) mit Header-Informationen und hexadezimal kodierten Sensormessungen

Die Rohdaten ermöglichen die Berechnung verschiedener abgeleiteter Parameter wie Aktivitätsintensität, Schlaf-Wach-Zyklen und Bewegungsmuster.

7.1.2 Datenverarbeitungspipeline

Die Analyse der GENEActiv-Sensordaten folgt einer Verarbeitungspipeline, die in beiden Analyseskripten implementiert ist. Der Prozess gliedert sich in folgende Hauptphasen:

1. **Datensegmentierung:** Die Rohdaten werden mittels der Funktionen `activity_combine_segment_data()` bzw. `sleep_combine_segment_data()` in Zeitabschnitte unterteilt.
2. **Klassifizierung der Bewegungsmuster:** Jedes Datensegment wird anhand vordefinierter Schwellenwerte und statistischer Parameter klassifiziert.
3. **Schlaf-Wach-Detektion:** Spezielle Algorithmen erkennen Bett- und Aufstehzeiten durch die Analyse charakteristischer Bewegungsmuster und Lichtverhältnisse.
4. **Zustandsneubewertung:** Die initialen Klassifizierungen werden unter Berücksichtigung der erkannten Schlaf-Wach-Zyklen überarbeitet und kontextuell angepasst.
5. **Ergebnisaggregation:** Die klassifizierten Daten werden zu aussagekräftigen Kennzahlen wie Aktivitätsdauer, Schlafeffizienz oder Sitzzeiten zusammengefasst und als strukturierte `.csv`-Datei bereitgestellt.

Sowohl die Schlaf- als auch die Aktivitätsanalyse durchlaufen diese Verarbeitungsschritte.

7.1.3 Herausforderungen

Die Analyse der bestehenden R-Infrastruktur zeigt mehrere kritische Problembereiche auf, die eine Überarbeitung erforderlich machen.

- **Projektrestrukturierung und Coderedundanz:** Die beiden separaten Analyseskripte für Schlaf- und Aktivitätsdaten enthalten nahezu identische Verarbeitungsroutinen und sollen zu einem einheitlichen Skript zusammengeführt werden.
- **Code-Qualität:** Der bestehende Code ist unstrukturiert und schwer zu verstehen. Eine Überarbeitung zur besseren Organisation ist notwendig.
- **Performance-Optimierung der Datenverarbeitung:** Die bestehenden Analyseskripte sind zu langsam und die Performance muss optimiert werden, um im klinischen Alltag praktikabel zu sein.
- **Paket-Repository-Management:** Eine Refactoring erfordert Anpassung an den unterliegenden Libraries, weshalb es notwendig sein wird einen Weg zu finden, die Libraries in einem Custom repository zu hosten.
- **Lokales Deployment:** Da der R-Code weiterhin auf dem Rechner des Nutzers ausgeführt wird, muss eine zuverlässige Lösung entwickelt werden, die ein lokales Deployment erlaubt und eine vorhandene R-Installation nutzt.

7.2 Projektrestrukturierung und -integration:

7.2.1 Entfernen von knitr Paket und redundanter Schritte

Die bestehenden R-Skripte waren eng mit dem knitr-Paket gekoppelt. Da diese Funktionalität für die Desktop-Applikation nicht erforderlich war, musste der Code zunächst von knitr-Abhängigkeiten getrennt werden.

7.2.2 Zusammenführen bestehender libaries

Für eine erfolgreiche Neustrukturierung war es zunächst erforderlich, die Gemeinsamkeiten zwischen den beiden R-Skripten zu identifizieren.

Beide Skripte organisierten ihre Kernfunktionalitäten in einem gemeinsamen functions-Ordner. Die anschliessende Analyse ergab, dass mehrere Dateien in beiden Skripten identisch implementiert waren und somit zusammengeführt werden konnten:

Redundante Dateien:

- `01_library_installer.R`

- `03_naming_protocol.R`
- `05_number_of_days.R`
- `06_bed_rise_detect.R`

Separate Dateien für Aktivitäts- und Schlafanalyse:

- `02_activity_combine_segment_data.R` und `02_sleep_combine_segment_data.R`
- `04_activity_create_df_pcp.R` und `04_sleep_create_df_pcp.R`
- `07_activity_state_rearrange.R` und `07_sleep_state_rearrange.R`
- `08_activity_summary.R` und `08_sleep_summary.R`

7.2.3 Dateistruktur:

Die bestehende Projektstruktur erwies sich als unzureichend organisiert und erforderte eine Neustrukturierung.

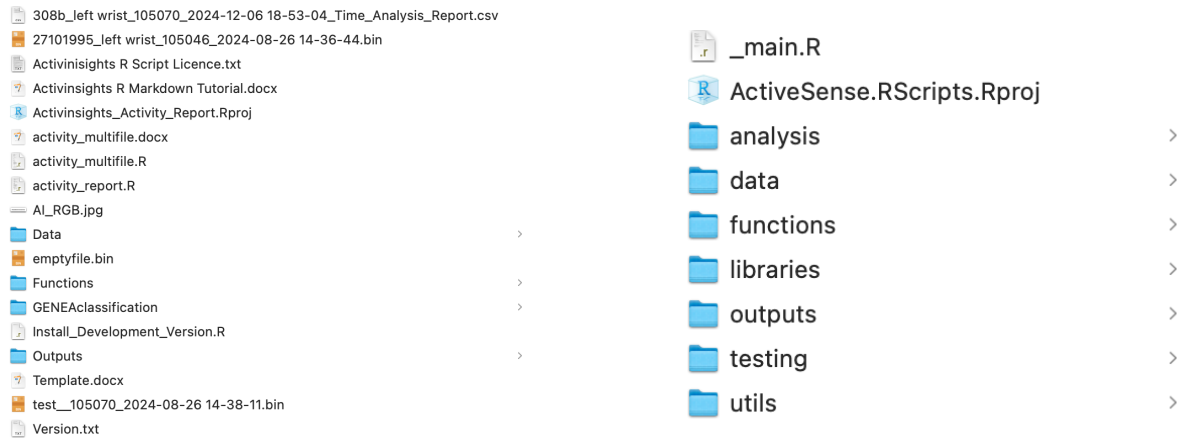


Abbildung 7.2: Dateistruktur: Vorher (links) und Nachher (rechts)

- **main.R:** Zentraler Einstiegspunkt für beide Analyseskripte. Koordiniert die Ausführung der Aktivitäts- und Schlafanalyse.
- **analysis:** Beinhaltet die beiden spezialisierten Analyseskripte `activity_analysis.R` und `sleep_analysis.R`.
- **data:** Unverändert gegenüber der ursprünglichen Struktur. Enthält die zu analysierenden Rohdaten.
- **functions:** Beinhaltet die neu strukturierten Hilfsfunktionen für die Datenverarbeitung, die sich aus der Zusammenführung beider R-Skripte ergaben, wie im Abschnitt Datenverarbeitungspipeline beschrieben. Siehe 7.1.2 und 7.2.2
- **libraries:** Anstatt die Bibliotheken im systemweiten R-Verzeichnis zu speichern, werden

diese nun im Applikationsordner verwaltet. Dies gewährleistet eine bessere Isolation.

- **outputs:** Unverändert gegenüber der ursprünglichen Struktur. Speicherort für die generierten Analyseergebnisse.
- **testing:** Neuer Ordner für Regressionstests zur Qualitätssicherung.
- **utils:** Zusätzlicher Ordner für allgemeine Hilfsfunktionen.

7.2.4 Code-Bereinigung

Die ursprüngliche Codebase enthielt strukturelle Probleme, die eine Weiterentwicklung erschwerten. Der Code wies nicht mehr verwendete Komponenten auf und unklare Verantwortlichkeiten.

Zur Verbesserung wurde eine Neustrukturierung der kritischen Codebereiche durchgeführt. Überflüssige Elemente wurden entfernt, Konfigurationselemente in separate Dateien ausgelagert und wiederkehrende Operationen in Utility-Funktionen überführt.

Die überarbeitete Codebase ist kompakter organisiert und konzentriert sich auf die jeweiligen Kernaufgaben. Diese Aufteilung verbessert die Lesbarkeit und erleichtert künftige Wartungsarbeiten.

Verteilte Konfiguration

- **Problem:** Konfigurationselemente wie globale Variablen waren unstrukturiert über das gesamte Skript verteilt.
- **Änderung:** Alle Konfigurationselemente wurden in einer zentralen config.R-Datei zusammengeführt.

```r
# ==================================
# LIBRARIES
# ==================================

libraries <- c(
  "GENEAread",
  "GENEAclassify"
)

# If use_legacy == TRUE, only use CRAN
if (use_legacy) { options(repos = c(CRAN = "http://cran.r-project.org")) }

install_libraries(libraries)

library(GENEAread)
library(GENEAclassify)

# ==================================
# SOURCE
# ==================================

# functions
source("analysis/activity_analysis.R")
source("analysis/sleep_analysis.R")
source("functions/01_library_installer.R")
source("functions/02_activity_combine_segment_data.R")

# Restliche Implementation
```

Abbildung 7.3: Code-Snippet von config.R

Doppelte Instanziierung

- **Problem:** Bestimmte Objekte wurden an verschiedenen Stellen im Code mehrfach erstellt, was zu Redundanzen führte.
- **Änderung:** Objekte werden nun einmalig instanziiert und wiederverwendet.

Nicht verwendeter Code

- **Problem:** Code, der nicht mehr verwendet wurde und keine Funktion erfüllte.
- **Änderung:** Entfernung des nicht verwendeten Codes zur Verbesserung der Übersichtlichkeit.

Fehlende Dokumentationsstruktur

- **Problem:** Die vorhandenen Kommentare waren unstrukturiert und boten keine klare Orientierung im Code.
- **Änderung:** Implementierung einer einheitlichen Kommentierungsstruktur für bessere Übersichtlichkeit und Navigation.

7.2.5 Timer Funktionalität

Das ursprüngliche Skript enthielt verschiedene Zeitstempel zur Messung der Wall-Clock-Performance. Diese waren jedoch unstrukturiert über den Code verteilt, was die Analyse von Performance-Problemen erschwerte.

Zur Verbesserung wurde eine Timer-Klasse entwickelt, die alle Zeitmessungen zentral verwaltet. Diese sammelt während der Skriptausführung die relevanten Zeitdaten und stellt sie nach Abschluss übersichtlich zusammen.

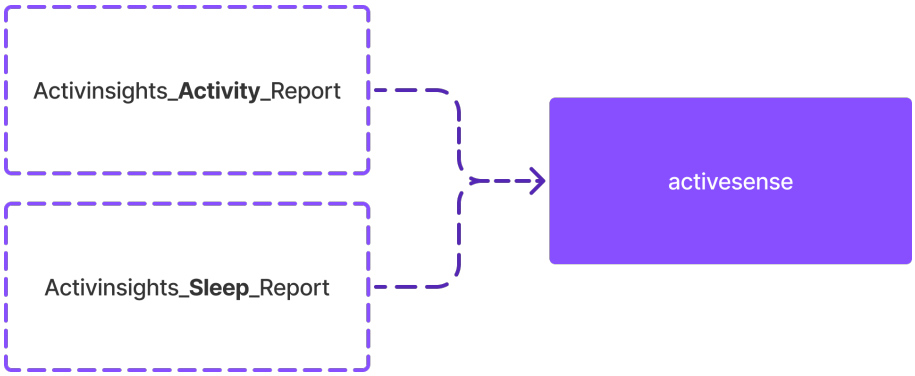


Abbildung 7.4: Fusion beider Projekte

Nach diesem Refactoring konnten beide Analyseskripte erfolgreich zu einer einheitlichen Codebasis zusammengeführt werden.

7.3 Performance-Optimierung

Da die Datenanalyse ineffizient und zeitaufwändig war, haben wir eine Performance-Analyse durchgeführt, um Engpässe zu identifizieren und die Effizienz zu steigern.

7.3.1 Baseline-Messung

Für die Performance-Analyse haben wir das R-Package `profvis` verwendet, welches eine detaillierte und interaktive Visualisierung der Ausführungszeiten ermöglicht.

> **Hinweis:**
> `profvis` visualisiert die CPU-Zeit und zeigt präzise, welche Funktionen die meiste Zeit beanspruchen. Das Tool ermöglicht eine zielgerichtete Optimierung.

Als Referenzdatensatz haben wir eine repräsentative Rohdatei mit 750 MB gewählt, die einen typischen Datensatz darstellt. Diese Datei wird für alle nachfolgenden Performance-Messungen verwendet.

Hinweis:
Alle Performance-Tests wurden auf einem MacBook Pro 15-inch (2019) durchgeführt, ausgestattet mit einem 2.6 GHz 6-Core Intel Core i7 Prozessor und 32 GB Arbeitsspeicher. Diese einheitliche Testumgebung gewährleistet die Vergleichbarkeit aller Messergebnisse.

Die folgenden Zeitangaben messen die CPU-Zeit, nicht die Wall-Clock-Zeit.

Aktivitätsanalyse
Das initiale Profiling der Aktivitätsanalyse ergab eine Gesamtlaufzeit von **13 Minuten, 5 Sekunden**.

Die zeitaufwändigsten Komponenten waren:

- Datenaufbereitung: **83%** der Gesamtzeit
  - Segmentierung: 51%
  - Datenimport: 32%
- Visualisierung: **14%** der Gesamtzeit
- Klassifizierung: **2%** der Gesamtzeit

Schlafanalyse
Die Schlafanalyse benötigte initial **15 Minuten, 40 Sekunden** mit folgender Zeitverteilung:

- Datenaufbereitung: **84%** der Gesamtzeit
  - Segmentierung: 57%
  - Datenimport: 27%
- Visualisierung: **13%** der Gesamtzeit
- Klassifizierung: **2%** der Gesamtzeit

7.3.2 Optimierung

Basierend auf der Performance-Analyse wurden gezielt die identifizierten Bottlenecks adressiert. Die nachfolgenden Optimierungsmassnahmen werden schrittweise durchgegangen, wobei jede neue Verbesserung auf den Ergebnissen der vorherigen aufbaut.

— Entfernung der Legacy-Visualisierung

Da die neue Desktop-Anwendung die Visualisierung selbst übernimmt, werden die ursprünglichen R-basierten Visualisierungsfunktionen nicht mehr benötigt. Diese erste Optimierungsmassnahme konzentrierte sich daher auf die Eliminierung dieser ungenutzten Komponenten.

Identifiziertes Problem: Die bestehenden Visualisierungsroutinen verursachten 14% der Gesamtlaufzeit bei der Aktivitätsanalyse und 13% bei der Schlafanalyse. Diese Funktionen führten redundante Auslesevorgänge der Rohdaten durch, obwohl die generierten Ausgaben für den neuen Anwendungsfall nicht benötigt wurden.

Implementierte Lösung: Vollständige Entfernung der Legacy-Visualisierungsbibliotheken aus dem Analyseprozess, da diese durch die neue Desktop-Anwendung ersetzt wurden.

Erzielte Performance-Verbesserung:

- **Aktivitätsanalyse:** Reduktion von **13m 5s** auf **10m 25s**
- **Schlafanalyse:** Reduktion von **15m 40s** auf **11m 8s**

— Optimierung der StepCounter-Funktion

Nach der ersten Optimierung konzentrierte sich die Analyse auf die Segmentierungsphase, da diese den grössten Anteil der Verarbeitungszeit beanspruchte. Das gezielte Profiling identifizierte die `stepCounter`-Funktion als kritischen Performance-Bottleneck innerhalb der Segmentierung.

Identifiziertes Problem:
Die stepCounter-Funktion enthielt eine ineffiziente Schleifenimplementierung, die iterativ über grosse Datenmengen lief, was sich negativ auf die Laufzeit auswirkte.

```
state = -1                                                  # initialise step state
interval = 0                                                # initialise the interval counter
cadence = numeric(0)                                        # initialise first element of array for intervals
samples = length(filteredData)                              # loop through all samples

for (a in 1:samples) {
  if ((filteredData[a] > hysteresis) && (state < 0)){       # new step started
    state = 1                                               # set the state
    cadence[length(cadence)] = interval + 1                 # write the step interval
    cadence[length(cadence) + 1] = 0                        # initialise to record the next step
    interval = 0                                            # reset the step counter
  } else if ((-1*filteredData[a] > hysteresis) && (state > 0)) { # hysteresis reset condition met
    state = -1                                              # reset the state
    interval = interval + 1                                 # increment the interval
  } else {
    interval = interval + 1                                 # increment the interval
  }
  cadence[length(cadence)] = interval                       # capture last part step
}

cadence = cadence/samplefreq                                # divide by the sample frequency to get seconds
```

Abbildung 7.5: Code-Snippet von altem Step-Counter

Implementierte Lösung:
Die Optimierung erfolgte zweistufig, abhängig vom Analysetyp:

Schlafanalyse: Da Schrittdaten für die Schlafauswertung nicht relevant sind, wurde die gesamte StepCounter-Berechnung aus der Konfiguration entfernt.

```
sleep_datacols = c(
  "UpDown.mean", "UpDown.var", "UpDown.sd",
  "Degrees.mean", "Degrees.var", "Degrees.sd",
  "Magnitude.mean", "Magnitude.var", "Magnitude.meandiff", "Magnitude.mad",
  "Light.mean", "Light.max",
  "Temp.mean", "Temp.sumdiff", "Temp.meandiff", "Temp.abssumdiff",
  "Temp.sddiff", "Temp.var", "Temp.GENEAskew", "Temp.mad",
 "Step.GENEAcount", "Step.sd", "Step.mean", "Step.median"
  "Principal.Frequency.mean", "Principal.Frequency.median"
)
```

Abbildung 7.6: Entfernen von Datacols

Aktivitätsanalyse: Teilweise Neuentwicklung der StepCounter-Logik unter Verwendung vektorisierter Operationen.

```
samples <- length(filteredData)

zone <- ifelse(filteredData > hysteresis, 1L,
               ifelse(filteredData < -hysteresis, -1L, 0L))

prev_zone <- c(0L, zone[-samples])

enters_pos_zone_idx <- which(zone == 1L & prev_zone != 1L)
enters_neg_zone_idx <- which(zone == -1L & prev_zone != -1L)

N_actual_step_starts <- 0
cadence_for_stats_seconds <- numeric(0)

if (length(enters_pos_zone_idx) > 0 || length(enters_neg_zone_idx) > 0) {

  events <- data.frame(
    idx = c(enters_pos_zone_idx, enters_neg_zone_idx),
    type = rep(c(1L, -1L), c(length(enters_pos_zone_idx), length(enters_neg_zone_idx)))
  )
  events <- events[order(events$idx), , drop = FALSE]

  simulated_state <- -1L
  valid_step_start_indices <- numeric(0)

  for (i in 1:nrow(events)) {
    current_event_idx <- events$idx[i]
    current_event_type <- events$type[i]

    if (current_event_type == 1L) {
      if (simulated_state == -1L) {
        valid_step_start_indices <- c(valid_step_start_indices, current_event_idx)
        simulated_state <- 1L
      }
    } else {
      if (simulated_state == 1L) {
        simulated_state <- -1L
      }
    }
  }
```

Abbildung 7.7: Code-Snippet von neuem Step-Counter

Durch den Einsatz vektorisierter Filteroperationen und die strukturierte Vorverarbeitung relevanter Übergangspunkte minimiert die neue Implementierung die Anzahl notwendiger Schlei-

fen und erhöht die Rechenleistung.

