Hochschule Fulda
University of Applied Sciences

Entwicklung und Implementierung eines Modells zur Erfassung und Bewertung der User Experience am Beispiel der Anwendung eDok des LWV Hessen

Ali Alkhiami

Bachelorarbeit am Fachbereich AI der HAW Fulda

Matrikelnummer: 844620
Erstbegutachtung: Prof. Dr. Alexander Gepperth
Zweitbegutachtung: Prof. Dr. Dr. YYY

Eingereicht am 05.mm.yyyy

0.1 Abstract

Diese Arbeit befasst sich mit der Entwicklung und Implementierung eines Modells zur umfassenden Erfassung und Bewertung der User Experience (UX), exemplarisch angewendet auf die Anwendung eDok des Landeswohlfahrtsverbands Hessen (LWV Hessen). Ziel des Modells ist es, einen detaillierten Überblick über die UX innerhalb der Anwendung zu schaffen, indem während der produktiven Nutzung anonymisierte Daten gesammelt werden.

Die gesammelten Daten sind bewusst nicht benutzerorientiert, um Datenschutzrichtlinien einzuhalten und die Privatsphäre der Nutzer zu schützen. Das Modell präsentiert diese aggregierten Daten in einem Dashboard, das ausschließlich für Nutzer mit Administratorrechten zugänglich ist. Dieses Dashboard bietet anpassbare Grafiken und Visualisierungen, die durch Filter und weitere Kriterien modifiziert werden können. Dadurch erhalten Administratoren nicht nur Einblicke in die aktuelle UX, sondern können auch Trends und Entwicklungen über einen bestimmten Zeitraum analysieren.

Ein wesentlicher Aspekt des Modells ist die Möglichkeit für Projektverantwortliche, die Datenerfassung auf bestimmten Seiten der Anwendung gezielt an- oder auszuschalten. Dies stellt sicher, dass nur relevante Bereiche beobachtet werden, ohne die Performance auf anderen Seiten zu beeinträchtigen. Das Modell ermöglicht es, die Verweildauer der Nutzer auf spezifischen Seiten zu überwachen, häufig durchlaufene Navigationsmuster zu identifizieren und die Häufigkeit bestimmter Fehler zu erfassen, die von den Nutzern gemacht werden.

Die gewonnenen Erkenntnisse unterstützen die Projektverantwortlichen dabei, problematische Stellen innerhalb der Anwendung zu identifizieren und gezielte Maßnahmen zur Verbesserung der Usability zu ergreifen. Darüber hinaus ermöglicht das Modell eine nachträgliche Überprüfung der implementierten Änderungen, um deren Effektivität zu bewerten und weitere Optimierungen vorzunehmen. Durch diesen iterativen Prozess trägt das Modell wesentlich zur Steigerung der Benutzerzufriedenheit und Effizienz der Anwendung bei.

Inhaltsverzeichnis

0.1 Abstract 1

Inhaltsverzeichnis **1**
Abkürzungsverzeichnis 3

1 Einleitung **4**
1.1 Kontext 4
1.2 Motivation 5
1.3 Ziel der Arbeit 6
1.4 Fragestellung 7
1.5 Eigenleistung 7

2 Grundlagen und Methodik **7**
2.1 Architektur und Softwarekomponenten . . . 7
2.2 Datenbankmodell . . . 8
2.3 Methodisches Vorgehen bei der Datenerfassung . . . 9
2.4 SQL-Datenbankmodell . . . 9
2.5 Zusammenfassung . . . 11
2.6 Relevanz der Arbeit . . . 11
2.7 Datenschutz und DSGVO . . . 12
2.8 Performance-Überlegungen . . . 12

3 Einführung in das Thema UX **13**
3.1 Herausforderungen und Offenheit der UX-Definition . . . 13
3.2 Forschungsperspektiven: UX als vielschichtiges Phänomen . . . 13
3.3 Zusammenfassung der UX-Definitionen . . . 14
3.4 Usability und ihre Bedeutung . . . 14
3.5 Usability in eDok . . . 15
3.6 Erweiterte Ziele der UX- und Usability-Bewertung in eDok . . . 15
3.6.1 Abgrenzung von UX- und Usability-Evaluationsmethoden . . . 16
3.6.2 Fokussierung auf Usability in dieser Arbeit . . . 16
3.6.3 Vergleich von Nutzerinterviews, Fragebögen und der Sammlung quantitativer Daten . . . 19
3.6.4 Nutzerinterviews und Fragebögen . . . 19
3.7 Sammlung und Analyse quantitativer Daten . . . 19
3.8 Erkenntnisse und Schlussfolgerung . . . 20
3.9 Methodik . . . 20
3.9.1 Systemarchitektur und Datenerfassung . . . 21
3.9.2 Datenpersistenz und Datenbankmodell . . . 21
3.9.3 Backend-Logik und REST-Schnittstellen . . . 22
3.9.4 Anonymisierung und Datenschutz . . . 22
3.9.5 Visualisierung und Heatmap-Darstellung . . . 22
3.9.6 Iterative Verbesserung und Anpassung . . . 22

4 Kapitel 3 **22**
4.1 Anforderungen . . . 22
4.2 Spezifikation der Erfassten UX-Daten in eDok . . . 23
4.3 Implementierung der Heatmap . . . 23
4.4 Zweck und Ziele der Heatmap . . . 23
4.5 Datenerfassung und Speicherung . . . 24
4.6 Implementierungsstrategie . . . 24
4.6.1 Global Heatmap Service . . . 24
4.6.2 Overlay Directive . . . 24
4.6.3 Admin Toggle Control . . . 24
4.6.4 Beispiele für Auswertungen . . . 25

4.7 Umsetzung – Backend . . . 25
4.8 Datenbankentitäten und Objekt-Relationales Mapping . . . 25
4.8.1 Vorteile des objekt-relationalen Mappings (ORM) . . . 25
4.8.2 Beispiel: ClickEvent . . . 26
4.8.3 Weitere Entity-Klassen . . . 27
4.8.4 Gründe für die gewählte Struktur . . . 28
4.9 Fazit . . . 29
4.10 DAO-Schicht . . . 29
4.10.1 ClickEventDAO . . . 29
4.10.2 Weitere DAOs . . . 30
4.11 Controller-Schicht . . . 31
4.11.1 AdminDashboardController . . . 31
4.11.2 ClickEventController . . . 32
4.11.3 ClickSequenceController . . . 33
4.11.4 ErrorEventController . . . 33
4.11.5 Weitere Controller . . . 33
4.12 Zusammenfassung der Backend-Architektur . . . 34
4.13 Frontend: Pfade und Endpunkte (Kurzüberblick) . . . 36
4.14 Beispielhafter Angular-Service zur Heatmap-Visualisierung (Kurzfassung) 37
4.15 Kurz gefasster Code-Ausschnitt (HeatmapCaptureService) . . . 39
4.16 Beispielhafter RequestLogService (Kurzfassung) . . . 41
4.17 HTTP-Interceptor zum Timing und Loggen von Requests . . . 43
4.17.1 Registrierung des Interceptors . . . 43
4.17.2 Funktionsweise (Kurzfassung) . . . 43
4.17.3 Beispielcode (Auszüge) . . . 43
4.17.4 Leistungsaspekte und Erweiterungen . . . 45
4.18 Globaler Error-Handler und manuelles Fehlermanagement . . . 46
4.18.1 GlobalErrorHandler (Auszug) . . . 46
4.18.2 Manuelles Fehlermanagement in der `ErrorTestComponent` . . . 47
4.18.3 ErrorEventService (Auszug) . . . 48
4.19 Bezug zu den Zielen (Kapitel 1) . . . 48

5 Diskussion **50**
5.1 Stärken der Umsetzung . . . 50
5.2 Einschränkungen und mögliche Verbesserungen . . . 50
5.3 Ausblick und Weiterentwicklung . . . 51
5.4 Fazit zur Diskussion . . . 51

UX User Experience

LWV Landeswohlfahrtsverband

SUS System Usability Scale

UEQ	User Experience Questionnaire
UI	User Interface
eDok	elektronische Dokumentenerstellung
HCI	Humin Computer Interaction

1 Einleitung

Die **Mensch-Computer-Interaktion** (Human-Computer Interaction, HCI) ist ein interdisziplinäres Forschungsfeld, das sich mit der Gestaltung, Evaluation und Implementierung interaktiver Computersysteme für die menschliche Nutzung sowie mit der Untersuchung der damit verbundenen Phänomene beschäftigt. Innerhalb dieses Feldes haben sich zwei zentrale Ansätze zur **User Experience** (UX) herausgebildet: das **UX-Design** und das **nutzerzentrierte Design** (User-Centered Design) [**glanznig**].

Das **UX-Design** strebt danach, ein ganzheitliches Nutzererlebnis zu schaffen, das über die reine Funktionalität eines Produkts hinausgeht. Es berücksichtigt emotionale, psychologische und ästhetische Aspekte, um Produkte zu entwickeln, die nicht nur effizient, sondern auch angenehm und zufriedenstellend in der Nutzung sind. Ziel ist es, ein positives Gesamterlebnis zu gestalten, das die Bedürfnisse und Erwartungen der Nutzer übertrifft.

Im Gegensatz dazu fokussiert das **nutzerzentrierte Design** auf die Einbindung der Endnutzer während des gesamten Entwicklungsprozesses. Durch aktive Einbindung von Nutzerfeedback wird sichergestellt, dass das Endprodukt intuitiv bedienbar ist und den Anforderungen der Zielgruppe entspricht. Dieser Ansatz betont die Bedeutung, die tatsächlichen Bedürfnisse, Ziele und Fähigkeiten der Nutzer zu verstehen und in die Gestaltung einzubeziehen.

Obwohl beide Ansätze entscheidend für die Entwicklung qualitativ hochwertiger Produkte sind, liegt der Schwerpunkt dieser Arbeit nicht auf der Gestaltung selbst, sondern darauf, **wie UX erfasst und gemessen werden kann**. In einer zunehmend digitalisierten Welt, in der Anwendungen immer komplexer werden und die Erwartungen der Nutzer stetig steigen, ist die systematische Messung der UX von entscheidender Bedeutung. Nur durch ein tiefgehendes Verständnis des Nutzerverhaltens und der Nutzerzufriedenheit können Produkte entwickelt werden, die den hohen Ansprüchen gerecht werden.

1.1 Kontext

Der **Landeswohlfahrtsverband Hessen** (LWV Hessen) ist eine zentrale Organisation, die als überörtlicher Träger der Eingliederungshilfe tätig ist. Er fördert die soziale Inte-

gration und Unterstützung von Menschen mit Behinderungen in Bereichen wie Wohnen, Bildung, Mobilität und Arbeit. Mit der Umsetzung des Bundesteilhabegesetzes sowie Programmen wie dem *Persönlichen Budget* und dem *Budget für Arbeit* ermöglicht der Verband eine selbstbestimmte Lebensführung und gesellschaftliche Teilhabe. Zusätzlich bietet der LWV Hessen spezifische Förderangebote für Kinder und Jugendliche mit geistigen, emotionalen oder körperlichen Einschränkungen, darunter Frühberatungsstellen und Förderschulen. Auch Menschen, die Leistungen des Sozialen Entschädigungsrechts benötigen, finden beim LWV Hessen Unterstützung.

Die **ANLEI-Service GmbH**, eine Tochtergesellschaft des LWV Hessen, ergänzt dessen Tätigkeiten durch IT-Dienstleistungen für die Sozialverwaltung. Sie entwickelt Systeme zur Unterstützung von Antragsbearbeitung, Leistungsgewährung und Abrechnung von Sozialleistungen, insbesondere in den Bereichen Eingliederungshilfe, Sozialhilfe und Kriegsopferfürsorge. Zu den zentralen Anwendungen gehören das *Integrierte Berichtssystem* (IBS) für betriebswirtschaftliche Analysen und *MASS*, das die maschinelle Abrechnung mit Einrichtungen und Trägern erleichtert.

Ein weiterer wichtiger Bestandteil ist *eDok*, ein moderner Service für die Erstellung barrierefreier Dokumente, der das bisherige *Schriftstück-Erstellungssystem* (SE) ablöst, das auf Microsoft Word und Makros basierte. *eDok* integriert *Doxee/Infinica* als Subsystem für die Dokumentbearbeitung und Output-Generierung, was sowohl eine interaktive Bearbeitung der Dokumente als auch eine automatisierte Verarbeitung im Hintergrund ermöglicht. Ziel ist es, eine universelle und generische Schnittstelle für die Anwendungen des LWV Hessen bereitzustellen, die eine einfache, effiziente und performante Erstellung barrierefreier Dokumente sicherstellt und die Daten aus den Fachverfahren nahtlos einbezieht.

Diese Bachelorarbeit wird im Rahmen einer praktischen Zusammenarbeit mit dem **Landeswohlfahrtsverband Hessen** (LWV Hessen) erstellt. Ziel ist es, ein innovatives System zu entwickeln, das Nutzungsdaten anonymisiert erfasst. Diese Arbeit verbindet praxisorientierte Softwareentwicklung mit wissenschaftlicher Forschung und trägt dazu bei, die Usability und Effizienz der Anwendung *eDok* zu optimieren.

Durch die direkte Einbindung in die Arbeitsprozesse des LWV Hessen wird nicht nur die Relevanz des Themas unterstrichen, sondern es entsteht auch eine wertvolle Gelegenheit, die entwickelten Konzepte und Technologien in einer realen Umgebung zu testen und anzuwenden. Somit leistet diese Arbeit einen Beitrag zur Verbesserung der digitalen Arbeitsabläufe des LWV Hessen und zur praxisnahen Weiterentwicklung moderner UX-Methoden.

1.2 Motivation

Die rasante technologische Entwicklung und die immer stärkere Integration des Internets in den Alltag haben die Art und Weise, wie Menschen mit digitalen Systemen interagieren, grundlegend verändert. Laut der ARD/ZDF-Onlinestudie **[ard]** nutzen mittlerweile rund 90 Prozent der deutschsprachigen Bevölkerung ab 14 Jahren regelmäßig das

Internet, wobei insbesondere die mediale Internetnutzung in den letzten Jahren signifikant zugenommen hat. Diese Entwicklung stellt neue Anforderungen an die Gestaltung von Nutzererlebnissen und unterstreicht die Bedeutung einer herausragenden UX.

In diesem Kontext wird es immer wichtiger, digitale Inhalte und Dienste so zu gestalten, dass sie den sich wandelnden Ansprüchen der Nutzer gerecht werden. Anwendungen müssen nicht nur funktional sein, sondern auch intuitiv bedienbar und ästhetisch ansprechend gestaltet sein, um im Wettbewerb bestehen zu können. Dies gilt insbesondere für komplexe Webanwendungen wie *eDok*, die von Institutionen wie dem Landeswohlfahrtsverband Hessen (LWV Hessen) genutzt werden.

Eine besondere Herausforderung besteht darin, innerhalb solcher Anwendungen Bereiche zu identifizieren, die ein verbessertes Design oder zusätzliche Informationen erfordern. Nutzer können unsicher sein, wie sie ihre Ziele innerhalb der Anwendung erreichen können, oder sich von der Informationsmenge überfordert fühlen. Hier setzt das entwickelte Modell an, das die UX automatisiert erfasst und eine fundierte Analyse ermöglicht. So können gezielt Schlüsselaspekte identifiziert werden, die optimiert werden sollten, um die Nutzerzufriedenheit zu steigern.

Zudem führen die regelmäßige Einführung neuer Funktionen und Veränderungen in der Benutzeroberfläche zu einer dynamischen Komplexitätsentwicklung in verschiedenen Bereichen der Anwendung. Ein Überblick über die Entwicklung der UX im Laufe der Zeit ermöglicht eine proaktive Anpassung und Optimierung, sodass die Anwendung trotz zunehmender Komplexität benutzerfreundlich bleibt.

Die wissenschaftliche Forschung betont die Wichtigkeit von Methoden zur UX-Messung und -Bewertung, um sicherzustellen, dass die Produktentwicklung in die richtige Richtung geht [**Virpi**]. Die Fähigkeit, UX quantitativ und qualitativ zu erfassen, bietet Unternehmen und Entwicklern einen unschätzbaren Vorteil. Durch das Verständnis des Nutzerverhaltens und der Nutzerzufriedenheit können gezielte Verbesserungen vorgenommen werden, die nicht nur die Effizienz steigern, sondern auch die Bindung der Nutzer an das Produkt fördern.