Erzielte Performance-Verbesserung:

- **Aktivitätsanalyse:** Reduktion von **10m 25s** auf **4m 8s**

  `StepCounter`-Funktion: **6min 30s** auf **14s**

- **Schlafanalyse:** Reduktion von **11m 8s** auf **5m 15s**

  `StepCounter`-Funktion: **6min 24s** auf **0s**

— Vorberechnung von pagerefs

Die `read.bin`-Funktion ist für das Auslesen der Rohdateien verantwortlich und muss dabei alle Recorded Data-Einträge in der Datei lokalisieren. Zu diesem Zweck verwendet sie die grepraw-Funktion, um für jeden Dateneintrag den exakten Index zu ermitteln und diese Referenzen in einem Vektor namens pagerefs zu speichern.

Abbildung 7.8: Ausschnitt aus Recorded Data

Identifiziertes Problem:
Die Berechnung der `pagerefs` wurde sowohl für die Schlaf- als auch für die Aktivitätsanalyse separat durchgeführt, obwohl beide Analysen dieselbe Rohdatei verwenden. Diese Redundanz führte zu unnötigen Verarbeitungszeiten.

Implementierte Lösung:
Zunächst wurde ein Ansatz evaluiert, bei dem die `grepraw`-Funktion durch eine einfache Offset-Berechnung ersetzt werden sollte. Dieser Ansatz wurde jedoch verworfen, da die Dokumentation explizit darauf hinweist, dass manche Rohdateien nicht konsistenten Offset-Mustern folgen.

Die finale Lösung bestand darin, die `pagerefs`-Berechnung einmalig durchzuführen und das Ergebnis an beide Analyseskripte zu übergeben.

```
my_pagerefs <- getPages(binfile)

if (analyze_activity) {
  activity_analysis(
    binfile = binfile,
    summary_name = getSummaryName("Activity_Summary_Metrics_"),
    pagerefs = my_pagerefs
  )
}

if (analyze_sleep) {
  sleep_analysis(
    binfile = binfile,
    summary_name = getSummaryName("Sleep_Summary_Metrics_"),
    pagerefs = my_pagerefs
  )
}
```

Abbildung 7.9: Ausschnitt aus Recorded Data

Erzielte Performance-Verbesserung:

- **Aktivitätsanalyse:** Reduktion von **4m 8s** auf **2m 46s**
- **Schlafanalyse:** Reduktion von **5m 15s** auf **2m 50s**

7.3.3 Gesamtergebnis der Performance-Optimierungen

Die durchgeführten Optimierungsmassnahmen zeigten Verbesserungen der Analysezeiten. Die nachfolgende Zusammenfassung dokumentiert die kumulativen Ergebnisse aller implementierten Optimierungen im Vergleich zu den ursprünglichen Baseline-Werten.

Erzielte Performance-Verbesserung:

- **Aktivitätsanalyse:** Reduktion von **13m 5s** auf **2m 46s**
- **Schlafanalyse:** Reduktion von **15m 40s** auf **2m 50s**

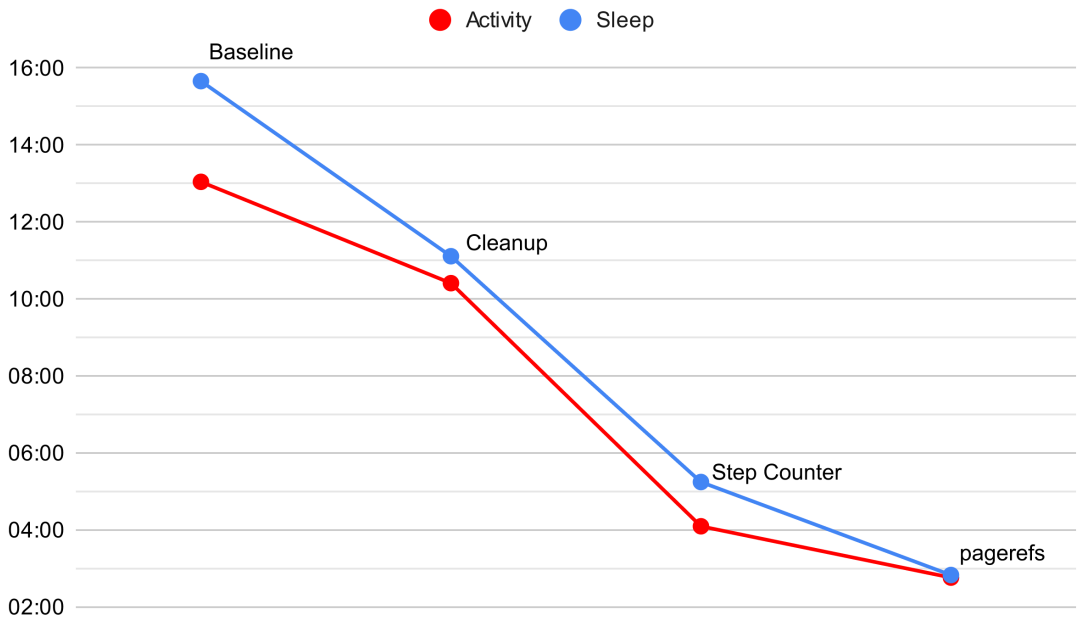


Abbildung 7.10: Auswirkungen der schrittweisen Performance-Optimierungen

Die erzielten Optimierungen konzentrierten sich auf die zugänglichsten Verbesserungsmöglichkeiten mit dem besten Aufwand-Nutzen-Verhältnis. Weitere Performance-Steigerungen würden Ansätze wie die in Kapitel 7.6 beschriebenen Optimierungsmöglichkeiten erfordern.

7.4 Regressions Tests

Die Implementierung der Regressions-Tests wird in Abschnitt 9.5.3 detailliert beschrieben.

7.5 Deployment-Strategien und -entscheidungen

Das lokale Deployment von R stellte eine zentrale technische Herausforderung dar. Die nachfolgenden Abschnitte dokumentieren sämtliche evaluierte Ansätze und erläutern die Entscheidungsfindung für die finale Deployment-Strategie.

Der implementierte Ansatz basiert auf der lokalen Ausführung von R beim Endnutzer.

7.5.1 Einbindung von R

Da sich die Mitlieferung einer vollständigen R-Runtime als übermässig komplex erwies, wurde die Entscheidung getroffen, R als externe Dependency zu behandeln und dessen Installation beim Endnutzer vorauszusetzen.

Die Anwendung führt beim Start der Analyse eine automatisierte Überprüfung der lokalen R-Installation durch. Falls keine kompatible R-Version erkannt wird, erscheint eine benutzerfreundliche Aufforderung zur Installation der erforderlichen R-Umgebung. Diese Validie-

rung gewährleistet, dass alle nachgelagerten Analyseprozesse auf einer funktionsfähigen R-Installation basieren.

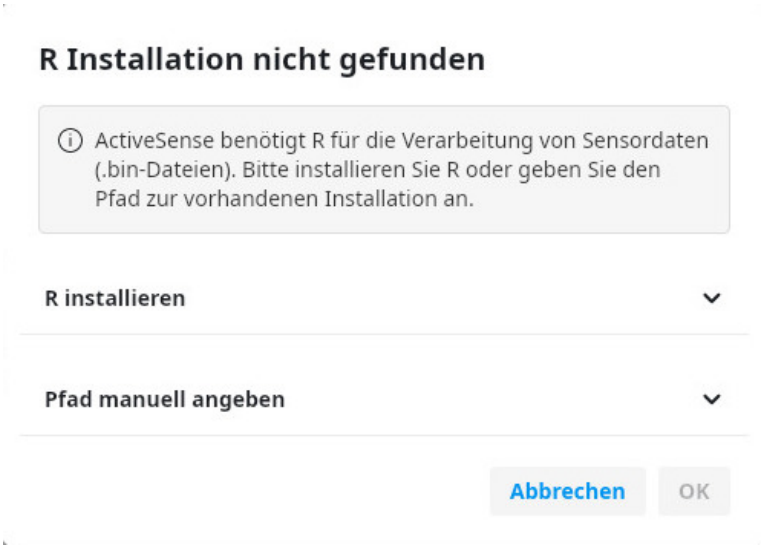


Abbildung 7.11: Popup-Aufforderung zur R-Installation

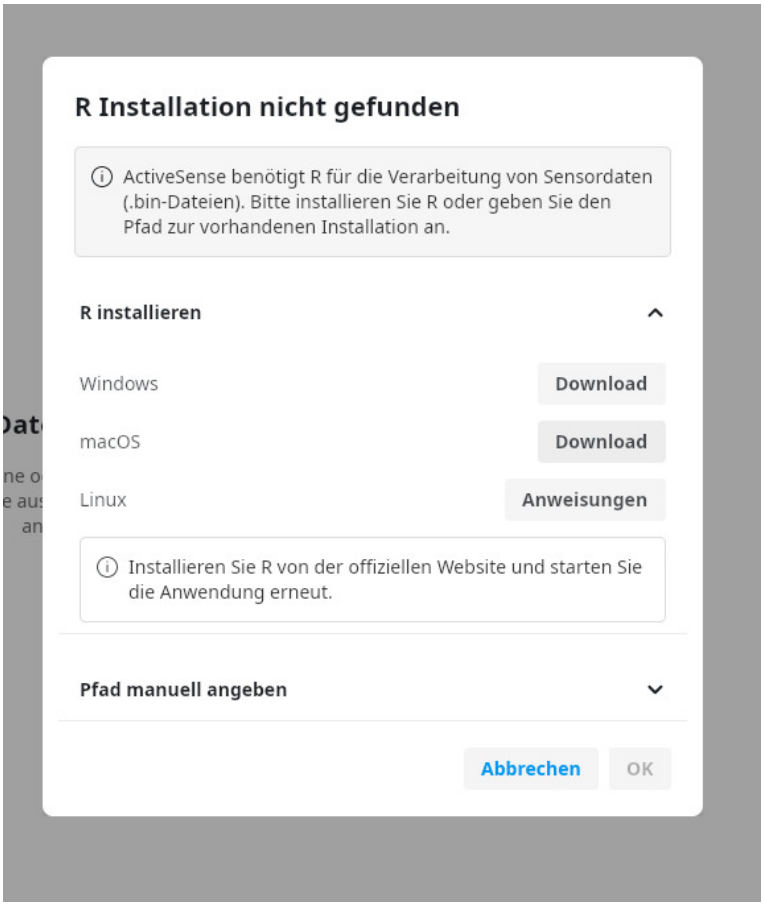


Abbildung 7.12: Expandierte Ansicht der R-Installationsaufforderung

7.5.2 Hosting von Custom-Libraries

Die durchgeführten Modifikationen an den Paketen **GENEAread** und **GENEAclassify**. Zur Performance-Optimierung führten dazu, dass diese nicht mehr über die Standard-CRAN-Repositories bezogen werden konnten. Stattdessen war ein alternatives Hosting für die angepassten Paketversionen erforderlich.

> **Hinweis:**
> CRAN (Comprehensive R Archive Network) ist das offizielle Repository-System für R-Pakete und stellt den Standard-Distributor für verifizierte R-Libraries dar.

Zur Lösung dieses Problems wurde das Open-Source-Projekt R-Universe eingesetzt, welches kostenloses Hosting für benutzerdefinierte R-Pakete anbietet. Das Repository ist unter `https://activesense.r-universe.dev/` verfügbar und ermöglicht die automatisierte Erstellung neuer Paketversionen sowie die Kompilierung plattformspezifischer Binaries bei Änderungen im `main`-branch.

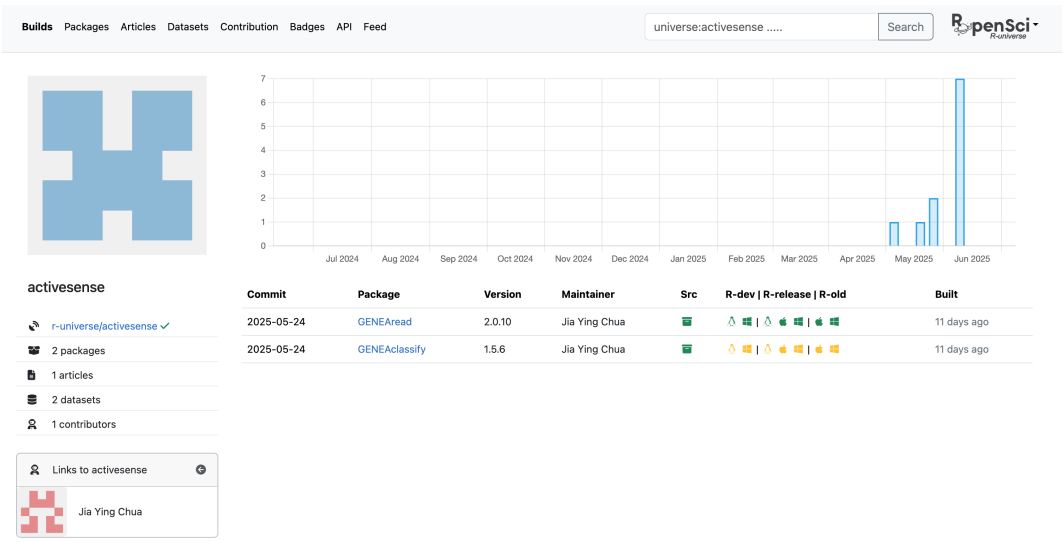


Abbildung 7.13: Snapshot von R-Universe

7.5.3 Einbindung der Dependencies

Die korrekte Verwaltung und Integration der R-Package-Dependencies stellte die komplexeste Herausforderung im gesamten Deployment-Prozess dar und erforderte die Evaluation mehrerer Ansätze.

> **Hinweis:**
> Eine Standardinstallation via `install.packages()` installiert Dependencies im systemweiten R-Library-Verzeichnis der lokalen R-Installation.

Precompiled libraries + renv

Der erste evaluierte Ansatz basierte auf der Vorkompilierung aller Dependencies im Build-Prozess und deren Mitlieferung mit der Anwendung. Zusätzlich wurde das `renv`-Paket eingesetzt, um spezifische Paketversionen zu fixieren und reproduzierbare Environments zu gewährleisten.

> **Hinweis:**
> Das `renv`-Paket ermöglicht projektspezifische R-Environments mit versionsspezifischen Package-Dependencies und reproduzierbaren Installationen.

Die Kombination aus renv und R führt zu systemspezifischen Kompilierungen der Libraries für die jeweilige Kombination aus Betriebssystem, Architektur und R-Version, was sich in folgenden plattformabhängigen Pfadstrukturen manifestiert:

- **Windows:** `/renv/library/R-4.3/x86_64-w64-mingw32/`
- **macOS:** `/renv/library/R-4.3/x86_64-apple-darwin20.0/`
- **Linux:** `/renv/library/R-4.3/x86_64-pc-linux-gnu/`, `/renv/library/R-4.3/x86_64-redhat-linux-gnu/`, `/renv/library/R-4.3/x86_64-conda-linux-gnu/`

Da sowohl die R-Versionen als auch die spezifischen Plattform-Architekturen der Zielumgebungen variieren können, erwies sich dieser Ansatz als zu fragil für eine robuste Deployment-Strategie.

Die Notwendigkeit, für jede mögliche Kombination aus OS, Architektur und R-Version separate precompiled Libraries bereitzustellen, führte zur Verwerfung dieser Strategie.

Run time compiliation + renv

Nach der Verwerfung des vorkompilierten Ansatzes wurde eine Strategie evaluiert, bei der Libraries während der ersten Ausführung heruntergeladen und installiert werden. Auch hier kam das `renv`-Paket zum Einsatz, um spezifische Paketversionen zu fixieren und Reproduzierbarkeit sicherzustellen.

Das Problem entstand bei Paketen mit dem Flag `NeedsCompilation: yes`, die C++-Code enthalten. Bei der Installation dieser Pakete traten Fehler auf, da die erforderlichen Kompilierungstools auf den Zielsystemen nicht verfügbar waren:

- **Windows:** RTools
- **macOS:** Xcode Command Line Tools

- **Linux:** Distributionsspezifische Build-Tools

Die Mitlieferung vollständiger Compiler-Toolchains mit einer Endbenutzer-Anwendung widerspricht grundlegenden Software-Engineering-Prinzipien und ist keine akzeptable Lösung. Aufgrund dieser Überlegungen wurde der renv-basierte Ansatz verworfen und eine pragmatischere Strategie entwickelt.

Runtime Compilation ohne renv

Nach den Erkenntnissen, dass das `renv`-Paket für den Anwendungsfall ungeeignet ist, wurde eine vereinfachte Runtime-Installation-Strategie entwickelt. Der Ansatz basiert auf der Installation von Paketen bei der ersten Programmausführung unter folgenden Voraussetzungen:

- **Verwendung von Binary-Files:** CRAN stellt für Windows und macOS vorkompilierte Binaries zur Verfügung, die ohne Source-Compilation installiert werden können. Bei Linux-Systemen hingegen sind Compiler-Tools erforderlich, da CRAN keine universellen Linux-Binaries bereitstellt und die benötigten Build-Tools je nach Distribution variieren.
- **Integration von Custom-Libraries:** Da renv zuvor für die Einbindung der modifizierten GENEAread und GENEAclassify-Pakete zuständig war, musste eine alternative Integrationsstrategie entwickelt werden. Dies erfolgte durch die Konfiguration des R-Universe-Repositories als zusätzliche Paketquelle.

Zur Automatisierung wurde ein spezialisiertes Installer-Script entwickelt, das die Binary-Installation für Windows und macOS forciert, während es für Linux-Systeme auf die standard Source-Installation zurückgreift. Das Script überprüft automatisch den Installationsstatus und führt nur notwendige Installationen durch, wodurch redundante Operationen vermieden werden.

Diese Lösung gewährleistet eine zuverlässige Package-Installation ohne die Komplexitäten von renv, erfordert jedoch bei Linux-Systemen das Vorhandensein entsprechender Entwicklungstools.

7.6 Weitere Optimierungsmöglichkeiten

Neben den implementierten Performance-Verbesserungen existieren zusätzliche Optimierungsansätze, die Leistungssteigerungen ermöglichen würden, jedoch ausserhalb des aktuellen Projektumfangs lagen.

7.6.1 Serverbasierte Architektur mit REST-API

Eine alternative Systemarchitektur könnte die Verlagerung der Analysefunktionalität auf eine serverbasierte REST-API umfassen. Dieser Ansatz wurde aufgrund von Datenschutzbedenken bei medizinischen Daten verworfen, wäre jedoch durch entsprechende Verschlüsselungsmassnahmen realisierbar.

Vorteile:

- **Vereinfachte Datenbereitstellung::** Die .NET-Anwendung könnte Analyseergebnisse direkt über Web-APIs beziehen, wodurch die lokale Installation der R-Umgebung entfallen würde.
- **Konsistente Performance:** Eine zentrale Serverinfrastruktur würde gleichbleibende Analysezeiten unabhängig von der Hardware-Ausstattung der Endgeräte gewährleisten.

7.6.2 Parallelisierung der Analyseprozesse

Als weitere Optimierungsmöglichkeit wurde ein parallelisierter Ansatz getestet, bei dem sowohl mehrere Dateien gleichzeitig verarbeitet als auch Schlaf- und Aktivitätsanalysen simultan auf separaten Prozessorkernen ausgeführt werden. Diese Implementierung versprach eine erhebliche Reduzierung der Gesamtlaufzeit durch die simultane Verarbeitung aller verfügbaren Analyseprozesse.