1.3 Ziel der Arbeit

Das Hauptziel dieser Arbeit ist die Entwicklung und Implementierung eines Modells, das eine umfassende Erfassung und Bewertung der UX ermöglicht. Dieses Modell soll spezifische, anonymisierte Daten sammeln, um die UX präzise und systematisch zu erfassen. Durch die Analyse dieser Daten über Zeiträume hinweg sollen dynamische Veränderungen erkannt und gezielte Optimierungsmöglichkeiten der Anwendung erschlossen werden.

Exemplarisch wird dieses Modell auf die Anwendung *eDok* des LWV Hessen angewendet. Dabei soll ein System geschaffen werden, das während der produktiven Nutzung Daten sammelt, ohne die Privatsphäre der Nutzer zu beeinträchtigen. Diese Daten werden in einem Dashboard visualisiert, das ausschließlich für Administratoren zu-

gänglich ist, um fundierte Entscheidungen zur Optimierung der Anwendung treffen zu können.

Ein weiterer Aspekt ist die Möglichkeit für Entwickler und Produktverantwortliche, die Datenerfassung auf spezifische Seiten der Anwendung zu beschränken oder zu erweitern. Dadurch kann sichergestellt werden, dass nur relevante Bereiche beobachtet werden, ohne die Performance auf anderen Seiten zu beeinträchtigen. Zudem ermöglicht das Modell, die Effektivität von Anpassungen zu überprüfen, indem Veränderungen im Nutzerverhalten nach Implementierung von Updates sichtbar gemacht werden.

1.4 Fragestellung

Welche spezifischen, anonymisierten Daten sind erforderlich, um ein Modell zu entwickeln und zu implementieren, das die User Experience präzise und systematisch erfasst, dynamische Veränderungen über Zeiträume hinweg analysiert, Verantwortliche bei fundierten Entscheidungen zur Optimierung der Anwendung unterstützt und dabei auftretende Herausforderungen bewältigt?

1.5 Eigenleistung

Im Rahmen der vorliegenden Arbeit wurde auf keinerlei nicht-öffentliche Vorarbeiten zurückgegriffen. Sämtliche entwickelte Softwarekomponenten (Angular-Frontend, Java-Spring-Backend sowie die SQL-Datenbankstrukturen) wurden von mir selbst erstellt oder basieren ausschließlich auf frei verfügbaren bzw. quelloffenen Bibliotheken, die unter entsprechend erlaubten Lizenzen stehen (Angular CLI, Spring Boot, SQLite).

2 Grundlagen und Methodik

In diesem Kapitel werden die methodischen und technischen Grundlagen vorgestellt, welche die Basis für die spätere Umsetzung im Rahmen dieser Arbeit bilden. Dabei wird auf bereits existierende Konzepte und Bibliotheken verwiesen, die im Projekt verwendet wurden. Zudem wird erläutert, wie die Datenbankstruktur und die zugrunde liegende Architektur gewählt wurden, um die Zielsetzungen der Arbeit (Erfassung, Analyse und Visualisierung von Nutzerinteraktionen) zu erfüllen. Auf detaillierte Implementierungsschritte wird hier verzichtet; diese finden sich in den folgenden Kapiteln und im Anhang.

2.1 Architektur und Softwarekomponenten

Die gewählte Systemarchitektur besteht aus einem Angular-Frontend und einem Spring-Boot-Backend, das auf einer relationalen Datenbank (z. B. SQLite) aufsetzt:

- **Frontend (Angular)**
  - Zuständig für die Erfassung der Nutzerinteraktionen (z. B. Klicks, Fehlermeldungen) und die Darstellung der Ergebnisse im Admin-Dashboard.
  - Setzt u. a. auf asynchrone Service-Aufrufe und reaktive Konzepte (RxJS), um Daten in Echtzeit oder in festgelegten Intervallen zu erfassen und an das Backend zu schicken.
- **Backend (Spring Boot)**
  - Stellt REST-Schnittstellen zur Verfügung, die vom Frontend konsumiert werden.
  - Übernimmt die Geschäftslogik, indem es eingehende Events (Klicks, Fehlermeldungen usw.) verarbeitet, validiert und in der Datenbank persistiert.
  - Aggregiert die Daten (z. B. zur Erstellung von Heatmaps oder zur Analyse von Navigationspfaden) und sendet die Ergebnisse zurück ans Frontend.
- **Datenhaltung (relationales Modell)**
  - Datenbanktabellen, in denen Sitzungen (Sessions), Klick-Events, Fehler-Events und Netzwerkabfragen erfasst werden.
  - Trennung zwischen dynamischen Nutzungsdaten (z. B. Klickereignissen) und statischen Entitäten (z. B. Komponenten-Beschreibungen), um eine klare Struktur und Erweiterbarkeit zu gewährleisten.

Diese Aufteilung ermöglicht eine lose Kopplung zwischen Frontend und Backend sowie eine modulare Erweiterbarkeit der Datenerfassung und Analyse.

2.2 Datenbankmodell

Zur Speicherung und Auswertung der erhobenen Interaktionsdaten kommt ein relationales Datenmodell zum Einsatz. Wesentliche Entitäten sind:

- **Usability Session**: Repräsentiert eine anonyme Sitzung und verknüpft diese mit einer konkreten Komponente (z. B. einem Modul der Anwendung). Enthält Start- und Endzeit sowie ein Session-Token als eindeutigen Bezeichner.
- **Click Event**: Zeichnet jede Klick-Interaktion innerhalb einer Session auf. Wichtige Felder sind das angeklickte UI-Element (`element_id`), eine Sequenznummer zur Rekonstruktion der Klickfolge und ein Zeitstempel.
- **Error Event**: Erfasst Fehler, die während einer Session auftreten (z. B. Ausnahmen im Frontend oder gemeldete Fehler über ein Formular). Ermöglicht die Kategorisierung und Zählung von Fehlern, um Problembereiche zu identifizieren.

- **Network Request**: Speichert Informationen zu einzelnen Netzwerkaufrufen wie Startzeit, Endzeit, HTTP-Methode und ggf. Antwortstatus. Wichtig zur Performance-Analyse und zum Auffinden von Engpässen.
- **Component**: Enthält Metadaten zu einer (Sub-)Anwendung bzw. UI-Komponente, sodass Sitzungen und Events eindeutig zugeordnet werden können.

Dieses Datenmodell ermöglicht eine gezielte Analyse von Nutzerwegen, Klickhäufigkeiten, Fehlerursachen und Performance-Daten, ohne dabei personenbezogene Daten zu erheben. Durch die Entkopplung zwischen „Sitzungen“ und konkreten Nutzern wird die Privatsphäre gewahrt.

2.3 Methodisches Vorgehen bei der Datenerfassung

- **Event Listener im Frontend**: In Angular-Komponenten werden Klick- und Fehlerevents über globale oder lokale Listener abgefangen. Die Daten (Element-ID, Zeitstempel, Session-Token usw.) werden minimal verarbeitet (z. B. Ergänzung eines Sequenzzählers) und anschließend an das Backend gesendet.
- **Persistierung und Anreicherung im Backend**: Das Spring-Boot-Backend nimmt die Daten entgegen, validiert sie und legt sie in den entsprechenden Tabellen ab (siehe Abschnitt **??**). Zudem können Zeitinformationen (z. B. `time_since_previous`) ergänzt oder bestehende Sessions automatisch geschlossen werden, wenn eine Inaktivität vorliegt.
- **Auswertung**: Um Heatmap-Daten, häufigste Pfade oder Fehlerstatistiken zu erzeugen, werden Aggregationsfunktionen und gruppierte Abfragen verwendet. Die Ergebnisse werden entweder direkt an das Frontend geliefert (z. B. für Live-Analysen) oder in periodischen Abständen abgerufen, um Berichte oder Dashboards zu aktualisieren.
- **Datenschutz**: Nur technische und kontextbezogene Daten werden gespeichert (z. B. Session-Token statt Nutzer-ID). Keine Speicherung von Klartext-Personendaten. Konzeption und Implementierung sind darauf ausgelegt, geltende Datenschutzvorgaben einzuhalten.

2.4 SQL-Datenbankmodell

```sql
CREATE TABLE click_event (
    id                  BIGINT NOT NULL PRIMARY KEY,
    element_id          VARCHAR(255),
    sequence_number     INTEGER NOT NULL,
    time_since_previous INTEGER,
    timestamp           TIMESTAMP,
```

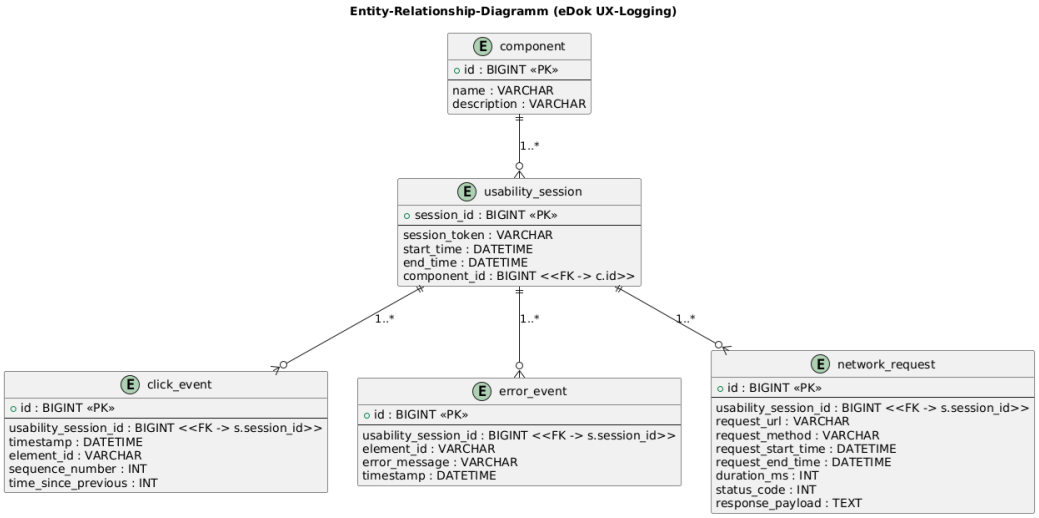

Abbildung 1: Datenbankmodell für das eDok UX-Logging

```
    usability_session_id BIGINT NOT NULL
);

CREATE TABLE component (
    id          BIGINT NOT NULL PRIMARY KEY,
    description VARCHAR(255),
    name        VARCHAR(255)
);

CREATE TABLE network_request (
    id                  BIGINT NOT NULL PRIMARY KEY,
    duration_ms         INTEGER,
    request_end_time    TIMESTAMP,
    request_method      VARCHAR(255),
    request_start_time  TIMESTAMP,
    request_url         VARCHAR(255),
    response_payload    CLOB,
    status_code         INTEGER,
    usability_session_id BIGINT NOT NULL
);

CREATE TABLE usability_session (
    session_id    INTEGER PRIMARY KEY,
    end_time      TIMESTAMP,
    session_token VARCHAR(255),
    start_time    TIMESTAMP,
    component_id  BIGINT NOT NULL
```

```
34 );
```

2.5 Zusammenfassung

Im Rahmen dieses Kapitels wurden die methodischen und technischen Grundlagen erläutert, welche für das Verständnis der späteren Umsetzung essenziell sind. Zentrale Punkte sind die lose gekoppelte Architektur (Angular-Frontend, Spring-Boot-Backend, relationale Datenbank), das klar strukturierte Datenmodell für Nutzungs- und Fehlerevents sowie das damit verbundene methodische Vorgehen bei der Datenerfassung und -auswertung.

Aufbauend auf diesen Grundlagen werden im folgenden Kapitel die konkreten Implementierungsdetails, Ablauflogiken und ausgewählte Code-Beispiele vorgestellt, um den Praxiseinsatz des Modells zu veranschaulichen. Längere Code-Auszüge erscheinen im Anhang und werden dort ausführlicher dokumentiert.

- **Kapitel 2** bietet einen ausführlichen Überblick über den theoretischen Hintergrund der UX-Messung, einschließlich gängiger Methoden und Techniken zur Datenerfassung und -analyse.
- In **Kapitel 3** wird das entwickelte Modell detailliert vorgestellt. Es werden die technischen Komponenten, die Datenbankstruktur sowie die Implementierungsdetails beschrieben.
- **Kapitel 4** widmet sich der Anwendung des Modells auf die *eDok*-Anwendung. Es werden die Ergebnisse der Datenerfassung präsentiert und analysiert.
- In **Kapitel 5** erfolgt eine kritische Diskussion der Ergebnisse, einschließlich der Vorteile des Modells, identifizierter Schwachstellen und möglichen Verbesserungen.
- **Kapitel 6** fasst die Arbeit zusammen und gibt einen Ausblick auf zukünftige Forschungs- und Entwicklungsmöglichkeiten.

2.6 Relevanz der Arbeit

Angesichts der wachsenden Bedeutung von UX in der Softwareentwicklung leistet diese Arbeit einen wichtigen Beitrag zur Praxis und Forschung. Das vorgestellte Modell ermöglicht es, UX-Daten in Echtzeit und unter realen Nutzungsbedingungen zu erfassen, ohne die Nutzererfahrung zu beeinträchtigen. Dies ist besonders relevant für Organisationen, die ihre Anwendungen kontinuierlich verbessern möchten, um den steigenden Erwartungen der Nutzer gerecht zu werden.

Durch die Anwendung auf die *eDok*-Anwendung des LWV Hessen wird demonstriert, wie das Modell in einer realen Umgebung implementiert werden kann. Die gewonnenen Erkenntnisse können als Grundlage für weitere Anwendungen dienen und dazu beitragen, die UX-Forschung in der Praxis voranzutreiben.

2.7 Datenschutz und DSGVO

Im Rahmen der Datenerfassung werden ausschließlich *technische* und *kontextbezogene* Informationen aufgezeichnet, die keine Rückschlüsse auf einzelne Personen zulassen. Die erhobenen Nutzungsdaten sind *anonymisiert* bzw. *pseudonymisiert*. Dazu wird jedem Nutzungsvorgang lediglich eine `session_token` zugeordnet, statt personenbezogene Identifikationsmerkmale zu verwenden.

Um die **DSGVO**-Konformität zu gewährleisten, wird zudem kein Bezug zwischen den erhobenen Daten und Nutzerprofilen hergestellt. Informationen wie IP-Adressen, Klarnamen oder E-Mail-Adressen werden nicht erfasst. Die protokollierten Daten (z. B. Klick-Ereignisse, Fehlermeldungen, Sitzungszeiten) enthalten ausschließlich Metadaten zum Nutzerverhalten.

Hinsichtlich der **Speicherdauer** existiert eine interne Aufbewahrungsrichtlinie, die eine regelmäßige Löschung oder Archivierung älterer Daten vorsieht. Auf diese Weise wird der Datenumfang kontrolliert und eine unbegrenzte Speicherung vermieden. Bei Bedarf kann das Löschkonzept an neue gesetzliche Vorgaben angepasst oder durch *Privacy-by-Design*-Konzepte ergänzt werden.

2.8 Performance-Überlegungen

Um eine hohe *Performance* zu gewährleisten, werden die erhobenen Daten nicht bei jedem Klick *sofort* synchron an das Backend gesendet, sondern *asynchron* verarbeitet. Konkret bedeutet dies, dass das Frontend eingehende Events zunächst lokal zwischenspeichert und in festgelegten Zeitintervallen (*Batching*) an das Backend übermittelt. Dadurch wird die Anzahl der HTTP-Anfragen reduziert, was eine deutliche Entlastung des Systems zur Folge hat.

Innerhalb des Backends erfolgt die *Persistierung* ebenfalls in asynchronen Prozessen, sodass umfangreiche `INSERT`- und `UPDATE`-Operationen nicht blockierend wirken. Diese Architektur stellt sicher, dass die Anwendung auch bei hohem Datenaufkommen performant bleibt und keine spürbaren Verzögerungen im Nutzerfluss auftreten. Bei Bedarf können die Zeitintervalle angepasst werden, um ein Gleichgewicht zwischen Datenaktualität und Systemlast herzustellen.