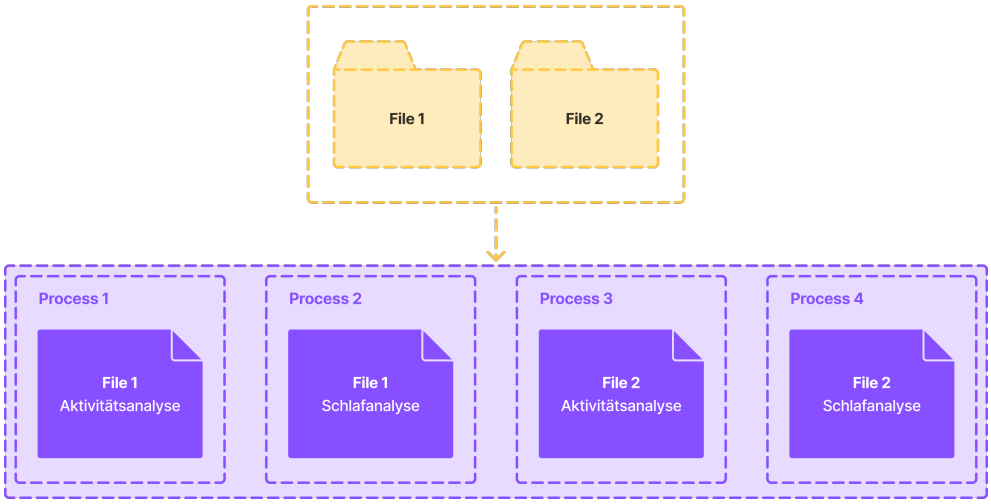


Abbildung 7.14: Parallelisierung

Gründe für die Verwerfung dieses Ansatzes

Der Test des parallelen Ansatzes zeigte jedoch erhebliche Probleme mit dem Arbeitsspeicherverbrauch. Die gleichzeitige Ausführung aller parallelisierten Prozesse vervielfachte den bereits erheblichen Memory-Bedarf, was bei Systemen mit begrenzten RAM-Ressourcen zu kritischen Speicherüberlastungen und Systeminstabilität führte. Wegen dieser Speicherbeschränkungen wurde die Parallelisierung verworfen.

Diese Erfahrung zeigt auch die Vorteile einer serverbasiertem REST-API-Lösung 7.6.1, die über ausreichende Hardware-Ressourcen verfügt und parallelisierte Verarbeitung ohne Client-seitige Speicherlimitierungen ermöglichen würde.

7.6.3 Anpassung der blocksize

Die Datensegmentierung erfolgt bereits in Chunks, um eine Überlastung des Arbeitsspeichers zu vermeiden. Eine weitere Optimierungsmöglichkeit liegt in der Verringerung der `blocksize`. Durch die Reduzierung dieses Parameters kann die Anzahl der Speicherzugriffe reduziert und die Verarbeitungseffizienz gesteigert werden.

```
for (blocknumber in 1: numblocks){
  index = Fullindex[1:min(blocksize, length(Fullindex))]
  Fullindex = Fullindex[-(1:blocksize)]
  proc.file <- NULL

  if (!mmap.load){
    #### 19. Using mmap = F ####
    tmpd <- readLines(fc2, n = (max(index) -lastread) * reclength   )
    bseq = (index - lastread -1 ) * reclength
      if (commasep) vdata = sub(",", ".", vdata, fixed = TRUE)
      voltages = c(voltages, as.numeric(substring(vdata, 17, nchar(vdata))))
    }

  # Restliche Implementation...
```

Abbildung 7.15: Code-Snippet der read.bin-Funktion mit konfigurierbarer Blockgrösse

7.6.4 Import-Funktion

Nach den implementierten Optimierungen zeigt das Profiling, dass die Import-Funktion zum grössten Bottleneck im gesamten Analyseprozess geworden ist. Diese Funktion liest die Rohdaten ein und verarbeitet sie initial, was den Grossteil der Gesamtverarbeitungszeit beansprucht.

Eine Überarbeitung dieser zentralen Funktion bietet das grösste Verbesserungspotenzial. Da alle nachfolgenden Analyseschritte auf dieser Funktion aufbauen, würden Verbesserungen hier direkte Auswirkungen auf die gesamte Systemleistung haben.

7.6.5 Rcpp für gezielte Performance-Optimierung

Das Rcpp-Paket ermöglicht die gezielte Neuimplementierung performance-kritischer Funktionen in C++, wodurch erhebliche Geschwindigkeitssteigerungen in rechenintensiven Bereichen des R-Codes erzielt werden können.

Hinweis:
Das `Rcpp`-Paket ist eine Schnittstelle zwischen R und C++, die es ermöglicht, C++-Code direkt in R-Umgebungen zu integrieren und auszuführen.

Diese Optimierungsstrategie bietet besonders bei schleifen- und berechnungsintensiven Operationen signifikante Verbesserungen gegenüber nativen R-Implementierungen. Der Ansatz erfordert jedoch fundierte C++-Kenntnisse und eine sorgfältige Analyse der bestehenden Codestruktur, um die geeigneten Funktionen für eine Portierung zu identifizieren.

Kapitel 8

Implementation dotnet

8.1 Überblick

Die Implementierung basiert auf dem .NET 9.0 Framework in Verbindung mit Avalonia UI als plattformübergreifendem UI-Framework. Das Projekt besteht aus den Hauptkomponenten Infrastructure, User Interface und Core (Services & Domain). Diese Trennung der Verantwortlichkeiten ermöglicht eine gute Testbarkeit, Wartbarkeit und Erweiterbarkeit der Anwendung. Die Architektur folgt dem MVVM-Pattern und nutzt Dependency Injection für lose Kopplung zwischen den Komponenten. Nachfolgend werden die einzelnen Schichten zusammenfassend beschrieben.

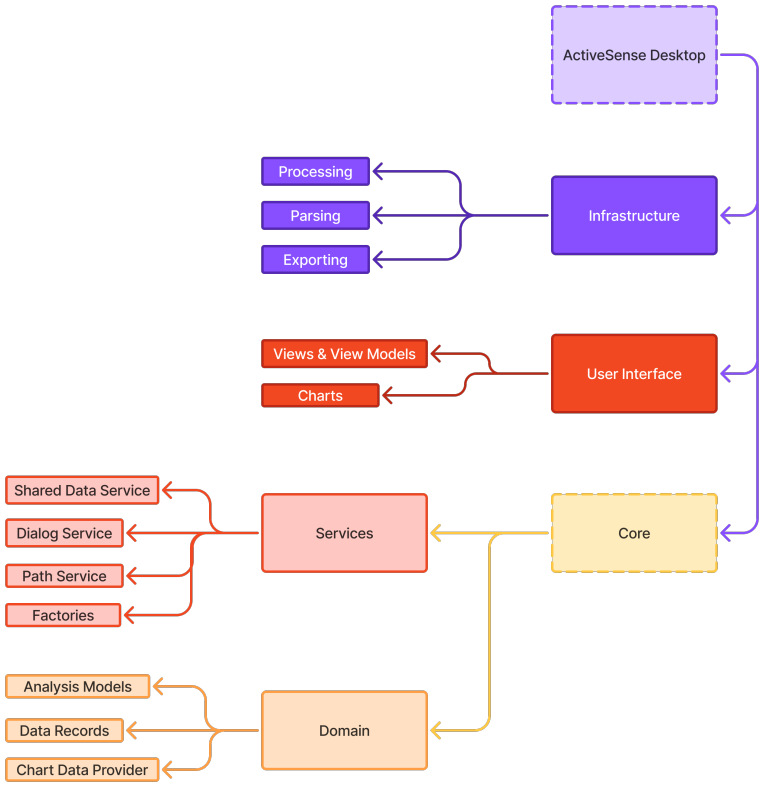


Abbildung 8.1: Projektstruktur der ActiveSense Desktop Anwendung

8.1.1 Infrastructure

Die Infrastructure-Schicht implementiert die technischen Aspekte der Sensordatenverarbeitung, bestehend aus drei Hauptfunktionen, welche mit dem Factory Pattern instanziiert werden. Sie gliedert sich in drei Hauptbereiche:

- **Processing:** Verwaltet die Ausführung der Analyse des entsprechenden Sensors.
- **Parsing:** Konvertiert die Ausgaben der Analyse in .NET-Objekte.
- **Exporting:** Ermöglicht den Export der Analyse.

8.1.2 User Interface

Die Präsentationsschicht implementiert das MVVM-Pattern mit Avalonia UI. Sie umfasst:

- **ViewModels:** Implementieren die Präsentationslogik und das Databinding.
- **Charts:** Generiert interaktive Visualisierungen, welche in den Views dargestellt werden.

8.1.3 Services

Die Service-Schicht stellt komponentenübergreifende Funktionalität sowie Factory-Methoden zur Instanziierung verschiedener Komponenten bereit. Die Factories sind im Projekt separat angeordnet, werden von uns jedoch trotzdem zur Service-Schicht gezählt.

8.1.4 Domain

Die Domain-Schicht repräsentiert die Abbildung der Sensordaten in der Applikation. Das `GeneActiveAnalysis`-Model bildet Daten von GeneActive-Sensoren auf Analyse-Objekte ab und stellt zusätzlich Methoden zur Berechnung statistischer Werte bereit.

8.2 User Interface - View Models

Das Kapitel beginnt mit der Beschreibung der ViewModels, um zu Beginn einen Überblick über die Applikation zu bieten. Die Architektur folgt dem MVVM-Pattern, wobei wir im Implementierungskapitel überwiegend die ViewModels behandeln. Die Views sind aus unserer Sicht weniger relevant, da sie nur wenig Logik enthalten. In ActiveSense existiert eine Hauptseite (`AnalysisPageViewModel`) sowie drei sensorspezifische Unterseiten (Allgemein, Schlaf und Aktivität). Die Seiten für den Import und Export werden in Form von Dialogen gehandhabt und verfügen ebenfalls über ihre eigenen ViewModels. Nachfolgend eine Übersicht über die Hierarchie der ViewModels.

8.2.1 Überblick

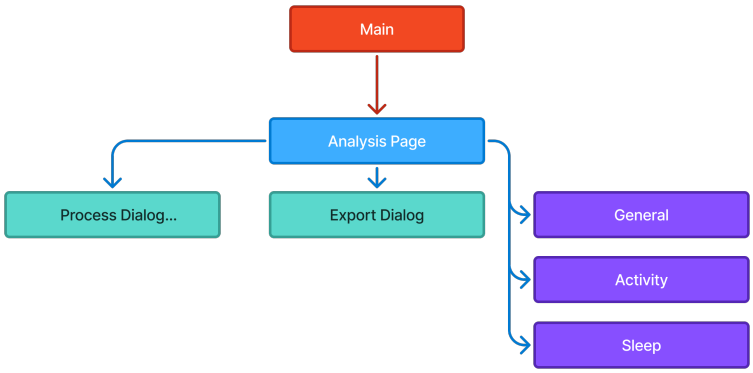


Abbildung 8.2: Hierarchie der ViewModels

8.2.2 MainViewModel

Das MainViewModel bildet die oberste Schicht in der Anordnung der ViewModels. Es instanziiert die AnalysisPage als Hauptseite. Die Entscheidung für diese Hierarchie wurde getroffen, um die Erweiterbarkeit sicherzustellen. Falls in Zukunft mehrere Seiten benötigt werden, kann im MainViewModel einfach eine Navigation hinzugefügt werden.

Automatische Dateibereinigung Das MainViewModel ist ausserdem für das Aufräumen temporärer Daten verantwortlich. Beim Schliessen des Fensters wird geprüft, ob ungespeicherte Analysen (Analysen, welche der Benutzer nicht exportiert hat) vorhanden sind. Falls nicht, werden alle zwischengespeicherten Analysen gelöscht. Gibt es Analysen, die noch nicht exportiert wurden, dann wird ein Dialog angezeigt und der Nutzer kann die Analysen noch speichern.

8.2.3 AnalysisPageViewModel

Das AnalysisPageViewModel implementiert die zentrale Benutzeroberfläche von ActiveSense und koordiniert den gesamten Workflow. Es verwaltet die Instanziierung der Unterseiten, die Darstellung verfügbarer Analysen sowie die Integration von Export- und Import-Funktionalitäten über entsprechende Buttons im Header.

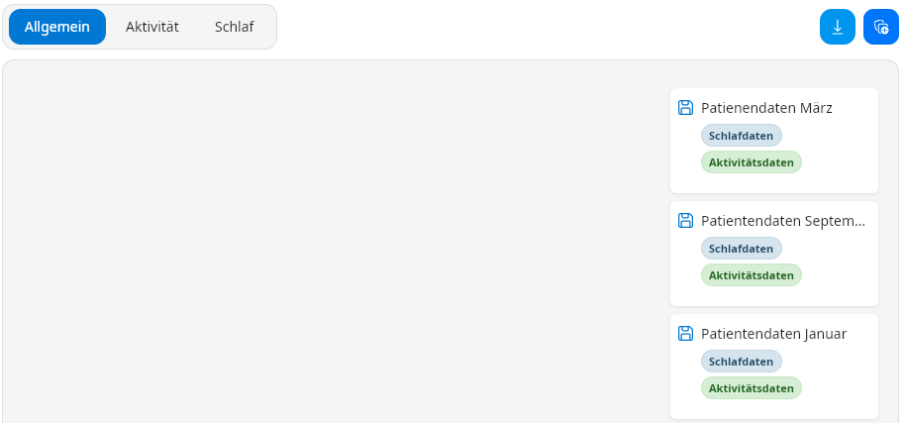


Abbildung 8.3: AnalysisPageViewModel Screenshot

Initialisierung Zu Beginn werden im View-Model mithilfe einer Factory-Methode die für den Sensor relevanten Unterseiten geladen (siehe links oben im Screenshot).

Global verfügbare Analysen Der Benutzer steuert über dieses ViewModel die momentan ausgewählten Analysen. Das ViewModel implementiert dazu einen Service, welcher die Analysen der ganzen Applikation bereitstellt. Über den Service werden zusätzlich Events ausgelöst beim Selektieren von Analysen, auf welche die Unterseiten reagieren können. Details zur Implementierung finden sich im Abschnitt 8.4.2.

8.2.4 ProcessDialogViewModel

Mit einem Klick auf den Import-Button im Header wird ein Dialog zum Import neuer Analysen geöffnet. Auf der linken Seite kann die Analyse konfiguriert werden, während auf der rechten Seite die Dateiauswahl erfolgt.

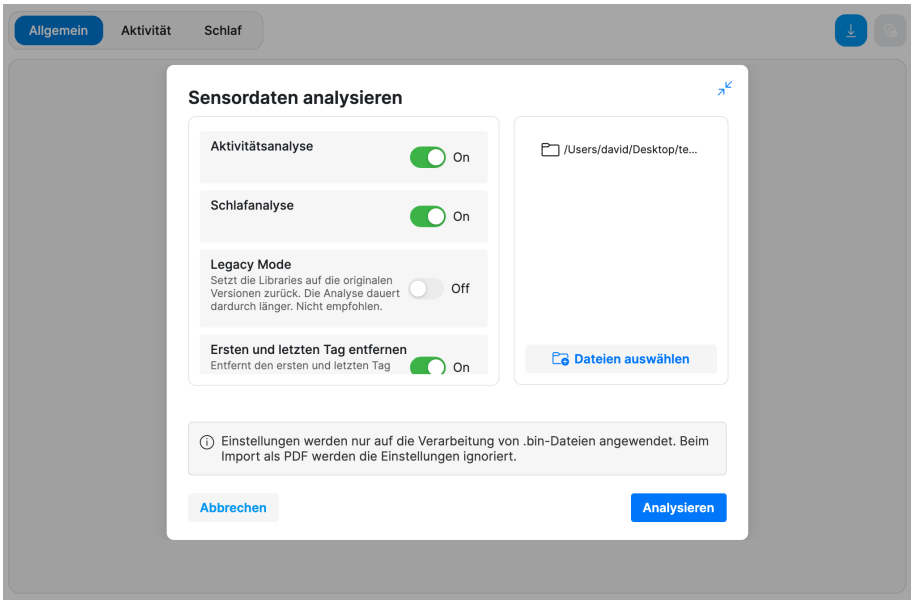


Abbildung 8.4: ProcessDialogView Screenshot

Initialisierung Beim Öffnen des Dialogs instanziiert das ViewModel über eine Factory-Methode einen entsprechenden `SensorProcessor` für den aktuell gewählten Sensortyp. Dieser stellt sowohl die verfügbaren Optionen zur Konfiguration als auch einen Informationstext zur Verfügung. Ebenso wird durch den Processor die Funktion festgelegt, welche beim Klick auf Analysieren ausgeführt wird.

Verarbeitungskoordination Beim Klick auf den Analysieren-Button initiiert das System die sensorspezifische Datenverarbeitung über die `ProcessFiles`-Methode. Diese Methode koordiniert den gesamten Verarbeitungsablauf und führt regelmässige UI-Updates durch, um den Benutzer über den aktuellen Verarbeitungsstand zu informieren. Mehr zur Verarbeitung im Kapitel 8.5.

Parsing Nach erfolgreichem Abschluss der Datenverarbeitung werden die generierten Analyseergebnisse der restlichen Applikation verfügbar gemacht. Hierzu wird mit einer Factory-Methode ein Parser-Objekt erstellt, welches die vom R-Script generierten Ausgabedateien in Analyse-Objekte konvertiert. Diese Objekte werden anschliessend über einen Service zentral gespeichert und stehen damit allen ViewModels zur Verfügung.

Verarbeitung im Hintergrund Es ist möglich, den Dialog während der Verarbeitung zu minimieren, ohne die Analyse zu verlieren. Diese Funktionalität wird durch die Persistierung des ViewModel-Objekts im ermöglicht. Der Verarbeitungszustand bleibt vollständig erhalten, einschliesslich Fortschrittsanzeige. Dies ermöglicht dem Benutzer, vorherige Analysen zu betrachten, ohne auf die Fertigstellung des Prozesses warten zu müssen.

8.2.5 ExportDialogViewModel

Wenn in der Applikation Analysen vorhanden sind, kann die Exportfunktion aufgerufen werden.

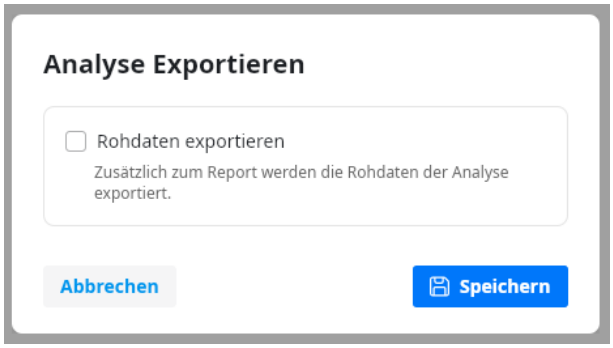


Abbildung 8.5: ExportDialogView Screenshot

Der Dialog zeigt dem Benutzer die Option zur Auswahl des Export-Formats, wobei zwischen Standard-Exporten und erweiterten Exporten mit Rohdaten unterschieden wird.

File Picker Integration Die Auswahl des Speicherorts erfolgt über einen im Code-Behind implementierten FilePicker. Hierfür kommt eine Komponente aus dem `Semi.Avalonia`-Paket zum Einsatz. Momentan ist der Export auf eine einzelne Datei beschränkt. Der gleichzeitige Export mehrerer Dateien wird derzeit nicht unterstützt.

Export Nach erfolgreicher Auswahl des Speicherorts initiiert das System die Export-Operation über eine weitere Factory-Methode. Der instanziierte Exporter verwendet den vom bereitgestellten Pfad zur Speicherung der Exportdatei. Bei erfolgreichem Export wird der Export-Status der Analyse aktualisiert. Dieser Status wird im `MainViewModel` zur Prüfung ungespeicherter Analysen sowie im `AnalysisPageViewModel` zur visuellen Markierung verwendet.

8.2.6 Analysis Pages

Die Analysis Pages sind das zentrale Element der Datenvisualisierung. Sie bestehen aus spezialisierten ViewModels, die jeweils einen bestimmten Aspekt der Sensordaten darstellen. Jedes dieser ViewModels hat Zugriff auf die vom Benutzer selektierten Analysen und reagiert auf Änderungen.