Die systematische Erfassung und Analyse der UX ist unerlässlich, um moderne Softwareanwendungen an die Bedürfnisse und Erwartungen der Nutzer anzupassen. Durch die Entwicklung und Implementierung eines Modells zur UX-Messung in *eDok* wird gezeigt, wie Daten genutzt werden können, um die Nutzererfahrung zu verbessern und die Anwendung effizienter und benutzerfreundlicher zu gestalten. Diese Arbeit leistet

somit einen wertvollen Beitrag zur Verbesserung von UX-Methoden und deren Anwendung in der Praxis.

3 Einführung in das Thema UX

Die **User Experience** (UX) ist ein Teilbereich der HCI, der in den frühen 1990er Jahren an Bedeutung gewann [**glanznig**]. Mit dem Aufkommen neuer Technologien und der zunehmenden Verbreitung digitaler Produkte rückte der Fokus verstärkt auf die Benutzererfahrung und deren Qualität. UX konzentriert sich auf die Interaktion zwischen Mensch und System und zielt darauf ab, Produkte und Anwendungen so zu gestalten, dass sie nicht nur funktional sind, sondern auch emotional ansprechend und benutzerfreundlich.

3.1 Herausforderungen und Offenheit der UX-Definition

Die Offenheit der UX-Definition erlaubt es, das Thema aus verschiedenen Perspektiven zu betrachten und unterschiedliche Meinungen einzubeziehen, was zu einem umfassenderen und tieferen Verständnis des UX-Konzepts führt und die Entwicklung der Disziplin fördert. Gleichzeitig erschweren jedoch verschiedene fachliche Hintergründe und Vokabularien den Fortschritt [**glanznig**].

Hassenzahl und Tractinsky thematisierten in ihrer Arbeit *User Experience – A Research Agenda* die Komplexität der UX und erklärten: „User Experience ist ein interessantes Phänomen: Es wurde von der HCI-Community schnell angenommen, aber oft kritisiert, da es vage und schwer fassbar ist. Der Begriff umfasst verschiedene Bedeutungen, von klassischer Usability bis hin zu Schönheit, hedonischen, affektiven und erfahrungsbasierten Aspekten der Technologienutzung“ [**research**].

3.2 Forschungsperspektiven: UX als vielschichtiges Phänomen

Sie erläutern weiter, dass sich die frühe Forschung im Bereich der HCI hauptsächlich auf Verhaltensziele in Arbeitsumgebungen konzentrierte. Dieser Fokus wurde jedoch später durch alternative Ansätze infrage gestellt, die die Bedeutung von Ästhetik, Emotionen und subjektiven Erfahrungen betonten [**research**].

In einem frühen Versuch, UX zu definieren, betonte Alben (1996) die Bedeutung von Ästhetik als wesentlichem Qualitätsaspekt von Technologie [**research**]. Hassenzahl argumentierte, dass interaktive Produkte aus zwei Perspektiven betrachtet werden können: den instrumentellen Aspekten (z. B. Usability) und den nicht-instrumentellen (hedonischen) Aspekten, die sich auf das emotionale und ästhetische Erleben der Nutzer beziehen [**hassenzahl2003**].

Die Nielsen Norman Group definiert UX als die Gesamtheit aller Aspekte der Interaktion eines Endnutzers mit einem Unternehmen, seinen Dienstleistungen und Produkten. Hervorgehoben wird, dass herausragende UX nicht nur die spezifischen Bedürfnis-

se des Nutzers erfüllt, sondern auch durch Einfachheit und Eleganz überzeugt, sodass die Nutzung und der Besitz des Produkts als angenehm empfunden werden. Zudem wird betont, dass eine qualitativ hochwertige UX eine nahtlose Integration verschiedener Disziplinen erfordert, darunter Ingenieurwesen, Marketing, Grafik- und Industriedesign sowie Interface-Design [**nngroup**].

Die ISO 9241-210:2019 beschreibt UX als „sämtliche Wahrnehmungen und Reaktionen von Nutzern, die durch die tatsächliche oder erwartete Nutzung eines Systems, Produkts oder einer Dienstleistung hervorgerufen werden“ [**ISO**].

3.3 Zusammenfassung der UX-Definitionen

Ungeachtet der unterschiedlichen Definitionen ist klar, dass UX sowohl funktionale als auch emotionale Dimensionen der Nutzererfahrung umfasst. Faktoren wie Usability, ästhetische Wahrnehmung, inhaltliche Relevanz und das Vertrauen der Nutzer sind entscheidend dafür, wie effektiv und zufriedenstellend eine Anwendung wahrgenommen wird und tragen wesentlich zur Gestaltung positiver Nutzererlebnisse bei [**toolbox**].

In dem Papier *Towards Practical User Experience Evaluation Methods* [**evaluationmethods**] wird darauf hingewiesen, dass die UX-Forschung eine Vielzahl von Modellen und Frameworks entwickelt hat. Diese Modelle adressieren die zentralen Herausforderungen der UX, wie ihre subjektive, kontextabhängige und dynamische Natur sowie die Balance zwischen pragmatischen und hedonischen Aspekten des Nutzererlebnisses. Gleichzeitig wird betont, dass UX zunehmend in der Industrie übernommen wird, die Produktentwicklung jedoch noch immer stark auf traditionellen Usability-Methoden basiert. Die Arbeit unterstreicht die Notwendigkeit praxisnaher UX-Bewertungsmethoden für die Produktentwicklung in der Industrie.

3.4 Usability und ihre Bedeutung

Usability bezieht sich auf die funktionalen Aspekte eines Produkts und beschreibt, wie effektiv und effizient sich das Produkt nutzen lässt. Viele Experten sehen Usability als Teil der UX [**GOISTAI**]. Während sich Usability auf die Fähigkeit des Nutzers konzentriert, ein System erfolgreich zu nutzen, umfasst UX auch die gesamte Nutzererfahrung und die dabei entstehenden Emotionen.

Die Nielsen Norman Group definiert Usability als ein Qualitätsmerkmal, das bewertet, wie einfach und angenehm eine Benutzeroberfläche zu nutzen ist. Sie umfasst fünf Hauptkomponenten:

- **Erlernbarkeit**: Wie leicht können Benutzer grundlegende Aufgaben beim ersten Mal ausführen?
- **Effizienz**: Wie schnell können Benutzer Aufgaben ausführen, nachdem sie die Schnittstelle gelernt haben?

- **Merkfähigkeit**: Wie einfach können Benutzer ihre Fähigkeiten wiederherstellen, wenn sie das Design nach einer Pause erneut verwenden?
- **Fehlerrate**: Wie viele Fehler machen Benutzer, wie schwerwiegend sind diese und wie leicht können sie sich von ihnen erholen?
- **Zufriedenheit**: Wie angenehm ist die Nutzung der Benutzeroberfläche?

Die Nielsen Norman Group erläutert den Unterschied zwischen Usability Testing und Usability Evaluation klar und prägnant. Beim Usability Testing beobachten Usability-Experten die Nutzer, um potenzielle Usability-Probleme zu identifizieren. Im Gegensatz dazu geht die Usability Evaluation einen Schritt weiter: Sie kombiniert das Testing mit direktem Nutzerfeedback und einer Überprüfung des Systemdesigns.

3.5 Usability in eDok

Im Kontext von *eDok* liegt der Schwerpunkt auf der effizienten und fehlerarmen Erstellung und Bearbeitung von Dokumenten. Für optimale Usability ist es entscheidend, dass Nutzer die Funktionen der Anwendung intuitiv verstehen und nutzen können. Klare Benutzerführung, schnelle Ladezeiten sowie eine übersichtliche und reduzierte Benutzeroberfläche sind dabei zentral.

Ein wesentlicher Aspekt ist die Unterstützung von Fehlertoleranz, sodass auch bei Eingabefehlern ein produktiver Arbeitsfluss erhalten bleibt. Funktionen wie automatisches Speichern, leicht zugängliche Rückgängig-Optionen und verständliche Fehlermeldungen helfen, die Nutzererfahrung positiv zu gestalten.

Durch die kontinuierliche Evaluation der Usability über systematisch gesammelte Daten können Schwachstellen identifiziert und gezielt Optimierungen vorgenommen werden. Ergänzende Nutzerbefragungen können zusätzliche Einblicke bieten, sind jedoch nicht direkt in das System integriert und dienen daher eher unterstützenden Zwecken.

Insgesamt zielt *eDok* darauf ab, eine nutzerfreundliche Anwendung zu bieten, die den Arbeitsablauf bei der Dokumentenerstellung optimiert. Diese Arbeit strebt an, bestimmte Merkmale der Usability zu messen, zu speichern und zu bewerten, um ein umfassendes Bild der Nutzererfahrung zu erhalten.

3.6 Erweiterte Ziele der UX- und Usability-Bewertung in eDok

Das Ziel der UX- und Usability-Bewertung in *eDok* ist es, gezielte Einblicke in die Nutzungsmuster der Anwendung zu gewinnen und Optimierungspotenziale klar zu identifizieren. Das entwickelte Modell liefert durch systematische Erfassung und Analyse der Nutzerinteraktionen Daten, die Entwickler und Produktverantwortliche dabei unterstützen, Nutzerbedürfnisse und häufige Herausforderungen sichtbar zu machen.

Die implementierte Heatmap bietet eine visuelle Übersicht der am stärksten genutzten Bereiche, sodass gezielt Verbesserungspotenziale an zentralen Interaktionspunkten

erkannt werden können. Durch die Analyse der Verweildauer auf spezifischen Seiten, der häufig durchlaufenen Navigationspfade und der Häufigkeit bestimmter Fehler können gezielte Maßnahmen zur Optimierung der Anwendung abgeleitet werden.

Diese Daten helfen, die Effizienz und Benutzerfreundlichkeit der Anwendung kontinuierlich zu steigern und eine datengetriebene Produktoptimierung zu ermöglichen. Darüber hinaus ermöglicht das Modell eine nachträgliche Überprüfung der implementierten Änderungen, um deren Effektivität zu bewerten und weitere Optimierungen vorzunehmen. Durch diesen iterativen Prozess trägt das Modell wesentlich zur Steigerung der Nutzerzufriedenheit und Effizienz der Anwendung bei.

3.6.1 Abgrenzung von UX- und Usability-Evaluationsmethoden

Während Usability-Tests sich primär auf die Leistung bei der Aufgabenbearbeitung konzentrieren, fokussieren UX-Evaluationsmethoden auf das subjektive Erleben der Nutzenden. Objektive Metriken wie Ausführungszeit oder Klickanzahl reichen nicht aus, um die UX vollständig zu erfassen; vielmehr müssen auch Motivation, Erwartungen und Emotionen der Nutzenden berücksichtigt werden [**dev**].

Nach eingehender Recherche und Analyse der verschiedenen UX-Evaluationsmethoden hat sich herausgestellt, dass die Anforderungen dieser Arbeit eher einem Usability-Test entsprechen als traditionellen UX-Evaluationsmethoden wie Fragebögen oder der Beobachtung von Nutzenden. Unsere Arbeit konzentriert sich auf die systematische Erfassung von Nutzungsdaten während des Produktiveinsatzes der Anwendung. Diese Herangehensweise erlaubt es, objektive Daten über die Interaktion der Nutzenden mit der Anwendung zu sammeln und spezifische Aspekte der Usability zu bewerten.

3.6.2 Fokussierung auf Usability in dieser Arbeit

Die Entscheidung, den Schwerpunkt auf Usability-Tests zu legen, basiert auf den spezifischen Anforderungen des Projekts. Da die gesammelten Daten nicht benutzerorientiert sein sollen und keine direkten Nutzerbefragungen oder Beobachtungen stattfinden, sind traditionelle UX-Evaluationsmethoden weniger geeignet. Stattdessen ermöglicht die Messung von Interaktionsdaten, wie zum Beispiel Klickpfade, Verweildauer auf bestimmten Seiten und Fehlerhäufigkeiten, eine objektive Analyse der Benutzerfreundlichkeit der Anwendung.

Diese Daten liefern wertvolle Erkenntnisse darüber, wie effizient und effektiv die Nutzenden die Anwendung bedienen können. Sie helfen dabei, potenzielle Usability-Probleme zu identifizieren und gezielte Verbesserungen vorzunehmen. Obwohl dieser Ansatz nicht das gesamte Spektrum der UX abdeckt, trägt er wesentlich zur Optimierung der Anwendung bei und unterstützt die Nutzenden in ihrer täglichen Arbeit.

Jordan (2008)[**jordan2008auswahl**] betont die Bedeutung einer sorgfältigen Auswahl der Usability-Evaluationsmethode unter Berücksichtigung spezifischer Projektanforderungen und Kontextfaktoren. In Anlehnung an seine Kriterien hat sich herausgestellt, dass eine Methode, die auf der automatisierten Erfassung von Nutzungsdaten basiert,

geeignet ist. Diese Entscheidung ermöglicht es, objektive Daten zu sammeln, ohne die Privatsphäre der Nutzenden zu beeinträchtigen, und entspricht somit den praktischen und ethischen Anforderungen des Projekts.

Im Jahr 1980 führte das Xerox Palo Alto Research Center (PARC)[**keystroke**] eine bahnbrechende Studie durch. Sie verglichen die vorhergesagten Ausführungszeiten mit den tatsächlich gemessenen Zeiten und stellten fest, dass das Keystroke-Level Model KLM die Benutzerleistung mit einer Genauigkeit von etwa 21% vorhersagen konnte. Damit zeigten sie, dass ihr Modell ein nützliches Werkzeug für Designer von interaktiven Systemen ist, um die Effizienz von Benutzerschnittstellen quantitativ zu bewerten und zu verbessern.

Die von Paz und Pow-Sang durchgeführte Studie [**Paz2016**] identifizierte die am häufigsten verwendeten Methoden zur Evaluierung der Benutzerfreundlichkeit (Usability Evaluation Methods, UEMs) in Softwareentwicklungsprozessen:

- **Umfragen/Fragebögen (26,26%)**
  - Die am häufigsten verwendete Methode.
  - Geschätzt für ihre Einfachheit und Effektivität bei der Erfassung von Benutzerzufriedenheitsdaten.
- **Benutzertests (14,14%)**
  - Beinhaltet die Beobachtung realer Benutzer, die mit der Software interagieren.
  - Ziel: Direkte Identifikation von Usability-Problemen.
- **Heuristische Evaluation (12,63%)**
  - Experten bewerten die Software anhand etablierter Usability-Prinzipien.
  - Ziel: Potenzielle Probleme aufdecken.
- **Interviews (10,35%)**
  - Direkte Gespräche mit Benutzern.
  - Ziel: Tiefgehende Einblicke in Usability-Bedenken und -Erfahrungen.
- **Lautes Denken (9,60%)**
  - Benutzer äußern ihre Gedanken während der Nutzung der Software.
  - Ziel: Echtzeit-Feedback zu ihren Interaktionen liefern.

Im Kontext der Ergebnisse der Studie, obwohl diese Methoden nicht zu den fünf am häufigsten verwendeten zählen (diese sind Umfrage/Fragebogen, Benutzertests, heuristische Evaluation, Interview und "Thinking AloudProtokoll), werden sie dennoch als anerkannte und wertvolle Techniken in der Usability-Evaluierung betrachtet.

Durch den Einsatz dieser Methoden verwenden Sie quantitative Datenanalyse, um die Benutzerfreundlichkeit zu evaluieren. Dieser Ansatz ermöglicht es, spezifische Probleme in der Benutzeroberfläche basierend auf dem tatsächlichen Nutzerverhalten zu identifizieren, was zu gezielteren und effektiveren Verbesserungen führt.

Die Studie von Srivastava et al. (2000)[**Srivastava2000**] untersucht das Konzept des *Web Usage Mining*, bei dem Data-Mining-Techniken angewendet werden, um Nutzungsmuster aus Web-Daten zu entdecken. Ziel ist es, das Verhalten von Web-Nutzern zu verstehen und daraus Erkenntnisse für die Verbesserung von Web-Anwendungen zu gewinnen.

Die Autoren identifizieren verschiedene Arten von Web-Daten, darunter Inhaltsdaten, Strukturdaten, Nutzungsdaten und Benutzerprofildaten. Sie beschreiben einen dreistufigen Prozess des Web Usage Mining:

1. **Datenvorverarbeitung**: Reinigung und Transformation der Rohdaten, Identifikation von Benutzern und Sessions.