Bestandteile

Die AnalysisPageViewModels folgen alle derselben Struktur, die aus drei wesentlichen Komponenten besteht.

- **Properties:** Jedes Diagramm wird durch eine standardisierte Attributkombination definiert, welche die notwendigen Properties für Data-Binding und die UI-Darstellung bereitstellt.

```
[ObservableProperty] private string _vigorousTitle = "Intensive Aktivität";
[ObservableProperty] private string _vigorousDescription = "Intensive Aktivität pro Tag in Stunden";
[ObservableProperty] private ObservableCollection<BarChartViewModel> _vigorousCharts = [];
[ObservableProperty] private bool _isVigorousExpanded;
```

Abbildung 8.6: Chart Properties Definition im ViewModel

- **Aktualisierung der Diagramme:** Die Aktualisierungslogik wird über einen Event-Handler gehandhabt, welcher auf Änderungen in den ausgewählten Analysen reagiert und eine Methode zur Aktualisierung der Diagramme aufruft.
- **Generierung der Diagramme** Die Chart-Generierung erfolgt durch Instanziierung von Generator-Klassen. Diese Klassen werden mit den erforderlichen Daten befüllt und geben ein ViewModel des jeweiligen Diagramms zurück.

UI-Implementierung und Anpassbarkeit Die Visualisierungen werden manuell in den jeweiligen Views implementiert. Dies bedeutet zwar einen erhöhten Entwicklungsaufwand, ermöglicht jedoch vollständige Kontrolle über das Layout und spezifische Anpassungen für individuelle Diagrammtypen.

Vergleich von Analysen Die Implementierung unterscheidet zwischen zwei Vergleichsmodi, basierend auf der Kompatibilität der Diagramme.

Bei kompatiblen Diagrammen werden mehrere Datenreihen in einem einzigen Diagramm überlagert dargestellt, wodurch direkte Vergleiche und Trendanalysen ermöglicht werden.

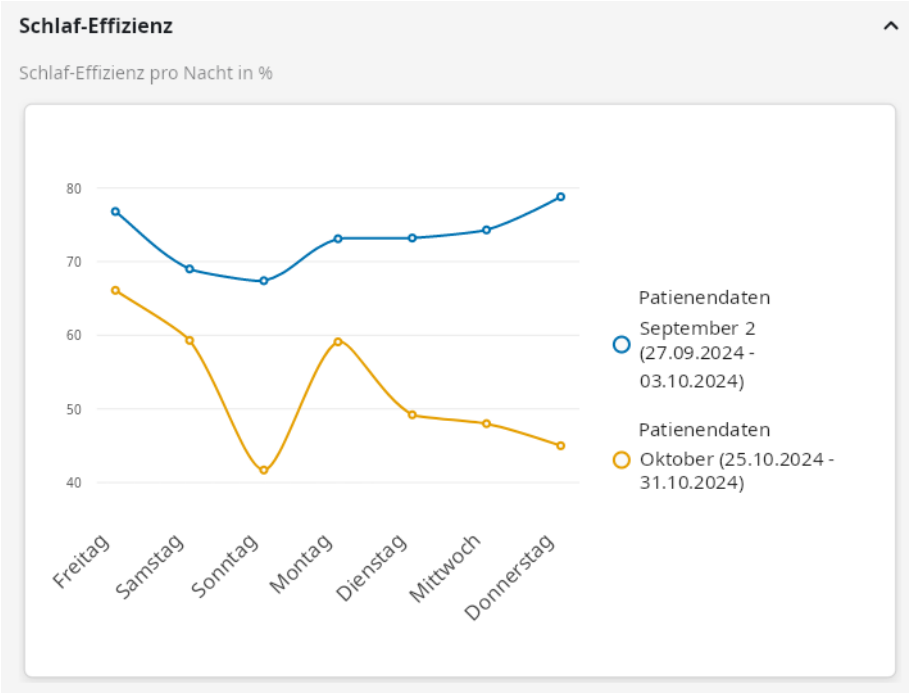


Abbildung 8.7: Überlagerte Darstellung kompatibler Analysen

Bei inkompatiblen Diagrammen generiert das System separate Diagramme für jede ausgewählte Analyse. Vergleiche sind somit trotzdem möglich.

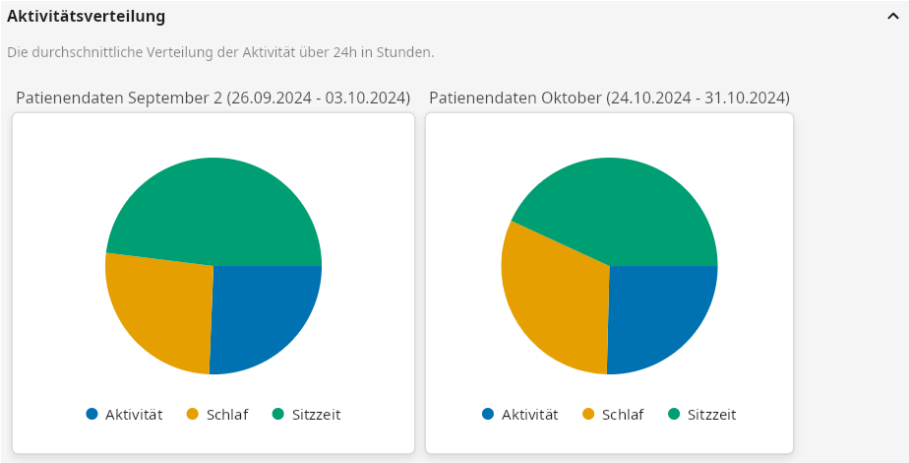


Abbildung 8.8: Separate Darstellung inkompatibler Diagramme

Die Entscheidung zwischen überlagerter und separater Darstellung wird sowohl durch die Anforderungen an die Diagramme als auch durch die technische Umsetzbarkeit bestimmt.

GeneralPageViewModel

Das `GeneralPageViewModel` stellt eine analyseübergreifende Darstellung bereit, welche die wichtigsten Kennzahlen aus Schlaf- und Aktivitätsdaten präsentiert.

ActivityPageViewModel

Das `ActivityPageViewModel` fokussiert sich auf die detaillierte Darstellung von Aktivitätsdaten und bietet Einblicke in Bewegungsmuster und Aktivitätsintensitäten.

SleepPageViewModel

Das `SleepPageViewModel` spezialisiert sich auf die Visualisierung von Schlafdaten und bietet detaillierte Einblicke in Schlafqualität und Schlafdauer.

8.3 User Interface – Charts

Zur Erstellung der Diagramme in den Analyse-Seiten wird ein bestimmtes System genutzt. Das Chart-System nutzt eine Abstraktionsschicht, um interaktive Datenvisualisierungen mit Hilfe von LiveCharts2 zu erstellen. Die Architektur trennt die Datenaufbereitung von der eigentlichen Erstellung der Diagramme. So entstehen flexible und wiederverwendbare Komponenten, die für verschiedene Analysetypen genutzt werden können.

8.3.1 Datenstruktur für Diagramme

Die Klasse `ChartDataDTO` ist die zentrale Datenstruktur für die Diagrammerstellung. Sie wird genutzt, um die Daten der Analyse in ein einheitliches Format zu bringen, das von allen Diagrammtypen unterstützt wird.

Die Klasse enthält drei Hauptbestandteile:

- `Labels`: Beschriftungen der X-Achse.
- `Data`: Die eigentlichen Zahlenwerte.
- `Title` (optional): Ein Titel zur Kennzeichnung einzelner Datenreihen, z. B. bei mehreren Analysen in einem Diagramm.

8.3.2 Generatoren

Das Chart-System implementiert Generator-Klassen für jede Diagrammart. Die Aufgabe der Generatoren besteht darin, die Daten zur Visualisierung aufzubereiten und letztendlich ein Objekt des bestimmten Diagramms zurückzugeben.

Balkendiagramme

Der `BarChartGenerator` ist der vielseitigste Generator im System. Er unterstützt sowohl einfache als auch kombinierte Balken- oder Liniendiagramme, bei denen mehrere Datensätze gleichzeitig dargestellt werden können.

Normalisierung der Daten Da mehrere Datensätze übereinandergelegt werden können, ist es notwendig, alle `ChartDataDTOs` vorab zu vereinheitlichen:

- Zuerst werden alle verwendeten Labels gesammelt und auf eindeutige Werte reduziert.
- Dann wird jeder Datensatz an diese Labels angepasst. Fehlen Daten für bestimmte Labels, wird dort der Wert `null` gesetzt.

So entsteht eine konsistente Struktur für die Darstellung mehrerer Datenreihen in einem Diagramm.

Kreisdiagramme

Der `PieChartGenerator` erstellt Kreisdiagramme. Die Logik dahinter ist relativ einfach: Die übergebenen Daten werden in ein `PieChartViewModel` überführt, das dann im Frontend angezeigt werden kann.

Gestapelte Balkendiagramme

Der `StackedBarGenerator` erweitert die Balkendiagramme um die Möglichkeit, mehrere Werte übereinander in einem einzigen Balken darzustellen. Damit lassen sich zum Beispiel tägliche

Aktivitätsverteilungen gut visualisieren. Auch dieser Generator gibt ein `BarChartViewModel` zurück, das dann in der Benutzeroberfläche angezeigt wird.

8.3.3 Rückgabewert der Generatoren

Jeder Chart-Generator gibt ein ViewModel zurück. Dieses ViewModel enthält alle Daten, die zur Darstellung der Diagramme gebraucht werden. Es gibt zwei Arten von Chart-ViewModels:

- `BarChartViewModel`
- `PieChartViewModel`

Diese ViewModels enthalten keine Logik. Sie dienen lediglich als Datenspeicher für die Darstellung in der Benutzeroberfläche.

8.3.4 Verwendung von LiveCharts2

Für die Erstellung der Diagramme wird die Bibliothek `LiveCharts2` genutzt. Diese moderne Charting-Library bietet eine moderne Lösung für interaktive Visualisierungen.

LiveCharts2 unterstützt MVVM-konformes Arbeiten, ist flexibel anpassbar und enthält Animationen sowie eine Vielzahl von Diagrammtypen. Durch die gute Integration in Avalonia eignet sich die Bibliothek ideal für unsere Anwendung. Auch für eine zukünftige Erweiterung der Diagramme bietet LiveCharts2 eine solide Grundlage, da auch viele Möglichkeiten zur Umsetzung komplexerer Datenvisualisierungen bestehen.

8.4 Core

Die Core-Schicht bildet die grundlegende Datenstruktur. Der Hauptbestandteil ist das Model, welches die verschiedenen Analyseobjekte beschreibt und zentrale Funktionalitäten für die Datenauswertung bereitstellt.

8.4.1 Domain

Eine zentrale Herausforderung bestand darin, das System so zu gestalten, dass zukünftig auch weitere Sensortypen unterstützt werden können. Analyseobjekte sind ein zentrales Element der Anwendung, da viele Mechanismen direkt mit ihnen interagieren. Ziel war es daher, ein flexibles Analyseobjekt zu schaffen, das beliebige Datenstrukturen verschiedener Sensoren speichern kann und dennoch einheitlich im System verwendet werden kann.

Zur Umsetzung dieses Ziels wurde das Interface `IAnalysis` als gemeinsame Basis aller Analysen eingeführt. Analyseobjekte werden im System als `IAnalysis` behandelt, implementieren jedoch zusätzlich sensorspezifische Interfaces, mit denen gezielt interagiert werden kann.

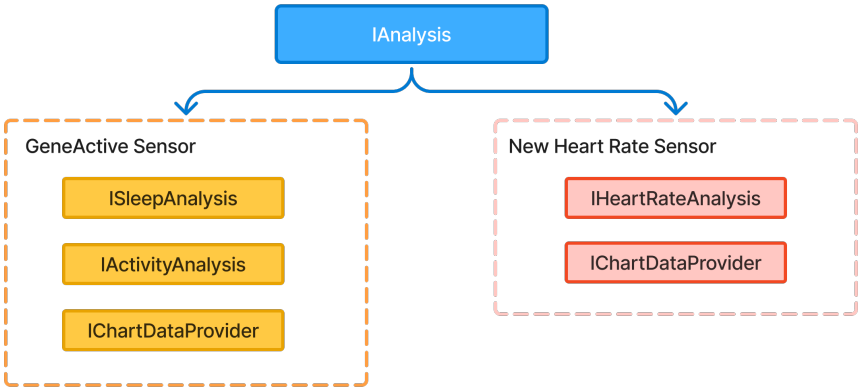


Abbildung 8.9: Analysemodell mit verschiedenen Interface-Implementierungen

Interface Aggregation

Das `GeneActiveAnalysis`-Modell verwendet Interface Aggregation, indem mehrere spezialisierte Interfaces in einer gemeinsamen Implementierungsklasse zusammengeführt werden. Dieses Muster ermöglicht eine flexible Erweiterung des Systems, ohne bestehende Strukturen ändern zu müssen.

Motivation Unterschiedliche Sensoren liefern unterschiedliche Datentypen. Ein Herzfrequenzsensor könnte nur kardiovaskuläre Werte liefern, während ein Aktivitätssensor auch Bewegungs- und Schlafdaten bereitstellt. Durch die gezielte Aggregation von Interfaces kann jedes Analyseobjekt nur jene Funktionalitäten implementieren, die für den jeweiligen Sensortyp relevant sind.

Spezialisierte Interfaces Die Architektur definiert klar getrennte Interfaces für unterschiedliche Analyseaspekte:

- `IActivityAnalysis` – für aktivitätsbezogene Eigenschaften und Methoden
- `ISleepAnalysis` – für schlafbezogene Daten
- `IChartDataProvider` – zur Bereitstellung von Daten für die Visualisierung

```
public interface IActivityAnalysis : IAnalysis
{
    IReadOnlyCollection<ActivityRecord> ActivityRecords { get; }
    double[] StepsPerDay { get; }
    double AverageSedentaryTime { get; }
    // Weitere aktivitätsspezifische Properties
}

public interface ISleepAnalysis : IAnalysis
{
    IReadOnlyCollection<SleepRecord> SleepRecords { get; }
    double AverageSleepTime { get; }
    double[] SleepEfficiency { get; }
    // Weitere schlafspezifische Properties
}
```

Abbildung 8.10: Modularer Aufbau durch Interface-Trennung

Einfache Erweiterbarkeit Neue Sensortypen können problemlos integriert werden, indem sie nur die jeweils relevanten Interfaces implementieren. Ein neuer Herzfrequenzsensor könnte z. B. ausschliesslich `ICardiovascularAnalysis` und `IChartDataProvider` unterstützen.

Polymorphe Verwendung Durch die Aggregation können Analyseobjekte je nach Anwendungskontext polymorph verwendet werden. Ein ViewModel benötigt beispielsweise nur das `ISleepAnalysis`-Interface, um mit schlafspezifischen Funktionen zu arbeiten, unabhängig vom konkreten Sensortyp.

```
private void ProcessAnalysisForSleep(IAnalysis analysis)
{
    if (analysis is ISleepAnalysis sleepAnalysis)
    {
        // Nur schlafspezifische Verarbeitung
        var sleepData = sleepAnalysis.SleepRecords;
        var efficiency = sleepAnalysis.SleepEfficiency;
    }
}

private void ProcessAnalysisForActivity(IAnalysis analysis)
{
    if (analysis is IActivityAnalysis activityAnalysis)
    {
        // Nur aktivitätsspezifische Verarbeitung
        var steps = activityAnalysis.StepsPerDay;
        var sedentary = activityAnalysis.AverageSedentaryTime;
    }
}
```

Abbildung 8.11: Polymorphe Nutzung der Analyseobjekte

8.4.2 Services

Ein weiterer Bestandteil der Core-Schicht sind die Services. Diese implementieren anwendungsweite Dienste, die von mehreren Komponenten genutzt werden.

PathService

Dieser Service übernimmt die zentrale Verwaltung aller systemweiten Pfade sowie die Durchführung wichtiger Dateisystemoperationen. Da die Integration einer externen R-Komponente eine konsistente Pfadverwaltung erfordert, stellt der `PathService` eine fundamentale Komponente der Anwendung dar.

Die vom Service bereitgestellten Pfade sind:

- **Output Directory**: Zentraler Ablageort für alle Analyseergebnisse zur Weiterverarbeitung. Das externe R-Skript speichert fertige CSV-Dateien in diesem Verzeichnis. Nach Abschluss des R-Prozesses durchsucht der zugehörige Parser (siehe Abschnitt 8.6) diesen Ordner systematisch nach unterstützten Dateiformaten.
- **Pfade für die Ausführung von R**:
  - `ScriptBasePath`: Arbeitsverzeichnis für die Ausführung von R-Skripten und Grundlage für relative Pfadangaben.
  - `ScriptInputPath`: Verzeichnis für Eingabedaten, insbesondere GeneActive-Binärdateien, die von R verarbeitet werden. R analysiert alle Dateien, die sich in diesem Ordner befinden.
  - `MainScriptPath`: Pfad zum primären R-Skript (`_main.R`), das die Analyseprozesse steuert.
  - `ScriptExecutablePath`: Pfad zur R-Installation auf dem System, notwendig für die Ausführung von `_main.R`.

Der `PathService` unterscheidet zwischen Entwicklungs- und Produktivumgebung.

ScriptExecutablePath Damit ein R-Skript ausgeführt werden kann, muss auf dem System eine R-Installation vorhanden sein. Die Ausführung der Analyse ist ohne R nicht möglich. Beim Zugriff auf die Property `ScriptExecutablePath` wird abhängig vom Betriebssystem das System nach einer gültigen R-Installation durchsucht. Wird R gefunden, gibt die Methode den Pfad zur ausführbaren Datei zurück. Falls keine Installation gefunden wird, wird eine `FileNotFoundException` ausgelöst, die im Frontend entsprechend behandelt wird (siehe Abschnitt 8.5.3).

SharedDataService

Der SharedDataService implementiert das zentrale Zustandsmanagement der Anwendung und stellt eine reaktive Datenverteilungsschicht für alle analysebezogenen Informationen bereit. Diese Komponente ermöglicht die Kommunikation zwischen verschiedenen ViewModels.

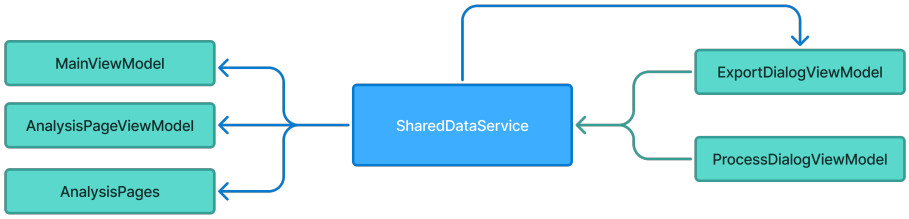


Abbildung 8.12: SharedDataService Architektur und Event-System

Die Implementierung nutzt ObservableCollections für die Verwaltung von allen und aktuell ausgewählten Analysen, wodurch UI-Updates über Data-Binding ermöglicht werden. Event-basierte Benachrichtigungen informieren Komponenten über Änderungen an den Collections und initiieren entsprechende Aktualisierungsmethoden in abhängigen ViewModels.