2. **Musterdetektion**: Anwendung von Data-Mining-Techniken wie Assoziationsanalyse und Clustering zur Identifikation von Nutzungsmustern.

3. **Musteranalyse**: Interpretation der entdeckten Muster, um nützliche Erkenntnisse zu gewinnen.

Die Studie zeigt, dass die Analyse von Nutzungsdaten wie Heatmaps, Fehlerzählungen, Klickanalysen und Verweildauer entscheidend ist, um das Benutzerverhalten zu verstehen und die Usability von Websites zu verbessern. Durch Web Usage Mining können spezifische Probleme in der Benutzeroberfläche identifiziert und gezielte Verbesserungen vorgenommen werden.

Die Autoren fanden heraus, dass Web Usage Mining in verschiedenen Anwendungsbereichen wie Personalisierung, Systemoptimierung, Website-Modifikation und Geschäftsanalysen wertvolle Einblicke liefert. Sie betonen jedoch auch die Herausforderungen, insbesondere in Bezug auf Datenqualität und Datenschutz.

Insgesamt unterstreicht die Studie die Bedeutung von Web Usage Mining als effektives Werkzeug zur Verbesserung von Web-Anwendungen durch ein besseres Verständnis des Nutzerverhaltens.
Diese Arbeit befasste sich mit der Sammlung und Darstellung von Daten, es wurden jedoch keine Data-Mining-Techniken angewendet. Es ist jedoch üblich, solche Methoden auf die gesammelten Daten anzuwenden, was zu interessanteren Erkenntnissen führen kann, wie in der Studie erwähnt.
In dieser Arbeit wurden stattdessen statistische Operationen durchgeführt, um einige Kennzahlen wie die Fehlerquote zu veranschaulichen.

3.6.3 Vergleich von Nutzerinterviews, Fragebögen und der Sammlung quantitativer Daten

In der Usability-Forschung gibt es verschiedene Methoden, um Nutzererfahrungen und -probleme zu evaluieren. Zu den häufig eingesetzten Ansätzen gehören qualitative Methoden wie Interviews und Fragebögen, die direkte Rückmeldungen der Nutzer einholen, und quantitative Ansätze wie das Logging und die Analyse von Nutzungsdaten. Nach eingehender Analyse und Berücksichtigung der Ergebnisse aus dem Papier *Extracting Usability Information from User Interface Events* von Hilbert und Redmiles [**Hilbert2000**] wurde entschieden, dass eine Kombination dieser Methoden die umfassendsten Erkenntnisse liefert.

3.6.4 Nutzerinterviews und Fragebögen

Nutzerinterviews und Fragebögen bieten eine direkte Möglichkeit, subjektive Rückmeldungen zu sammeln. Die Vorteile dieser Ansätze sind:

- **Qualitative Einblicke:** Diese Methoden liefern tiefgehende Informationen zu Meinungen, Zufriedenheit und Emotionen der Nutzer.
- **Flexibilität:** Während der Interaktion können gezielte Fragen gestellt werden, um spezifische Probleme zu beleuchten.
- **Direktes Feedback:** Nutzer äußern direkt wahrgenommene Schwächen oder Verbesserungsvorschläge.

Dennoch zeigen sich auch wesentliche Nachteile:

- **Subjektivität:** Ergebnisse hängen stark von individuellen Meinungen und Wahrnehmungen der Nutzer ab.
- **Hoher Aufwand:** Diese Methoden erfordern erheblichen Zeit- und Ressourcenaufwand.
- **Begrenzte Skalierbarkeit:** Sie sind für kleine Nutzergruppen geeignet, aber schwer auf größere Stichproben anzuwenden.
- **Fehlende Objektivität:** Die Erfassung tatsächlichen Nutzerverhaltens ist begrenzt.

3.7 Sammlung und Analyse quantitativer Daten

Der Ansatz von Hilbert und Redmiles [**Hilbert2000**] beschreibt die Vorteile der Erfassung von Nutzungsdaten, wie Mausbewegungen, Klicks und Navigationspfade, über automatisiertes Logging. Die wesentlichen Vorteile dieses Ansatzes sind:

- **Objektivität:** Die Erfassung basiert auf realem Nutzerverhalten und eliminiert subjektive Verzerrungen.

- **Automatisierung:** Daten können kontinuierlich und effizient gesammelt werden.
- **Skalierbarkeit:** Die Methode eignet sich für große Nutzerzahlen und breite Analysen.
- **Erkennung von Mustern:** Systematische Probleme, wie ineffiziente Workflows oder häufige Fehler, können identifiziert werden.

Allerdings gibt es auch Einschränkungen:

- **Fehlende qualitative Einblicke:** Emotionen oder Zufriedenheit der Nutzer können nicht direkt erfasst werden.
- **Komplexe Analyse:** Die Interpretation der Daten erfordert spezialisierte Expertise.
- **Abhängigkeit von Logging-Systemen:** Eine unzureichende Erfassung kann die Ergebnisse verzerren.

3.8 Erkenntnisse und Schlussfolgerung

Die Ergebnisse aus dem Papier von Hilbert und Redmiles [**Hilbert2000**] zeigen, dass die Kombination von qualitativen und quantitativen Ansätzen essenziell ist, um umfassende Usability-Einsichten zu gewinnen. Während Interviews und Fragebögen subjektive Erlebnisse und Zufriedenheit der Nutzer erfassen, bietet die Analyse quantitativer Daten objektive und skalierbare Einblicke in tatsächliches Nutzerverhalten. Das Event-Logging ermöglichte es, systematische Probleme zu identifizieren, die bei traditionellen Methoden übersehen wurden, wie ineffiziente Navigationspfade oder wiederkehrende Fehler.

Nach Prüfung der Ergebnisse wurde die Entscheidung getroffen, dass qualitative und quantitative Methoden nicht gegeneinander stehen, sondern sich ergänzen. Die Kombination beider Ansätze stellt sicher, dass sowohl die subjektiven als auch die objektiven Aspekte der Usability umfassend berücksichtigt werden können.

3.9 Methodik

Die in dieser Arbeit entwickelte Methodik zur Erfassung, Speicherung und Analyse von UX- und Usability-Daten in der Anwendung *eDok* umfasst mehrere aufeinander abgestimmte Schritte. Sie gewährleistet eine datenschutzkonforme, modulare und erweiterbare Infrastruktur, um Nutzungsinteraktionen systematisch zu sammeln, zu aggregieren und visuell darzustellen.

3.9.1 Systemarchitektur und Datenerfassung

Die Datenerfassung erfolgt clientseitig in der Frontend-Anwendung (Angular), wobei bei Nutzungsinteraktionen, wie etwa Mausklicks, Navigationsaktionen oder Fehlermeldungen, entsprechende Ereignisse (Events) abgefangen und in strukturierter Form an den Backend-Server (Spring Boot) gesendet werden. Diese Vorgehensweise ermöglicht eine kontinuierliche Erfassung von Interaktionsdaten während des Produktiveinsatzes, ohne dass Nutzer zusätzliche Handlungen vornehmen müssen.

3.9.2 Datenpersistenz und Datenbankmodell

Für die Speicherung der erfassten Daten wird eine relationale Datenbank eingesetzt, in der die zentralen Entitäten und ihre Beziehungen abgebildet sind. Hierfür kommen die Tabellen `click_event`, `component`, `error_event`, `network_request`, `ui_element` sowie `usability_session` zum Einsatz. Dabei bilden diese Tabellen folgende Kernaspekte der Datenerfassung ab:

- `usability_session`: Enthält Sitzungsinformationen, wie etwa Start- und Endzeitpunkte sowie einen eindeutigen Session-Token. Diese Tabelle dient als Ankerpunkt für die Zuordnung sämtlicher Interaktionsdaten.
- `click_event`: Erfasst sämtliche Klickinteraktionen, inklusive Zeitstempel, betroffenen `ui_element`-Referenzen und Sequenznummern, um Reihenfolgen nachvollziehbar zu machen.
- `error_event`: Dokumentiert aufgetretene Fehlerzustände, deren Art, Häufigkeit und Position. Diese Daten ermöglichen es, gezielt fehleranfällige Bereiche der Anwendung zu identifizieren.
- `network_request`: Speichert Informationen zu Netzwerkaufrufen (Ladezeiten und Fehlermeldungen), um mögliche Performance-Engpässe und technische Probleme sichtbar zu machen.
- `component` und `ui_element`: Beschreiben statische Strukturen der Anwendung, indem sie Komponenten und deren zugehörige UI-Elemente referenzieren. Diese Aufteilung ermöglicht eine flexible Identifikation der Interaktionspunkte.