Code-Beispiel für Datenverteilung Die Implementierung der Datenverteilung zeigt sich in der Subscription-Logik der ViewModels und der Event-Behandlung für Änderungen an der Analyseauswahl:

```
// AnalysisPageViewModel - Datenaktualisierung
partial void OnSelectedAnalysesChanged(ObservableCollection<IAnalysis> value)
{
    _sharedDataService.UpdateSelectedAnalyses(value);
    ShowExportOption = SelectedAnalyses.Count == 1;
}

// SleepPageViewModel - Event Subscription und Reaktion
public SleepPageViewModel(ISharedDataService sharedDataService)
{
    _sharedDataService = sharedDataService;
    _sharedDataService.SelectedAnalysesChanged += OnSelectedAnalysesChanged;
}

private void OnSelectedAnalysesChanged(object? sender, EventArgs e)
{
    UpdateSelectedAnalyses();
    CreatePieCharts();
    CreateTotalSleepChart();
    CreateSleepEfficiencyChart();
}
```

Abbildung 8.13: SharedDataService Code-Beispiel

DialogService

Der `DialogService` verwaltet die Anzeige von Dialogen innerhalb der Applikation und kapselt die Interaktion zwischen Host-ViewModels und den jeweiligen Dialog-Komponenten. Die Architektur des Services nutzt generische Methodensignaturen, um verschiedene Kombinationen von Host- und Dialog-ViewModels flexibel zu unterstützen.

Alle Dialoge basieren auf der Basisklasse `DialogViewModel`. Diese stellt die grundlegende Logik zum Öffnen und Schliessen von Dialogen bereit.

Im `MainViewModel` ist ein `DialogViewModel` als Property hinterlegt. Wenn ein Dialog mit dem `MainViewModel` als Host geöffnet werden soll, setzt der `DialogService` die Dialog-Property im `MainViewModel` auf die gewünschte Dialog-Instanz und setzt die Variable `IsDialogOpen` auf `true`. Durch Data-Binding wird der Dialog dann automatisch in der zugehörigen View des MainViewModels angezeigt.

Code-Beispiel für Dialog-Anzeige

```
var dialog = new WarningDialogViewModel
{
    Title = "Programm beenden?",
    SubTitle = "Ungespeicherte Analysen gehen verloren.",
    CloseButtonText = "Abbrechen",
    OkButtonText = "Schliessen"
};
await _dialogService.ShowDialog<MainViewModel, WarningDialogViewModel>(this, dialog);

if (!dialog.Confirmed) return false;
```

Abbildung 8.14: DialogService Code-Beispiel

8.4.3 Factories

Das Factory-Pattern in ActiveSense ermöglicht die Erstellung sensorspezifischer Komponenten, ohne dass direkte Abhängigkeiten zu den konkreten Klassen bestehen. Die Factory-Klassen verweisen auf den DI-Container, in dem die eigentliche Logik zur Bereitstellung der Klassen implementiert ist. Die Implementierung erfolgt durch die Registrierung von Delegates im Dependency Injection Container während der Anwendungsinitialisierung. Diese Delegates fungieren als Zeiger, die erst zur Laufzeit ausgeführt werden und dabei Zugriff auf den `IServiceProvider` erhalten.

Die konkrete Instanziierung erfolgt in den Delegates über `switch`-Statements, welche abhängig vom Sensortyp die passenden Implementierungen per `GetRequiredService` laden.

Diese Architektur erlaubt die einfache Erweiterung um neue Sensortypen. Es genügt, im jeweiligen `switch`-Block einen neuen `case` hinzuzufügen, die bestehende Factory-Logik muss dabei nicht angepasst werden.

Folgende Factories sind implementiert:

- **SensorProcessorFactory**: Erstellt sensorspezifische Prozessoren zur Datenverarbeitung.
- **ResultParserFactory**: Instanziiert passende Parser zur Umwandlung von Rohdaten in Analyseobjekte.
- **ExporterFactory**: Erzeugt Export-Komponenten zur Ausgabe von Analyseergebnissen.
- **PageFactory**: Erstellt ViewModels für Seiten, basierend auf dem `ApplicationPageNames`-Enum zur Navigation.

8.5 Infrastructure - Processing

Wir widmen uns nun den Hauptkomponenten der Datenverarbeitung. In diesem Kapitel wird erklärt, wie die Analyse des GeneActive-Sensors ermöglicht wird.

8.5.1 Frontend-Integration

Das `ProcessDialogViewModel` 8.2.4 orchestriert den gesamten Verarbeitungsworkflow und stellt die Schnittstelle zwischen Benutzerinteraktion und Backend-Processing dar. Beim Klick auf Analysieren werden folgende Schritte zur Verarbeitung der Dateien mit Hilfe des SensorProcessor-Objekts ausgeführt:

1. **Instanziierung des Processors**: Korrekter SensorProcessor wird auf Basis einer Variable durch eine Factory-Methode instanziiert.
2. **Progress Tracking**: Initialisierung der Fortschrittsanzeige.
3. **File Copy**: Kopieren der Eingabedateien in das entsprechende Verarbeitungsverzeichnis.
4. **Script Execution**: Ausführung der R-Scripts.

8.5.2 Interface

Das Herzstück der Processing-Komponente bildet das `ISensorProcessor` Interface, welches eine flexible und erweiterbare Architektur für die Verarbeitung verschiedener Sensortypen ermöglicht:

```
public interface ISensorProcessor
{
    SensorTypes SupportedType { get; }
    IReadOnlyList<ScriptArgument> DefaultArguments { get; }
    string ProcessingInfo { get; }

    Task<(bool Success, string Output)> ProcessAsync(IList<ScriptArgument> arguments,
        CancellationToken cancellationToken);

    Task<TimeSpan> GetEstimatedProcessingTimeAsync(IEnumerable<string> files,
                                                   IList<ScriptArgument> arguments);

    Task CopyFilesAsync(string[] files, string processingDirectory, string outputDirectory);
}
```

Abbildung 8.15: ISensorProcessor Interface

Aktuell wird im Projekt der GeneActive Sensor unterstützt, dessen Datenauswertung durch ein externes R-Projekt erfolgt. Die abstrakte Definition des Interfaces erlaubt jedoch auch alternative Verarbeitungsansätze für andere Sensoren.

- **SupportedType**: Definiert den von der jeweiligen Implementierung unterstützten Sen-

sortyp. Wird momentan nicht genutzt, könnte jedoch bei einer Erweiterung von Wert sein.

- **DefaultArguments**: Stellt eine Liste von `ScriptArgument` Objekten bereit, welche die Verarbeitungsparameter definieren. Diese umfassen sowohl boolsche Parameter (z.B. Aktivierung von Schlaf-/Aktivitätsanalyse) als auch numerische Schwellenwerte für verschiedene Aktivitätslevel.

- **ProcessingInfo**: Liefert kontextspezifische Informationen zur Datenverarbeitung, die dem Benutzer wichtige Hinweise geben.

- **ProcessAsync**: Führt die asynchrone Datenverarbeitung aus und unterstützt Cancellation Tokens für vorzeitige Abbrüche. Die Methode orchestriert den gesamten Verarbeitungsprozess von der Argumentvalidierung bis zur Skriptausführung.

- **GetEstimatedProcessingTimeAsync**: Berechnet Zeitschätzungen basierend auf Dateigrösse, Systemleistung und gewählten Verarbeitungsparametern.

- **CopyFilesAsync**: Verwaltet das Kopieren von Dateien in die entsprechenden Verarbeitungsverzeichnisse. Diese Methode unterscheidet zwischen zwei Dateitypen (.bin für Verarbeitung, .pdf für direktes Parsing).

8.5.3 Implementierung

Folgend geben wir Einblick in Aspekte, die einen wichtigen Beitrag zum Verständnis der Funktionsweise liefern.

Verarbeitungs-Workflow

Zur besseren Verständlichkeit wurde der Prozess anhand eines Flussdiagramms veranschaulicht. Wichtig hierbei ist die zu Beginn ausgeführte Copy-Files-Methode. Diese Methode kopiert die Files vom Benutzer in die richtigen Verzeichnisse, in denen sie dann verarbeitet werden können.

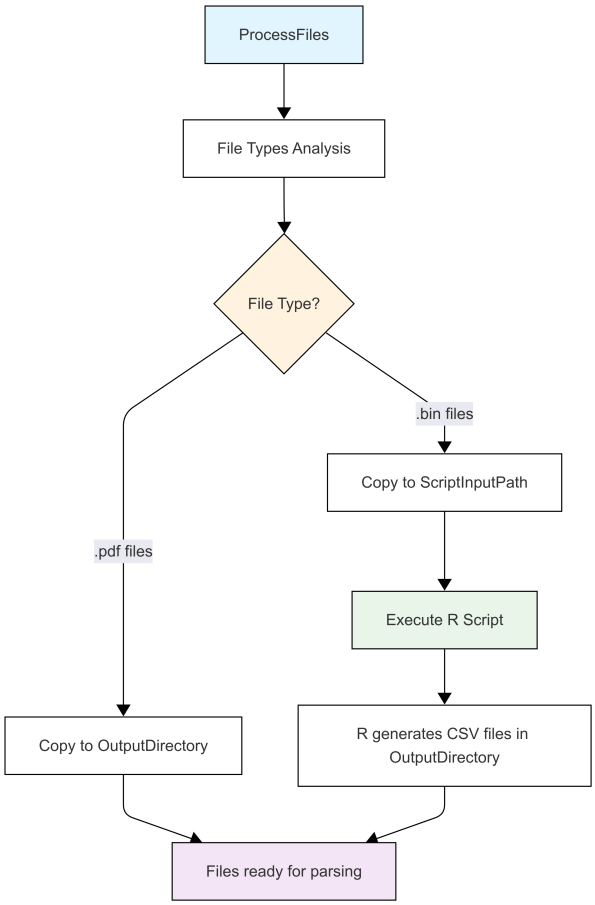


Abbildung 8.16: Flussdiagramm zur Veranschaulichung des Processors

`.bin`-Dateien werden zur weiteren Verarbeitung in das Eingabeverzeichnis der R-Komponente verschoben. PDF-Dokumente hingegen werden ohne weitere Verarbeitung direkt im Output-Verzeichnis abgelegt.

Der Output-Ordner wird im Anschluss vom Parser ausgewertet (siehe Abschnitt 8.6).

ProcessAsync

Die `ProcessAsync` Methode ist verantwortlich für das Ausführen der Analyse. Sie definiert zunächst die verarbeitungsrelevanten Pfade über den `IPathService`. Anschliessend werden die übergebenen `ScriptArgument` Objekte in ein Command-Line-Format konvertiert und mit dem Output Directory zu einer Argumentliste kombiniert. Abschliessend wird das R-Script als separater Systemprozess mit umfassendem Error-Handling und Cancellation-Support ausgeführt.

Exception Handling

Alle während der Verarbeitung auftretenden Exceptions werden abgefangen und für benutzerfreundliche Darstellung an das Frontend übertragen. Die Implementierung unterscheidet drei Exception-Typen mit spezifischen Behandlungsstrategien:

Die `OperationCanceledException` wird abgefangen und direkt an das Frontend weitergegeben, wo die Abbruch-Logik entsprechend behandelt wird. Diese Exception tritt auf, wenn der Benutzer die Verarbeitung vorzeitig beendet.

Die `FileNotFoundException` signalisiert eine fehlende R-Installation (siehe Abschnitt 8.4.2) und wird gezielt im Frontend abgefangen. In diesem Fall wird ein spezifischer Dialog angezeigt, der dem Benutzer Download-Links und Installationsanleitungen für verschiedene Betriebssysteme bereitstellt.

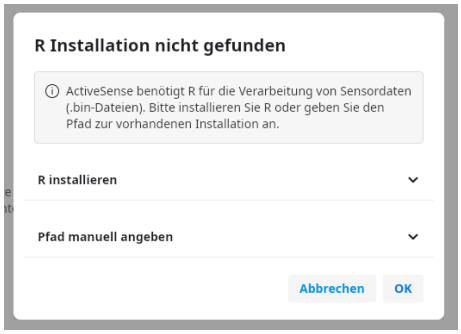


Abbildung 8.17: R-Installation Dialog

Allgemeine `Exception` Typen werden geloggt und als failed Result mit detaillierter Fehlermeldung zurückgegeben. Diese Exceptions umfassen unerwartete Verarbeitungsfehler, Dateisystem-Probleme oder Fehler bei der Ausführung von R.

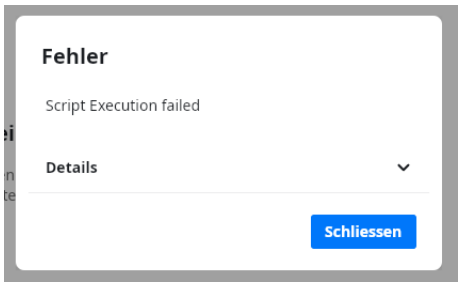


Abbildung 8.18: Allgemeiner Fehler Dialog

Alle Exception-Kategorien werden im Frontend durch einheitliche Dialog-Mechanismen präsentiert. Bei Script-Execution-Fehlern wird der komplette R-Script-Output zur Verfügung gestellt und kann über "Details"für Debugging-Zwecke angezeigt werden.

ProcessingTimeEstimator

Eine aus Nutzersicht sehr wichtige Komponente ist die Zeitschätzung. Die `ProcessingTimeEstimator` Klasse berechnet Verarbeitungszeiten durch die Kombination von Benchmark-Daten mit gerätespezifischer Kalibrierung. Die Zeitberechnung basiert auf einer 767.7 MB Datei mit einer gemessenen Verarbeitungszeit von 388 Sekunden auf einem Referenzgerät. Dies resultiert in einer Referenz-Geschwindigkeit von 1.98 MB pro Sekunde.

Gerätekalibrierung: Das System ermittelt die Performance eines Geräts durch die Berechnung von SHA256-Hashes. Die Zeit zur Berechnung der Hashes wird ins Verhältnis zur benötigten Zeit des Referenzgeräts gesetzt, wodurch ein gerätespezifischer Leistungsfaktor berechnet wird. Dieser wird anschliessend auf die Verarbeitungsgeschwindigkeit angewendet.

Die finale Zeitberechnung kombiniert Dateigrösse, gerätespezifischen Leistungsfaktor und argumentbasierte Anpassungen, um eine Zeitschätzung abzugeben.

8.6 Infrastructure - Parsing

Die Parsing-Komponente ordnet sich in der Verarbeitungsstruktur hinter dem Processing ein und bildet das zentrale Element für das Einlesen der Analysen. Zusätzlich stellt sie die verfügbaren Analyse-Seiten für die Benutzeroberfläche bereit. Das ResultParser-System wird an zwei Stellen der Anwendung eingesetzt: im AnalysisPageViewModel zur Initialisierung der Unterseiten sowie im ProcessDialogViewModel zur Einlesung der verarbeiteten Sensordaten.

8.6.1 Frontend-Integration

Das Parsing der Sensordaten erfolgt als finaler Schritt im `ProcessDialogViewModel` (siehe Abschnitt 8.2.4) nach der erfolgreichen R-Script-Ausführung. Die resultierenden Analyse-Objekte werden zentral im SharedDataService gespeichert und stehen somit allen ViewModels der Anwendung zur Verfügung.

8.6.2 IResultParser Interface

Das IResultParser Interface definiert die Schnittstelle für alle sensorspezifischen Parser-Implementierungen.

```
public interface IResultParser
{
    ApplicationPageNames[] GetAnalysisPages();
    Task<IEnumerable<IAnalysis>> ParseResultsAsync(string outputDirectory);
}
```

Abbildung 8.19: IResultParser Interface Definition

Das Interface stellt zwei wesentliche Funktionalitäten bereit: Die GetAnalysisPages-Methode liefert ein Array von ApplicationPageNames zurück, welche die verfügbaren Analyse-Seiten für die Hauptnavigation definieren. Für GeneActive-Sensoren umfasst dies die Seiten Allgemein, Schlaf und Aktivität. Die ParseResultsAsync-Methode implementiert die asynchrone Verarbeitung der Ausgabedateien und konvertiert diese in Analysis-Objekte für die weitere Verwendung in der Anwendung.

8.6.3 GeneActiveResultParser Implementation

Die GeneActiveResultParser-Klasse orchestriert die Verarbeitung verschiedener Dateiformate durch spezialisierte Sub-Parser. Die Implementierung unterscheidet zwischen zwei primären Datenquellen: exportierte PDF-Dateien, die bereits codierte Analysis-Objekte enthalten, und Verzeichnisse mit CSV-Dateien, die direkt von R-Scripts generiert wurden.

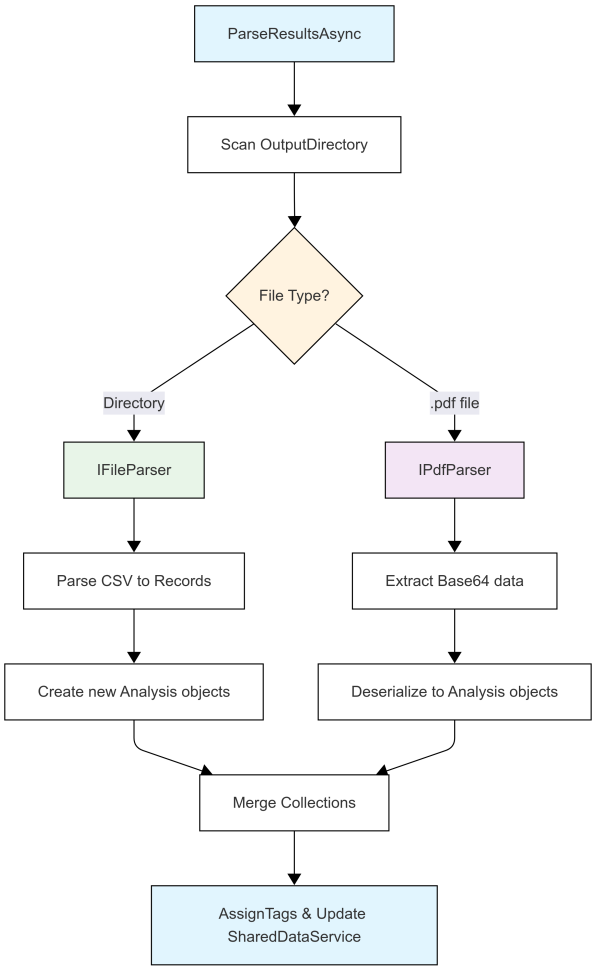


Abbildung 8.20: Flussdiagramm zur Veranschaulichung des Parsers

Der Parser nutzt eine mehrstufige Verarbeitungsstrategie: Zunächst werden PDF-Dateien über den IPdfParser verarbeitet, der eingebettete Base64-kodierte Analysis-Daten extrahiert (mehr dazu im Export). Diese PDF-Dateien stammen von vorherigen ActiveSense-Exporten (siehe Abschnitt 8.7.4) und enthalten vollständige Analyseergebnisse in serialisierter Form. Parallel dazu verarbeitet der Parser Verzeichnisse im Ausgabeordner, die CSV-Dateien mit Rohdaten der R-Script-Verarbeitung enthalten.

CSV-Verarbeitung

Die CSV-Verarbeitung erfolgt durch den IFileParser, der Verzeichnisse iterativ durchsucht und dabei zwischen verschiedenen Datentypen unterscheidet. Jedes Verarbeitungsverzeichnis

kann abhängig von den gewählten Script-Argumenten sowohl Schlaf- als auch Aktivitätsdaten enthalten. Der HeaderAnalyzer identifiziert den Analysetyp (Schlaf/Aktivität) anhand des CSV-Headers und leitet die entsprechende Verarbeitung ein. Das R-Script erstellt für eine Analyse einen Ordner, der dann alle CSV-Dateien enthält. Der FileParser erstellt für jedes Verzeichnis ein neues GeneActiveAnalysis-Objekt und befüllt dieses schrittweise mit den gefundenen Daten. Nach der erfolgreichen Verarbeitung werden alle Analyse-Objekte mit entsprechenden Tags versehen, um zu kennzeichnen, welche Analysen welche Daten enthalten.

8.7 Infrastructure - Exporting

Der Export erfolgt als letzter Schritt in der Verarbeitungs-Pipeline und implementiert die Funktionalität zur Ausgabe von Analyseergebnissen in verschiedenen Formaten. Das Export-System unterstützt sowohl PDF-Berichte als auch Exporte mit Rohdaten.

8.7.1 Frontend-Integration

Der Export-Prozess wird über das `ExportDialogViewModel` (siehe Abschnitt 8.2.5) gesteuert. Das Frontend bietet eine Benutzeroberfläche zur Auswahl des gewünschten Export-Formats und des Speicherorts. Das ExportDialogViewModel koordiniert den gesamten Export-Workflow durch die Integration der ExporterFactory, welche sensorspezifische Exporter-Implementierungen bereitstellt.

8.7.2 IExporter Interface

Das IExporter-Interface definiert die Schnittstelle für alle sensorspezifischen Exporter-Implementierungen.

```
public interface IExporter
{
    Task<bool> ExportAsync(IAnalysis analysis, string outputPath, bool exportRawData);
}
```

Abbildung 8.21: IExporter Interface Definition

ExportAsync akzeptiert ein IAnalysis-Objekt, einen Ausgabepfad und einen booleschen Parameter für die Rohdaten-Inklusion. Diese API-Gestaltung ermöglicht sowohl einfache PDF-Generierung als auch Multi-Format-Exporte durch denselben Methodenaufruf.

8.7.3 Implementation

Die Implementierung setzt sich aus mehreren spezifischeren Komponenten zusammen:

- `IPdfReportGenerator` generiert mithilfe von QuestPDF PDF-Reports für den Export.
- `ICsvExporter` konvertiert Analyse-Objekte in CSV-Dateien.

- `IArchiveCreator` verpackt Report und Rohdaten in ein Zip-Archiv.

Die Klasse implementiert eine bedingte Export-Strategie basierend auf dem exportRawData-Parameter. Bei Standard-Exporten wird ausschliesslich der PDF-Bericht erstellt, während erweiterte Exporte einen mehrstufigen Prozess durchlaufen.

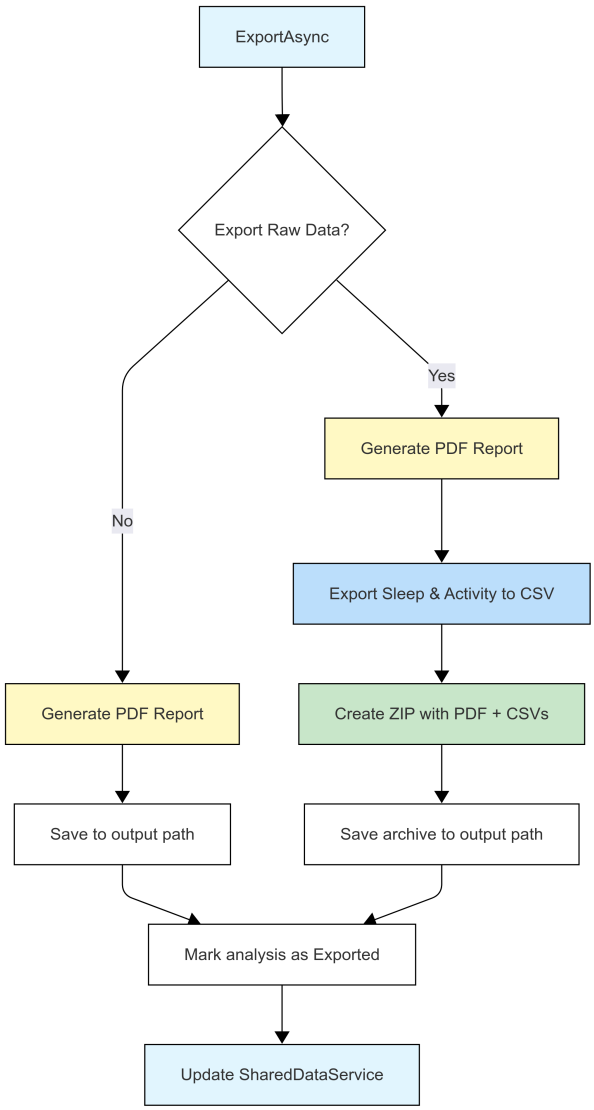


Abbildung 8.22: Flussdiagramm zur Veranschaulichung des Exporters

8.7.4 PDF-Berichtsgenerierung

Die PDF-Berichtsgenerierung erfolgt durch den `PdfReportGenerator`, welcher `QuestPDF` für die Erstellung des Dokuments verwendet. Der Generator erstellt auf Basis des zu exportierenden Analyse-Objekts einen Bericht, der die wichtigsten Durchschnittswerte und Kennzahlen enthält (siehe Anhang 12.5). Zusätzlich wird das Analyse-Objekt selbst in das Dokument eingebettet, um dieses zu einem späteren Zeitpunkt wieder importieren zu können.

Einbettung der Analyse

Damit exportierte Dokumente wieder importiert werden können, wird das Analyse-Objekt wie folgt eingebettet:

- Das Analyse-Objekt wird als JSON serialisiert.
- Die JSON-Daten werden anschliessend in einen Base64-String umgewandelt.
- Der Base64-String wird im PDF hinterlegt.

Im Dokument wird der Base64-String zwischen zwei speziellen Markern eingebettet:

`ANALYSIS_DATA_BEGIN` ...Base64-String... `ANALYSIS_DATA_END`

Die Marker dienen dazu, den eingebetteten String beim späteren Import eindeutig identifizieren zu können. Der gesamte Text wird dabei in transparenter Schrift dargestellt und ist somit auf dem PDF-Dokument nicht sichtbar.

8.7.5 CSV-Export

Der Rohdaten-Export wird durch den `CsvExporter` realisiert, welcher Analyse-Objekte in das CSV-Format konvertiert. GeneActive-Analyseobjekte können sowohl Schlaf- als auch Aktivitätsaufzeichnungen enthalten. Ein einzelner Aufzeichnungsdatenpunkt wird dabei als `ActivityRecord` oder `SleepRecord` Objekt gespeichert.

Ein Analyse-Objekt enthält jeweils ein Array von `SleepRecord`- und `ActivityRecord`-Einträgen. Die Implementierung nutzt die Bibliothek `CsvHelper`, um diese Arrays in das CSV-Format umzuwandeln. Der Header wird dabei automatisch anhand der Attributnamen der Record-Klassen generiert.

8.8 Allgemein

In diesem Abschnitt möchten wir kurz die Herangehensweisen für bestimmte Probleme beschreiben. Informationen dazu sind zum Teil bereits in den vorherigen Kapiteln zu finden.

8.8.1 Logging

Das Logging wird mit dem Paket `SeriLog` umgesetzt. Dieses bietet eine einfache Möglichkeit, Logs in der Konsole auszugeben. Logs werden im Terminal in folgendem Format ausgegeben:

> [09:32:23 INF] Parsing PDF file: /home/user/Documents

Aktuell haben wir zwei Arten von Logs implementiert:

1. `Information`-Logs: Diese protokollieren wichtige Ereignisse und unterstützen beim Debugging, indem sie Einblicke in die Funktionsweise der Anwendung geben.
2. `Error`-Logs: Diese werden bei Exceptions erzeugt und enthalten entweder die Exception-Message oder eine benutzerdefinierte Fehlermeldung.

Durch diesen Ansatz konnten wir während der Entwicklung Fehler leichter identifizieren und die Anwendung besser testen.

8.8.2 Exception Handling

Das Exception Handling ist insbesondere für die drei Hauptkomponenten Processor, Parser und Exporter von zentraler Bedeutung. An jeder dieser Stellen können Fehler auftreten. Es ist daher essenziell, diese Fehler gezielt zu behandeln, um Abstürze der Anwendung zu vermeiden und den Nutzer verständlich über die Art des Fehlers zu informieren.

Grundsätzlich verfolgen wir den Ansatz, jede kritische Funktion mit einer `try-catch`-Klausel abzusichern. Dies wird sowohl in den ViewModels als auch innerhalb der Komponenten-Implementierungen umgesetzt.

In den Komponenten selbst werden auftretende Fehler in der Regel mit einer Fehlermeldung versehen und anschliessend erneut geworfen. Die ViewModels fangen diese Exceptions ab und zeigen entsprechende Dialoge mit der Fehlermeldung an.

Dabei unterscheiden wir zwischen zwei Arten von Fehlern:

- **Spezifische Exceptions**: Diese werden gezielt abgefangen und meist mit einem eigenen, angepassten Dialog behandelt (siehe Abschnitt 8.5.3).
- **Generelle Exceptions**: Diese werden mit dem `InfoDialogViewModel` im Frontend als Dialog angezeigt. Dabei kann sowohl eine kurze Fehlermeldung als auch eine detaillierte Beschreibung über eine `ExtendedMessage` eingeblendet werden.

8.8.3 Dependency Injection

Die Applikation implementiert Dependency Injection mittels `Microsoft.Extensions.DependencyInjection`. Die Konfiguration erfolgt in `App.axaml.cs` während der Anwendungsinitialisierung.

Service-Registrierung

Die Services werden nach funktionalen Kategorien organisiert und registriert:

- **Factories**: Factory-Pattern-Implementierungen für Service-Erstellung (`ResultParserFactory`, `SensorProcessorFactory`, `PageFactory`, `ExporterFactory`)
- **Core Services**: Grundlegende Services wie `ISharedDataService` für Datenmanagement und `IPathService` für Dateisystemoperationen
- **Infrastructure Components**: Services für Prozessausführung, Export-Funktionalitäten und Parsing-Operationen
- **ViewModels**: UI-Komponenten

Service Lifetimes

Die Applikation verwendet zwei Lifetimes für Objekte:

- **Singleton**: Shared Services, Factories und Infrastructure-Komponenten
- **Transient**: ViewModels (ausser MainViewModel)

8.9 Erweiterung um neuen Sensor

Die folgenden Schritte sind erforderlich, um einen neuen Sensor vollständig zu implementieren:

8.9.1 Logik für das Teilen des Sensortyps implementieren

Momentan ist der Sensortyp in allen ViewModels, die ihn benötigen, hart codiert. Hier muss eine Lösung gefunden werden, wie der Sensortyp allen ViewModels verfügbar gemacht wird.

Empfehlung: Erweiterung des `SharedDataService`, sodass darüber auch der Sensortyp gemanagt wird. Zusätzlich könnte im Header der Applikation ein Dropdown zur Wahl des Sensors implementiert werden.

8.9.2 Sensor-Typ definieren

Der neue Sensortyp wird zur `SensorTypes` Enum in der Datei `ActiveSense.Desktop/Enums/SensorTypes.cs` hinzugefügt. Die Enum wird um den entsprechenden Wert für den neuen Sensortyp erweitert.

8.9.3 Model implementieren

Eine neue Analysis-Klasse wird erstellt, die mindestens das Interface IAnalysis implementiert.

8.9.4 Sensor Processor implementieren

Ein Processor für den neuen Sensor wird erstellt, der das `ISensorProcessor` Interface implementiert.

8.9.5 Result Parser implementieren

Ein Parser für die Ausgabe der Sensor-Verarbeitung wird erstellt, der das `IResultParser` Interface implementiert.

Da der Result Parser auch die Analysis-Seiten enthält, ist es wichtig, auch die entsprechenden Analysis-Seiten zu implementieren. Falls der Sensor dieselben Daten wie GENEActiv liefert, können die bestehenden Analyse-Seiten genutzt werden. Allerdings muss beachtet werden, dass das Model dann auch alle entsprechenden Interfaces implementiert.

8.9.6 Exporter implementieren

Ein Exporter für den neuen Sensor wird implementiert, der das `IExporter` Interface implementiert.

8.9.7 Dependency Injection konfigurieren

Service Registration in App.axaml.cs

Alle neuen Komponenten werden im DI-Container registriert. Hier ist es wichtig, dass die neuen Komponenten in die Factory-Methoden im DI-Container aufgenommen werden:

- Processor-Factory um den neuen Sensor erweitern
- Parser-Factory um den neuen Parser erweitern
- Exporter-Factory um den neuen Exporter erweitern
- Page-Factory um eventuelle neue Analysis-Seiten ergänzen

8.10 Deployment

ActiveSense nutzt die Cross-Platform-Fähigkeiten von .NET 9, um native Executables für Windows, macOS und Linux bereitzustellen.

8.10.1 Installer-Erstellung

Die Anwendung ist als Self-Contained Single-File Deployment konfiguriert. Für jede Plattform existieren eigene Build-Skripte, die sowohl das Executable als auch einen Installer generieren:

Windows: Der Installer wird mit Inno Setup erzeugt. Das PowerShell-Skript kompiliert automatisch die Setup-Datei und berücksichtigt das Manifest.

macOS: Ein vollständiges App-Bundle wird erstellt und in ein signiertes DMG mit Drag & Drop-Installation verpackt.

Linux: Ein `tar.gz`-Archiv mit Installationsskript sorgt auch unter Linux für eine einfache Installation.

Bereitstellung von R

Entgegen der ursprünglichen Planung wird R nicht direkt mit der Anwendung ausgeliefert. Stattdessen haben wir uns entschieden, die Installation von R dem Nutzer selbst zu überlassen. Dies bietet mehrere Vorteile: Sollte die Applikation zukünftig um Sensoren erweitert werden, die keine R-Komponente benötigen, entfällt für diese Nutzer die Notwendigkeit, R überhaupt zu installieren.

Auch aus Wartungs- und Update-Gründen erschien es uns praktikabler, nicht für jedes Betriebssystem eine eigene R-Distribution mitzuliefern. Stattdessen zeigt die Applikation betriebssystemspezifische Hinweise zur R-Installation an (siehe Abschnitt 8.5.3), sodass der Nutzer einfach zur offiziellen Downloadseite weitergeleitet wird. Dies reduziert den Wartungsaufwand und sorgt gleichzeitig für mehr Flexibilität in der Weiterentwicklung.

8.10.2 Asset-Handling

Zur Build-Zeit werden relevante Dateien automatisch eingebunden:

- R-Skripte aus `ActiveSense.RScripts/` ins Output-Verzeichnis
- Avalonia-Ressourcen und Icons als eingebettete bzw. plattformspezifische Assets

Konfiguriert ist dies in `ActiveSense.Desktop.csproj`.

8.10.3 CI/CD mit GitHub Actions

Die CI/CD-Pipeline automatisiert Builds, Tests und Releases über GitHub Actions:

- **Builds:** `dotnet.yml` für Unit-Tests
- **Installer:** `installer.yml` erstellt die plattformspezifischen Installer. Die Pipeline nutzt hierfür Windows-, macOS- und Linux-Runner zur Erstellung der Installer.
- **Releases:** Bei Git-Tags (`v*`) werden automatisch alle Installer generiert und in einem neuen GitHub Release als Draft gespeichert.

8.11 Unit Tests

Das Projekt implementiert eine umfassende Testsuite auf Basis des NUnit-Frameworks, die dem klassischen Arrange-Act-Assert-Prinzip folgt und alle kritischen Komponenten der Anwendung

abdeckt. Die Testarchitektur verwendet das Moq-Framework zur Erstellung von Mock-Objekten. Details zur Testabdeckung befinden sich im Abschnitt 9.4.5.

8.11.1 Arrange-Act-Assert Pattern

Das Arrange-Act-Assert Pattern bildet die Grundlage aller Tests, wobei in der Arrange-Phase Testdaten und Mock-Konfigurationen vorbereitet werden. Die Act-Phase führt die zu testende Methode aus, während die Assert-Phase die Ergebnisse validiert. Zusätzlich werden Mock-Verifikationen durchgeführt, um sicherzustellen, dass Dependencies korrekt aufgerufen wurden.

8.11.2 Mock-Framework und Dependency Isolation

Mock-Objekte werden über das Moq-Framework erstellt, insbesondere für Services wie `IPathService`, `ILogger`, `IPdfParser` und `ISharedDataService`. Diese Mocks werden über `Setup`-Methoden konfiguriert, um spezifisches Verhalten zu definieren, und über `Verify`-Methoden überprüft, um sicherzustellen, dass Methoden mit korrekten Parametern und der erwarteten Häufigkeit aufgerufen wurden.

8.11.3 Dependency Injection in Tests

Die Dependency Injection wird in den ViewModel-Tests über die `ServiceCollection` simuliert. Hierbei werden sowohl echte Services als auch Mock-Objekte registriert, um realistische Testbedingungen zu schaffen. Factory-Pattern werden über `Func<T>`-Delegates gemockt, wodurch die Erstellung verschiedener Komponenten basierend auf Enum-Werten getestet wird.

8.11.4 SetUp & TearDown

Jeder Test nutzt `SetUp`- und `TearDown`-Methoden zur Umgebungsvorbereitung. In `SetUp` werden temporäre Verzeichnisse über `Path.GetTempPath()` erstellt, Mock-Objekte konfiguriert und Dependencies injiziert. `TearDown` sorgt für die Bereinigung des Dateisystems und die Freigabe von Service-Providern.

8.11.5 Validierung und Assertion-Strategien

Die Testvalidierung erfolgt über NUnits `Assert.That`-Syntax. Für numerische Werte wird z.B. `Within(0.01)` verwendet, um Gleitkomma-Ungenauigkeiten zu berücksichtigen. Mock-Verifikationen nutzen `Times.Once`, `Times.Never` oder `Times.Exactly(n)` zur Überprüfung der Aufrufhäufigkeit.

8.12 Verbesserungsvorschläge

In diesem Abschnitt listen wir Punkte auf, die uns im Verlauf der Arbeit aufgefallen sind, für deren Umsetzung jedoch im Rahmen der Bachelorarbeit keine Zeit mehr blieb.

8.12.1 Model-Refactoring

Die Verwendung von Interface-Aggregation hat sich als gute Lösung erwiesen, um das Model flexibel und erweiterbar in der gesamten Applikation einzusetzen. Das `GeneActive`-Model ist mit knapp 400 Zeilen Code jedoch relativ umfangreich. Eine sinnvolle Massnahme wäre es, die Funktionalitäten der Interfaces `ISleepAnalysis`, `IActivityAnalysis` und `IChartDataProvider` in eigene Klassen auszulagern und innerhalb des `GeneActiveAnalysis`-Models zu delegieren. Dies würde die Wartbarkeit und Lesbarkeit des Codes verbessern.

8.12.2 Fehlerbehandlung

Der aktuell implementierte Mechanismus zur Fehlerbehandlung funktioniert grundsätzlich zuverlässig. Jedoch führt das Werfen von Exceptions in bestimmten Situationen zu unerwünschtem Verhalten: Wird beispielsweise eine fehlerhafte Datei innerhalb eines Batch-Prozesses analysiert, wird die gesamte Analyse abgebrochen, obwohl weitere Dateien erfolgreich verarbeitet werden könnten. Zukünftig könnte hier ein robusterer Ansatz gewählt werden, bei dem statt Exceptions strukturierte Resultat-Objekte zurückgegeben werden, die sowohl erfolgreiche als auch fehlerhafte Analysen enthalten. Im Frontend könnten diese entsprechend differenziert dargestellt werden, um dem Nutzer klar zu kommunizieren, welche Dateien erfolgreich verarbeitet wurden und welche nicht.

8.12.3 Dynamische Seitennavigation

Der `ResultParser` speichert, welche Analyse-Seiten für einen bestimmten Sensor verfügbar sind. Dieses Konzept ermöglicht prinzipiell eine dynamische Anzeige relevanter Seiten. Derzeit ist die Seitennavigation jedoch statisch implementiert. Eine zukünftige Verbesserung bestünde darin, nur jene Seiten anzuzeigen, die tatsächlich Analyseergebnisse enthalten. Sind beispielsweise ausschliesslich Schlafdaten vorhanden, könnten die Tabs für Allgemein- und Aktivitätsdaten ausgeblendet werden. Dies würde die Benutzerführung deutlich verbessern und unnötige Verwirrung vermeiden.

8.12.4 Datensicherheit und temporäre Dateien

Ein sicherheitsrelevanter Punkt betrifft die Handhabung temporärer Dateien. Bei einem unerwarteten Absturz der Applikation werden Analyseergebnisse nicht zuverlässig aus dem Speicher entfernt. Künftig könnten temporäre Dateien verschlüsselt abgelegt werden, um die Datensicherheit zu erhöhen.

8.12.5 Intelligente Dateiauswahl

Aus Gründen der Erweiterbarkeit erfolgt aktuell keine Prüfung der Dateiendungen im File Picker der `ProcessDialogView`. Eine künftige Verbesserung könnte darin bestehen, die Auswahlmöglichkeit auf Dateiformate zu beschränken, die zum gewählten Sensor passen. Dadurch würden potenzielle Bedienfehler reduziert und die Benutzerfreundlichkeit gesteigert, da nur relevante Dateien zur Auswahl angeboten werden.

Kapitel 9

Qualitätsmanagement

9.1 Guidelines

9.1.1 Coding Guidelines

Allgemeine Konventionen

- **Sprache**: C# mit .NET 9.0 als Zielframework
- **Architektur**: Core, Infrastructure und Presentation Layer
- **UI Framework**: Avalonia UI für plattformübergreifende Desktop-Anwendungen
- **Interfaces**: Für alle Services und grössere Komponenten definieren

Namenskonventionen

- **Klassen und Interfaces**: PascalCase (z.B. `IAnalysis`, `SharedDataService`)
- **Methoden**: PascalCase (z.B. `UpdateSelectedAnalyses`, `GetChartData`)
- **Properties**: PascalCase (z.B. `FileName`, `IsProcessingInBackground`)
- **Private Felder**: Underscore-Präfix mit camelCase (z.B. `_sharedDataService`, `_isProcessing`)
- **Parameter und lokale Variablen**: camelCase (z.B. `analyses`, `newAnalyses`)
- **Konstanten**: PascalCase (z.B. `BinPattern`)

Code-Organisation

- **Namespaces**: Hierarchisch strukturiert (`ActiveSense.Desktop.Core.Domain.Models`)
- **Interfaces**: Beginnen mit "I" und definieren Verträge (`IAnalysis`, `ISharedDataService`)
- **Services**: Implementieren entsprechende Interfaces für Dependency Injection
- **Interface-First Approach**: Für testbare und lose gekoppelte Komponenten

MVVM-Pattern

- **Observable Properties**: Nutzen `[ObservableProperty]` Attribut aus CommunityToolkit.Mvvm

- **ViewModels**: Erben von `ViewModelBase` (welches `ObservableObject` erweitert)
- **Property-Definition**: Observable Properties am Anfang der Klasse definieren
- **Field-Naming**: Private Felder für Observable Properties mit Underscore-Präfix (z.B. `_selectedAnalyses`)

Dependency Injection

- Services werden über Constructor Injection bereitgestellt
- Service-Registrierung in `App.axaml.cs`
- Interfaces für alle injizierten Dependencies definieren

Testing

- **Test-Framework**: NUnit für Unit Tests
- **Mocking**: Moq für Mock-Objekte
- **Teststruktur**: Arrange-Act-Assert Pattern
- **Naming**: `[MethodName]_[Condition]_[ExpectedResult]`
- **Assertions**: NUnit Assert-Pattern verwenden (`Assert.That(result, Is.True)`)

Fehlerbehandlung

- **Try-Parse Pattern**: Für sichere Typ-Konvertierungen verwenden
- **Exception Handling**: Try-Catch für erwartete Fehlerszenarien
- **Logging**: Serilog für strukturiertes Logging verwenden

Dokumentation

- **Inline-Kommentare**: Nur für komplexe Logik verwenden
- **TODO-Kommentare**: Für ausstehende Implementierungen verwenden

9.1.2 Version Control

Branch-Strategie

- **main**: Produktive Version
- **feature/**: Feature-spezifische Branches
- **bugfix/**: Bugfix-Branches

Commit-Nachrichten

- Prägnante Beschreibung der Änderung
- Imperativ-Form verwenden (z.B. Add chart functionality)

Pull Requests

- Code Review vor Merge
- Alle manuellen und automatisierten Tests müssen erfolgreich durchlaufen

Git-Workflow

1. Feature-Branch von main
2. Entwicklung und Testing
3. Pull Request erstellen
4. Code Review
5. Merge nach main

9.2 Readme

Das README richtet sich an Benutzer, die ActiveSense installieren und verwenden möchten. Es enthält wichtige Systemanforderungen, Installationsanweisungen und grundlegende Nutzungshinweise. Das README muss bei jeder Änderung an den Systemanforderungen oder Installationsprozessen aktualisiert werden.

9.3 Prototyp

Im Rahmen der Bachelorarbeit wurde zu Beginn des Projekts ein Prototyp definiert, um die technische Machbarkeit der geplanten Anwendung zu validieren. Zu diesem Zeitpunkt war noch keine Implementierung vorhanden, weshalb der Prototyp als Proof-of-Concept zur Risikominimierung diente.

9.3.1 Prototyp-Definition und Zielsetzung

Der Prototyp hatte das primäre Ziel, die kritischen Herausforderungen des Projekts frühzeitig zu identifizieren und Lösungsansätze zu validieren. Insbesondere sollte geklärt werden, ob die geplante Integration zwischen .NET und R-Umgebung technisch umsetzbar ist und welche Architekturentscheidungen für das finale System getroffen werden müssen.

9.3.2 Schwerpunkte der Prototyp-Entwicklung

.NET-Infrastruktur mit Avalonia UI

Der erste Schwerpunkt lag darauf, sicherzustellen, dass das .NET-Projekt mit Avalonia UI funktioniert. Die Implementierung einer minimalen Anwendung diente als Nachweis, dass das gewählte Framework den Projektanforderungen entspricht.

Konzeption der .NET-zu-R-Schnittstelle

Ein zentraler Punkt war die Evaluation verschiedener Ansätze zur Integration von R-Funktionalität in die .NET-Anwendung. Dabei wurden zwei Optionen, R.NET und der direkte Prozessaufruf, evaluiert. Das Ziel war es, eine stabile Möglichkeit zu finden, um das R-Script auszuführen.

Prototypische Schnittstellen-Implementierung

Nach der konzeptionellen Phase wurde eine minimale Implementierung der R-Schnittstelle entwickelt, um den gewählten Ansatz zu validieren. Diese prototypische Umsetzung sollte demonstrieren, dass R-Scripts aus der .NET-Umgebung heraus ausgeführt werden können und Ergebnisse zuverlässig zurückgegeben werden.

UX-Konzept und Benutzerfuehrung

Der vierte Schwerpunkt umfasste die Entwicklung eines UX-Konzepts für das Endprodukt. Hierbei wurden Überlegungen zur optimalen Benutzerführung angestellt; von der Dateiauswahl über die Konfiguration der Analyseparameter bis hin zur Darstellung der Ergebnisse (siehe Abschnitt 6.6.2).

9.3.3 Validierung und Erkenntnisse

Die Prototyp-Entwicklung bestätigte die Machbarkeit des Gesamtkonzepts. Die .NET-Avalonia-Kombination erwies sich als gut geeignet. Für die R-Integration wurde der Ansatz über Prozessaufruf gewählt.

Die im Prototyp gewonnenen Erkenntnisse bildeten die Basis für alle weiteren Implementierungen und bestätigten, dass das gewählte technische Setup für die Umsetzung der Applikation geeignet ist.

9.4 Code Metrics

Die Code Metrics wurden mit NDepend am 10. Juni 2025 erfasst und dienen der objektiven Bewertung der Codequalität des ActiveSense Desktop-Projekts. Die Analyse umfasst 2 Assemblies mit insgesamt 42 Namespaces und bietet Einblicke in Komplexität, Grösse und Qualität des Codes.

9.4.1 Projektübersicht

Das ActiveSense-Projekt umfasst **6.277 Lines of Code** verteilt auf 117 Types (Klassen und Interfaces) mit insgesamt 923 Methods. Die Codebasis enthält 310 Fields und 174 Source Files.

9.4.2 Zyklomatische Komplexität

Die durchschnittliche zyklomatische Komplexität beträgt **1,54**. Die maximale Komplexität liegt bei **24**. **Bewertung der Komplexitätswerte:**

- Durchschnittliche Komplexität: 1,54 (Sehr gut - Zielbereich < 3)
- Maximale Komplexität: 24 (Akzeptabel - unter kritischem Grenzwert von 30)
- Verteilung: Die niedrige durchschnittliche Komplexität bei moderater maximaler Komplexität zeigt eine gesunde Code-Struktur

9.4.3 Technical Debt

Das Projekt weist eine Technical Debt von **3,75%** auf, was einem geschätzten Aufwand von 58 Stunden entspricht. Mit einem Rating von A befindet sich das Projekt in der besten Qualitätskategorie.

9.4.4 Code Coverage

Die Code Coverage wurde mit JetBrains dotCover analysiert und zeigt eine **Gesamtabdeckung von 81,93%**. Die Coverage-Verteilung zeigt unterschiedliche Abdeckungsgrade in den verschiedenen Projektbereichen.

9.4.5 Code Coverage

Die Code Coverage wurde mit JetBrains dotCover analysiert und zeigt eine **Gesamtabdeckung von 81%**. Die Coverage-Verteilung zeigt unterschiedliche Abdeckungsgrade in den verschiedenen Projektbereichen.

Coverage nach Assemblies

- **ActiveSense.Desktop**: 81% Coverage
- **Core Domain**: 80% Coverage
- **Infrastructure**: 83% Coverage
- **ViewModels**: 77% Coverage

Coverage nach Funktionsbereichen

- **Factories**: 100% Coverage

- **AnalysisPages**: 99% Coverage
- **Domain Models**: 96% Coverage
- **Export**: 92% Coverage
- **Charts**: 90% Coverage
- **Parse**: 88% Coverage
- **Converters**: 81% Coverage
- **Process**: 72% Coverage
- **Services**: 54% Coverage

Bewusst von der Testabdeckung ausgeschlossen wurden UI-nahe Komponenten wie Views, XAML-Dateien, der `ViewLocator` sowie Dialog-Implementierungen, da diese primär Präsentationslogik enthalten und automatisiert nur schwer testbar sind. Ebenfalls ausgenommen wurden Avalonia-spezifische, vom Compiler generierte Methoden sowie Initialisierungscode der App.

Ein detaillierter Bericht zur Testabdeckung befindet sich im Anhang 12.2.

9.5 Testkonzept

Das Testkonzept umfasst eine mehrstufige Teststrategie zur Sicherstellung der Softwarequalität auf verschiedenen Ebenen. Die Tests gliedern sich in automatisierte Unit Tests, Regressionstests für die R-Analyse, manuelle Acceptance Tests und Usability Tests, wobei jede Kategorie spezifische Aspekte der Anwendung validiert.

Automatisierung: Die Unit Tests sind vollständig automatisiert und werden bei jedem Push über einen Github-Workflow ausgeführt. Die Acceptance Tests werden manuell durchgeführt, wobei standardisierte Testdaten und -prozeduren verwendet werden. Usability Tests erfolgen ebenfalls manuell mit echten Benutzern.

9.5.1 Unit Tests

Die Unit Tests sind entsprechend der Struktur des Projekts organisiert und spiegeln die Projektarchitektur wider.

Test-Organisation nach Projektstruktur:

- **CoreTests**: Tests für Domain Models, Services und Business Logic
- **InfrastructureTests**: Tests für Parse-, Process- und Export-Funktionalitäten
- **ViewModelTests**: Tests für UI-Logic und MVVM-Pattern-Implementierung

GitHub Workflow

Bei jedem Push in das GitHub Repository sowie bei Pull Requests zum main-Branch wird automatisch eine CI/CD-Pipeline ausgeführt. Der Workflow "`.NET Build and Test`" läuft auf Ubuntu und stellt sicher, dass Codeänderungen die bestehende Funktionalität nicht beeinträchtigen und alle Tests erfolgreich durchlaufen.

9.5.2 Acceptance Tests

Die Acceptance Tests validieren die End-to-End-Funktionalität der Anwendung aus Benutzersicht und decken die Sensordatenverarbeitung, Ergebnisanalyse und den Datenexport ab.

Test-Kategorien

Verarbeitungstests:

- **AC-P-001**: Grundlegende Dateiverarbeitung mit small_dataset.bin
- **AC-P-002**: Verarbeitung mit benutzerdefinierten Parametern (selektive Analyse-Aktivierung)
- **AC-P-003**: Fehlerbehandlung für ungültige Dateien (corrupted.bin)
- **AC-P-004**: Verhalten bei fehlender R-Installation

Export-Tests:

- **AC-E-001**: PDF-Bericht Export mit eingebetteten Analysedaten
- **AC-E-002**: ZIP-Export mit Rohdaten (PDF + CSV-Dateien)

Import-Tests:

- **AC-PR-001**: PDF-Bericht Import mit Datenextraktion
- **AC-PR-002**: Gemischter Datei-Import (PDF und BIN-Dateien)

Test-Durchführung

Die Acceptance Tests werden manuell durchgeführt und verwenden standardisierte Testdaten, die in einem separaten Testdatenordner (siehe Anhang 12.3) organisiert sind:

Standardisierte Testdaten:

- `small_dataset.bin`: Gültige Sensordaten für Basis-Tests
- `normal_dataset.bin`: Standard-Datensatz für Volltest
- `corrupted.bin`: Defekte Datei für Fehlerbehandlungs-Tests

- `complete.pdf`: Vollständiger PDF-Bericht für Import-Tests
- `no_data.pdf`: PDF ohne eingebettete Daten

Jeder Test definiert klare Akzeptanzkriterien und erwartete Ergebnisse. Die Tests validieren sowohl positive Szenarien (erfolgreiche Verarbeitung) als auch negative Szenarien (Fehlerbehandlung).

Test-Dokumentation: Alle Acceptance Tests sind im separaten Dokument 12.3 detailliert dokumentiert, welches Testschritte, erwartete Ergebnisse und Akzeptanzkriterien für jeden Test spezifiziert.

9.5.3 Regressions Tests

Die Performance-Optimierungen an den Kernpaketen GENEAread und GENEAclassify erforderten eine zuverlässige Validierungsstrategie. Um sicherzustellen, dass die algorithmischen Änderungen keine Auswirkungen auf die Analyseergebnisse haben, wurde ein Regressionstestsystem unter Verwendung der `testthat`-Bibliothek von R implementiert.

> **Hinweis:**
> `testthat` ist das Standard-Framework für Unit-Tests in R und ermöglicht die automatisierte Validierung von Funktionalitäten durch strukturierte Test-Assertions.

9.5.4 Referenz-Dateien

Das Testsystem basiert auf fünf repräsentativen Datensätzen, die als Referenz-Dateien fungieren. Für diese Referenzdatensätze wurden zunächst mit den ursprünglichen, unmodifizierten Algorithmen Baseline-Ergebnisse generiert.

9.5.5 Automatisierte Validierung

Der Validierungsprozess führt die optimierten Algorithmen auf denselben Rohdateien aus und vergleicht die resultierenden CSV-Ausgabedateien direkt mit den Referenzergebnissen. Bei Abweichungen zwischen den Datensätzen schlägt der Test mittels der testthat-Bibliothek von R fehl und erfordert eine Überprüfung der Implementierung.

Diese direkte Vergleichsmethodik gewährleistet die vollständige funktionale Äquivalenz zwischen ursprünglicher und optimierter Implementierung.

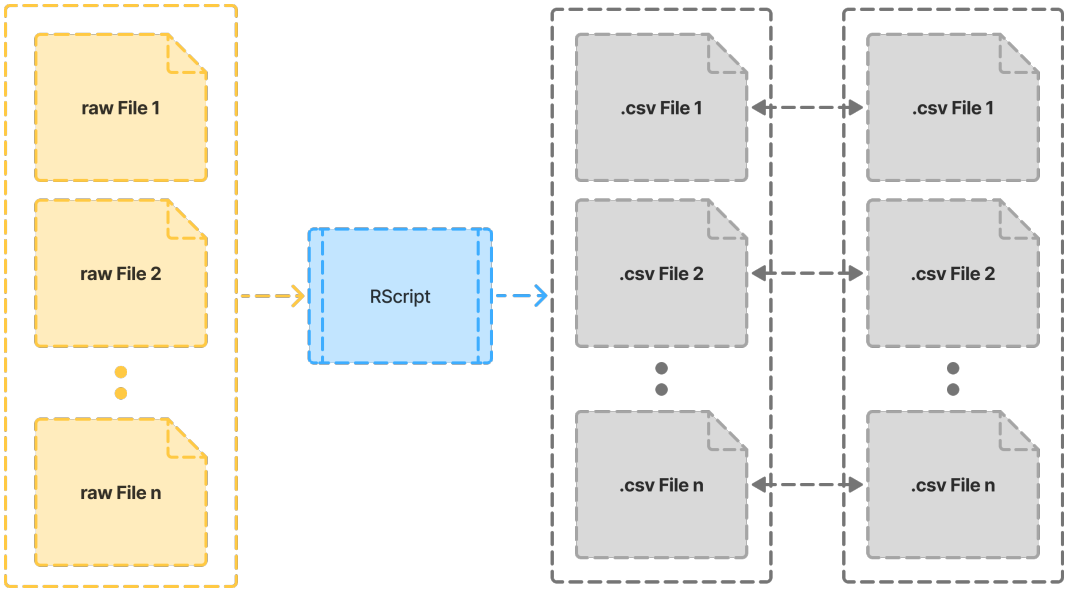


Abbildung 9.1: Workflow für die Datenverarbeitung und Berichtserstellung mit CRAN-Paketen

9.5.6 Usability Tests

Die Implementierung und Durchführung der Usability Tests wird in Abschnitt 6.8.1 detailliert beschrieben.

Kapitel 10

Ergebnisse / Resultate

10.1 Überblick der erreichten Ziele

Die vorliegende Bachelorarbeit hat eine Desktop-Anwendung zur Optimierung der GENEActiv-Sensordatenanalyse entwickelt. Die wesentlichen Projektziele konnten erreicht werden. Das Ergebnis ist eine funktionsfähige Anwendung, die den bestehenden RStudio-basierten Workflow durch eine Desktop-Lösung ersetzt.

10.2 Performance-Verbesserungen

10.2.1 Analysegeschwindigkeit

Die implementierten Performance-Optimierungen führten zu messbaren Verbesserungen. Die Aktivitätsanalyse wurde von 13 Minuten 5 Sekunden auf 2 Minuten 46 Sekunden reduziert, die Schlafanalyse von 15 Minuten 40 Sekunden auf 2 Minuten 50 Sekunden. Diese Verbesserungen entstanden hauptsächlich durch Vektorisierung der StepCounter-Funktion und Eliminierung redundanter Code-Abschnitte.

10.2.2 Landing Zones Evaluation

Die definierten Performance-Ziele wurden erreicht:

Analyse-Typ	Minimum	Ziel	Übertroffen	Erreicht
Schlafanalyse	10min	6min	4min	**2m 50s**
Aktivitätsanalyse	10min	6min	4min	**2m 46s**

Tabelle 10.1: Performance-Vergleich: Ziele vs. erreichte Werte

10.3 Benutzerfreundlichkeit

Der ursprünglich mehrstufige Prozess wurde auf einen 3-Schritt-Workflow reduziert: Datei-Import, Ergebnisbetrachtung und Export. Die durchgeführten Usability Tests mit MOVE-IT Mitarbeitern zeigten, dass alle Testpersonen die Grundfunktionen ohne Anleitung verwenden konnten.

Die Anwendung berücksichtigt Barrierefreiheit durch eine Okabe-Ito-Farbpalette für farbenblinde Nutzer und ausreichende Farbkontraste gemäss WCAG 2.1.

10.4 Technische Implementierung

Die entwickelte Softwarearchitektur erreichte eine Testabdeckung von 81% und eine Technical Debt von 3,75%. Die Anwendung wurde für Windows 11, macOS und Linux implementiert und folgt den Prinzipien der mehrschichtigen Architektur mit Interface-basierter Programmierung und Dependency Injection.

10.5 Anforderungserfüllung

Von 21 definierten funktionalen Anforderungen wurden 20 vollständig erfüllt (siehe Abschnitt 4.1). Alle 17 nicht-funktionalen Anforderungen konnten implementiert werden (siehe Abschnitt 4.2).

10.6 Limitationen

Die Anwendung benötigt weiterhin eine lokale R-Installation und deren Paketverwaltung. Der Export ist auf Einzeldateien beschränkt, was Batch-Verarbeitung limitiert. Zukünftige Verbesserungen könnten eine serverbasierte Architektur und Dateivalidierung umfassen.

10.7 Fazit

Die ActiveSense-Anwendung erfüllt die gesetzten Projektziele mit erheblichen Geschwindigkeitsverbesserungen und einer Vereinfachung des Analyseworkflows. Die entwickelte Lösung wandelt einen komplexen R-basierten Prozess in eine benutzerfreundlichere Desktop-Anwendung um und bietet eine Grundlage für zukünftige Erweiterungen.

Kapitel 11

Reflexion der Methodenwahl

11.1 Reflexion des technologischen Stacks

Der gewählte technologische Stack aus .NET 9 und Avalonia UI erwies sich als sehr zufriedenstellende Entscheidung für unser Projekt. Die Kombination ermöglichte es uns, unsere Vision einer plattformübergreifenden Desktop-Anwendung präzise umzusetzen und dabei moderne Entwicklungsprinzipien zu befolgen.

Stärken der gewählten Architektur:

- Avalonia UI bot die notwendige Flexibilität für eine intuitive Benutzeroberfläche
- Das MVVM-Pattern und Dependency Injection ermöglichten eine saubere, testbare Architektur

Kritische Reflexion der R-Integration: Eine der grössten architektonischen Herausforderungen war die Integration des bestehenden R-Codes. Retrospektiv betrachtet wäre eine serverbasierte Lösung mit REST-API deutlich vorteilhafter gewesen. Die lokale R-Installation brachte mehrere Nachteile mit sich:

- Komplexe Deployment-Strategien aufgrund der R-Abhängigkeit
- Schwierige Paketverwaltung mit Custom-Repositories
- Eingeschränkte Parallelisierungsmöglichkeiten durch Speicherlimitierungen

Die Entscheidung, weiterhin auf R zu setzen, lag in der Komplexität der bestehenden GENEActiv-Bibliotheken begründet. Diese waren zu stark miteinander verzahnt, sodass eine vollständige Neuimplementierung den Projektrahmen gesprengt hätte. Eine serverbasierte Architektur hätte jedoch folgende Vorteile geboten:

- Vereinfachtes Client-Deployment ohne R-Installation
- Optimierte Hardware-Ressourcen für rechenintensive Analysen
- Zentrale Wartung und Updates der Analysealgorithmen
- Bessere Skalierbarkeit und Parallelisierung

11.2 Reflexion über den Einsatz von KI-Werkzeugen

Der Einsatz von KI-Tools brachte sowohl erhebliche Vorteile als auch potenzielle Risiken mit sich.

Positive Aspekte:

- **Dokumentationserstellung:** KI-Assistenten halfen dabei, technische Konzepte klar und verständlich zu formulieren
- **Code-Optimierung:** Unterstützung bei der Identifikation von Performance-Bottlenecks und Refactoring-Möglichkeiten
- **Problemlösung:** Schnelle Recherche zu spezifischen technischen Fragestellungen
- **Code-Review:** Zusätzliche Perspektive bei der Qualitätskontrolle

Kritische Einschränkungen: Bei architektonischen Entscheidungen erforderte der KI-Einsatz besondere Vorsicht. KI-Systeme neigen dazu, scheinbar plausible, aber strukturell problematische Lösungsansätze vorzuschlagen. Dies zeigte sich besonders bei:

- Komplexen Architekturentscheidungen, wo oberflächlich korrekt erscheinende Lösungen langfristige Wartbarkeitsprobleme verursachen können
- Performance-kritischen Implementierungen, wo theoretische Optimierungen praktisch ungeeignet waren

Fazit zum KI-Einsatz:
KI-Werkzeuge erwiesen sich als wertvolle Unterstützung, niemals aber als Ersatz für kritisches Denken und fachliche Expertise. Der grösste Nutzen entstand bei klar definierten, abgegrenzten Aufgaben, während bei strategischen Entscheidungen weiterhin menschliche Expertise erforderlich war. Für zukünftige Projekte empfiehlt sich ein bewusster, reflektierter Einsatz dieser Technologien als Ergänzung, nicht als Grundlage der Entwicklungsmethodik.

11.2.1 Hilfsmittelverzeichnis

Im Rahmen dieser Bachelorarbeit wurden folgende KI-Assistenten und Tools eingesetzt:

Aufgabenbereich	Tools
Recherche und Problemlösung	Stack Overflow, YouTube, .NET Documentation, Avalonia Docs, Claude, Gemini, R docs
Texterstellung, Textoptimierung, Rechtschreibe- und Grammatikprüfung	DeepL Write, Claude, ChatGPT
Design und Prototyping	Figma

Aufgabenbereich	Tools
Coding	Avalonia Docs, Claude, .NET documentation, Gemini, Rider
DevOps	Github
Zusammenarbeit und Projektmanagement	Jira, Teams, Outlook, Word, Figma
Dokumentation	Latex mit Overleaf

Kapitel 12

Schlussfolgerungen

Diese Bachelorarbeit hat gezeigt, dass die Effizienz der GENEActiv-Sensordatenanalyse durch die Entwicklung einer benutzerfreundlichen Desktop-Lösung erheblich gesteigert werden kann. Die entwickelte Desktop-Anwendung reduziert die Analysezeit deutlich.

Die grössten Herausforderungen lagen in der Performance-Optimierung des bestehenden R-Codes und der Integration verschiedener Technologien. Durch Profiling konnten kritische Bottlenecks identifiziert und durch Vektorisierung sowie Code-Bereinigung behoben werden.

Die Architekturentscheidung für eine lokale Desktop-Anwendung erwies sich als angemessen für den Anwendungskontext. Das modulare Design ermöglicht zukünftige Erweiterungen um weitere Sensortypen.

Literaturverzeichnis

[Act24] ActiGraph LLC. Actilife software, 2024. Accessed: 2025-01-15.

[Bir12] Jennifer Birch. Worldwide prevalence of red-green color deficiency. *Journal of the Optical Society of America A*, 29(3):313–320, 2012. Prevalence rates: European Caucasians 8% males, 0.4% females.

[vHSA+19] Vincent T van Hees, Séverine Sabia, Kirstie N Anderson, Sarah J Denton, James Oliver, Michael Catt, Jessica G Abell, Mika Kivimäki, Michael I Trenell, and Archana Singh-Manoux. Ggir: A research community–driven open source r package for generating physical activity and sleep outcomes from multi-day raw accelerometer data. *Journal for the Measurement of Physical Behaviour*, 2(3):188–196, 2019.

[W3C18] W3C. Web content accessibility guidelines (wcag) 2.1. W3c recommendation, World Wide Web Consortium, 2018. Level AA conformance.

[Won11] Bang Wong. Points of view: Color blindness. *Nature Methods*, 8(6):441, 2011.

Anhang

12.1 NDepend Analyse

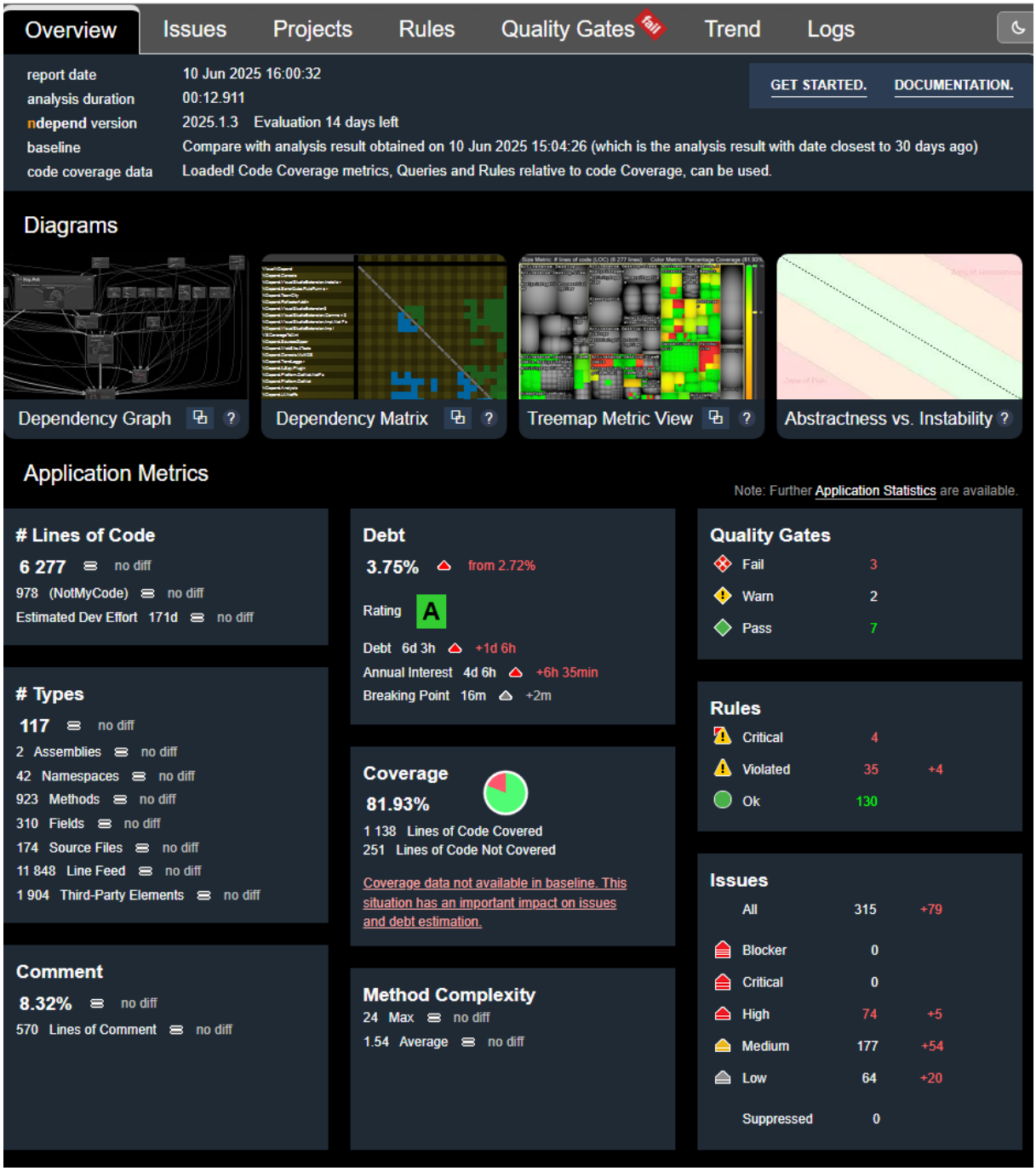


Abbildung 12.1: Screenshot der NDepend Analyse

12.2 dotcover Coverage Report

Siehe Datei `coverage.html`.

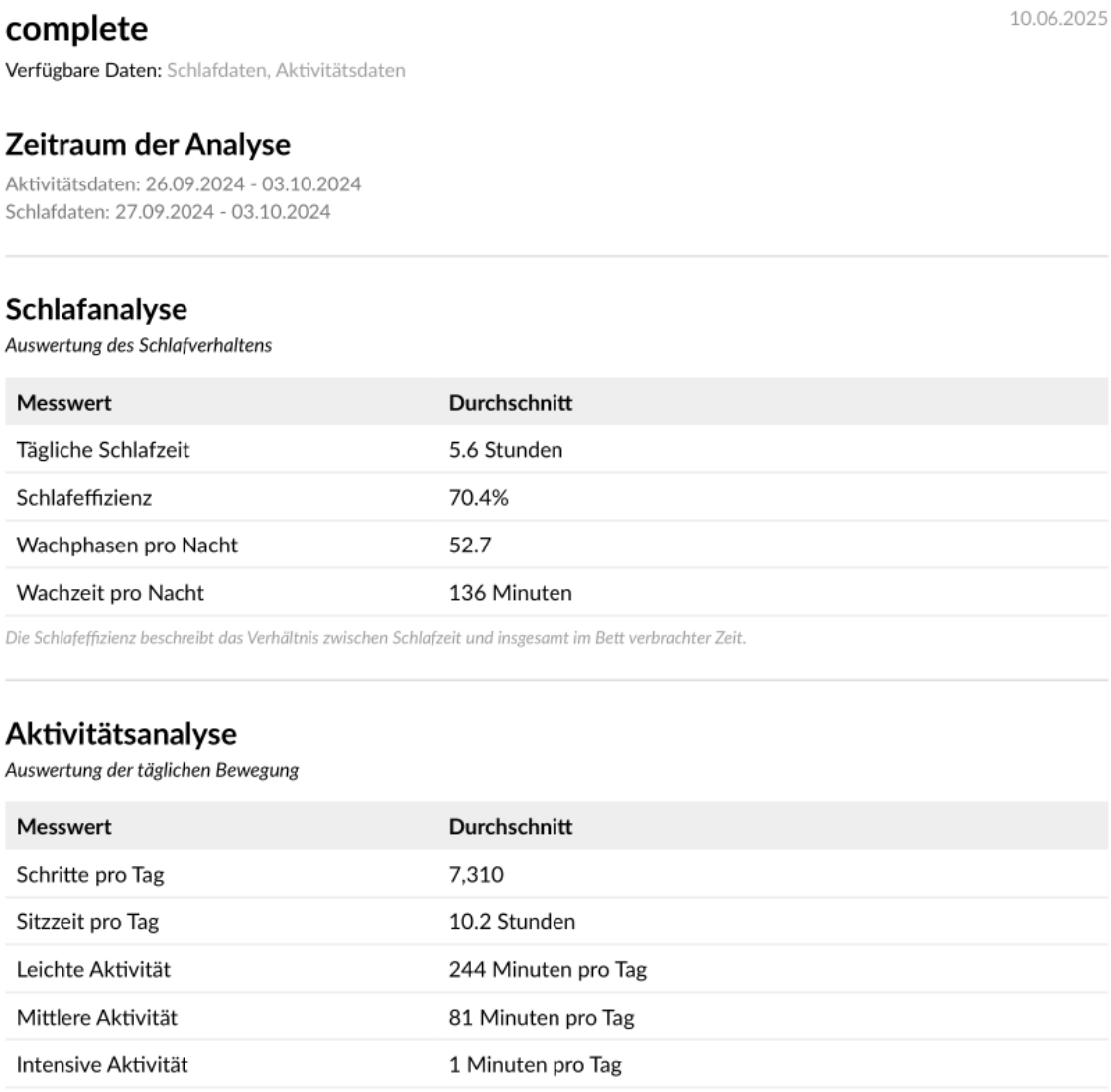
12.3 Acceptance Tests

Siehe Ordner `ActiveSense_AcceptanceTests.zip`.

12.4 Usability-Tests

Siehe das Dokument im Ordner Testkonzept_ActiveSense.docx

12.5 Analysis PDF-Report



complete

10.06.2025

Verfügbare Daten: Schlafdaten, Aktivitätsdaten

Zeitraum der Analyse

Aktivitätsdaten: 26.09.2024 - 03.10.2024
Schlafdaten: 27.09.2024 - 03.10.2024

Schlafanalyse

Auswertung des Schlafverhaltens

Messwert	Durchschnitt
Tägliche Schlafzeit	5.6 Stunden
Schlafeffizienz	70.4%
Wachphasen pro Nacht	52.7
Wachzeit pro Nacht	136 Minuten

Die Schlafeffizienz beschreibt das Verhältnis zwischen Schlafzeit und insgesamt im Bett verbrachter Zeit.

Aktivitätsanalyse

Auswertung der täglichen Bewegung

Messwert	Durchschnitt
Schritte pro Tag	7,310
Sitzzeit pro Tag	10.2 Stunden
Leichte Aktivität	244 Minuten pro Tag
Mittlere Aktivität	81 Minuten pro Tag
Intensive Aktivität	1 Minuten pro Tag

Abbildung 12.2: PDF-Report mit künstlichen Daten aus einem Export.

12.6 Verwendete 3rd Party Libraries

Library Name	Beschreibung	Lizenz
.NET/C# Libraries (NuGet Packages)		
Avalonia	Cross-platform UI Framework	MIT
Avalonia.Desktop	Desktop-spezifische Avalonia Features	MIT
Avalonia.Fonts.Inter	Inter Font für Avalonia UI	MIT
Avalonia.Diagnostics	Debug-Tools für Avalonia (nur Debug)	MIT
CommunityToolkit.Mvvm	MVVM Pattern Implementierung	MIT
CsvHelper	CSV Datei Lese-/Schreibbibliothek	Apache 2.0
Hardware.Info	System Hardware Informationen	MIT
itext7	PDF Erstellung und Manipulation	AGPL 3.0
LiveChartsCore.SkiaSharpView.Avalonia	Charting Library für Avalonia	MIT
Microsoft.Extensions.DependencyInjection	Dependency Injection Container	MIT
Newtonsoft.Json	JSON Serialisierung/Deserialisierung	MIT
NUnit3TestAdapter	Test Adapter für NUnit	MIT
QuestPDF	PDF Dokument Generierung	MIT
ScottPlot.Avalonia	Plotting Library für Avalonia	MIT
Semi.Avalonia	UI Theme für Avalonia	MIT
Serilog	Structured Logging Framework	Apache 2.0
Serilog.Extensions.Logging	Serilog Integration für MS Logging	Apache 2.0
Serilog.Sinks.Console	Console Output für Serilog	Apache 2.0
Test Libraries		
JetBrains.Annotations	Code Annotations für bessere Analyse	MIT
Microsoft.NET.Test.Sdk	.NET Test SDK	MIT
Moq	Mocking Framework für Unit Tests	BSD 3-Clause
MSTest.TestAdapter	Test Adapter für MSTest	MIT
MSTest.TestFramework	MSTest Framework	MIT
NUnit	Unit Testing Framework	MIT
R Libraries		
optparse	Kommandozeilen-Parsing für R	GPL-2
GENEAread	GENEA Sensor Daten lesen	GPL-2
GENEAclassify	GENEA Aktivitätsklassifikation	GPL-2

12.7 Messung der Startzeit von ActiveSense

Tabelle 12.1: Startzeit-Messungen der ActiveSense Desktop Applikation

System	Betriebssystem	Test 1	Test 2	Test 3	Test 4	Durchschnitt
ThinkPad T14s	Windows 11 (nativ)	1.6s	1.7s	1.5s	1.8s	1.65s
MacBook Pro	macOS	1.8s	1.9s	1.7s	1.8s	1.80s
ThinkPad T14s	Windows 11 (VM)	1.9s	2.0s	1.8s	1.9s	1.90s
ThinkPad T14s	Ubuntu 22.04 (VM)	1.7s	1.8s	1.6s	1.9s	1.75s

Testbedingungen:

- Messung vom Doppelklick auf die Anwendung bis zur vollständigen Benutzeroberfläche
- Keine anderen ressourcenintensiven Anwendungen während der Messung
- Jeder Test wurde 4 Mal wiederholt
- VM-Konfiguration: 8GB RAM, 4 CPU-Kerne zugewiesen

12.8 Visualisierungen

Siehe `Übersicht_Visualisierungen.pdf`.