Die in den Tabellen erfassten Daten können durch Filterung, Aggregation und Zeitreihenanalysen zu aussagekräftigen Metriken verdichtet werden. Diese Datenmodellierung gewährleistet eine klare Trennung von dynamischen Nutzungsdaten und statischen UI-Strukturen, wodurch sowohl technische als auch inhaltliche Veränderungen im Frontend flexibel abbildbar bleiben.

3.9.3 Backend-Logik und REST-Schnittstellen

Das Backend stellt REST-Schnittstellen bereit, um Nutzungsdaten entgegenzunehmen, abzufragen und für die Visualisierung aufzubereiten. Bei Eintreffen neuer Datensätze führt die Backend-Logik Validierungs- und Anreicherungsprozesse durch, bevor die Daten in die entsprechende Tabelle geschrieben werden. Zudem bietet das Backend administrativen Nutzern Endpunkte zur Konfiguration des Erfassungsmodus, um etwa die Aufzeichnung auf spezifischen Seiten ein- oder auszuschalten.

3.9.4 Anonymisierung und Datenschutz

Um den Schutz der Privatsphäre der Nutzenden sicherzustellen, werden alle erhobenen Daten strikt anonymisiert. Es werden keine personenbezogenen Merkmale gespeichert. Die Datensätze lassen keine Rückschlüsse auf einzelne Individuen zu, sondern konzentrieren sich auf aggregierte Nutzungsmuster, Fehlerhäufigkeiten und Interaktionspfade.

3.9.5 Visualisierung und Heatmap-Darstellung

Die aufbereiteten Daten werden im Admin-Dashboard visualisiert. Hierfür wurden unter anderem Heatmaps implementiert, welche die am stärksten genutzten Bereiche der Benutzeroberfläche hervorheben. Durch die Einbettung von Sequenzinformationen (Reihenfolge der Klickereignisse) können Navigationspfade und Nutzungskontexte ermittelt werden. Fehlerereignisse werden gesondert markiert und helfen dabei, potenziell problematische UI-Elemente oder Prozessschritte zu identifizieren.

3.9.6 Iterative Verbesserung und Anpassung

Die vorgestellte Methodik ist iterativ angelegt: Anhand der gewonnenen Erkenntnisse lassen sich gezielte Verbesserungsmaßnahmen im UI-Design, in den Navigationsstrukturen oder in der Performance der Anwendung ableiten. Nach erfolgten Anpassungen können erneut Daten erhoben und mit früheren Ergebnissen verglichen werden, um die Effektivität der Verbesserungen zu bewerten und den Optimierungsprozess fortlaufend zu steuern.

4 Kapitel 3

4.1 Anforderungen

- Ein Dashboard zur Darstellung von spezifischen UX- und Usability-Berichten, die durch das Modell erfasst wurden.
- Nur Benutzer mit Administratorrechten können auf das Dashboard zugreifen.

- Der Administrator hat die Möglichkeit, das Modell auf beliebigen Seiten ein- oder auszuschalten.
- Erfassung der Zeit, die Nutzer auf einer Seite verbringen, zur Analyse des Nutzeraufwands.
- Eine Heatmap mit Reihenfolgenverfolgung, um bestimmte Verhaltensmuster der Nutzer zu erkennen und visuell darzustellen.
- Zusätzlich zur Reihenfolgen-Funktion sollen Bearbeitungs-Eingabefelder erkannt und angezeigt werden.
- Die Heatmap wird dem Administrator im Dashboard zur Verfügung gestellt.
- Fehler werden erfasst, gespeichert und im Dashboard angezeigt, wobei die Anzahl und die Position der Fehler besonders hervorgehoben werden.

4.2 Spezifikation der Erfassten UX-Daten in eDok

Zur systematischen Bewertung der Usability von eDok wird die Erfassung und Analyse spezifischer Datenarten priorisiert. Im Mittelpunkt steht die Sammlung von Interaktionsdaten, die durch den Einsatz von Heatmaps visualisiert werden. Diese ermöglichen detaillierte Einblicke in Navigationsmuster, identifizieren potenzielle Stolpersteine und decken ineffiziente Seitengestaltungen sowie suboptimale Darstellungen innerhalb der Anwendung auf. Ziel ist es, Schwachstellen in der Nutzerführung zu erkennen und datenbasierte Optimierungen der Benutzeroberfläche zu ermöglichen.

4.3 Implementierung der Heatmap

Diese Sektion beschreibt die Gestaltung und Implementierung einer Heatmap zur Nachverfolgung und Visualisierung von Nutzerinteraktionen in der eDok-Anwendung. Die Heatmap dient dazu, häufig genutzte Bereiche, Nutzerverhaltensmuster und fehleranfällige Interaktionen zu identifizieren, indem sie Daten zu Klicks, Fehlern und zeitlichen Metriken erfasst und darstellt.

4.4 Zweck und Ziele der Heatmap

Die Heatmap bietet eine visuelle Darstellung des Nutzerverhaltens und unterstützt Entwickler*innen dabei, die am häufigsten verwendeten Bereiche zu erkennen und Usability-Probleme zu identifizieren. Die Anzeige der Klickreihenfolge über Linienverbindungen hilft dabei, den Navigationsfluss innerhalb der Anwendung besser zu verstehen. Fehlerpunkte werden zudem in der Heatmap hervorgehoben und in einem separaten Admin-Dashboard für umfassende Berichte angezeigt. So können schwerwiegende Fehlerquellen identifiziert und priorisiert behoben werden.

4.5 Datenerfassung und Speicherung

Um die Heatmap-Funktionalität zu realisieren, werden verschiedene Arten von Interaktionsdaten erfasst und in einer strukturierten Datenbank gespeichert. Das Datenbankschema wurde so gestaltet, dass es zeitbasierte Nachverfolgung, Fehleraufzeichnung und Vergleich zwischen verschiedenen Versionen ermöglicht.

4.6 Implementierungsstrategie

Die nachfolgenden Abschnitte beschreiben, wie der Heatmap-Modus technisch umgesetzt wird. Neben einem zentralen *Angular-Service* (**Global Heatmap Service**) kommen eine *Overlay-Direktive* sowie ein *Admin-Schalter* zum Einsatz, um die Klick- und Nutzungsdaten erfassen und visualisieren zu können.

4.6.1 Global Heatmap Service

Dieser Service übernimmt:

- Das Aktivieren und Deaktivieren des Heatmap-Modus.
- Das Abfragen relevanter Klickdaten vom Backend.
- Das Bereitstellen einer Struktur, die UI-Elemente (`element_id`) mit ihren Klickhäufigkeiten verknüpft.
- Die Weiterleitung von Änderungen (z. B. neuer Daten oder geänderter Status) an abonnierende Komponenten und Direktiven.

4.6.2 Overlay Directive

Die Overlay-Direktive wird in das DOM eingefügt und:

- Durchsucht die Seite nach Elementen mit spezifischen IDs.
- Erzeugt Overlays (z. B. eingefärbte Markierungen) auf Basis der Daten aus dem Global Heatmap Service.
- Entfernt diese Overlays bei Deaktivierung des Heatmap-Modus.

4.6.3 Admin Toggle Control

Ein einfacher Administrations-Schalter erlaubt es:

- Den Heatmap-Modus über einen Aufruf im Global Heatmap Service zu (de)aktivieren.
- Allen abhängigen Komponenten und Direktiven den neuen Status mitzuteilen.

4.6.4 Beispiele für Auswertungen

Aus den gesammelten *Klick-* und *Fehlerdaten* lassen sich unter anderem folgende Auswertungen ableiten:

- **Klickhäufigkeit pro UI-Element** zur Identifikation besonders häufig genutzter Bereiche.
- **Fehlerquote** (Fehler pro Zeit oder pro Klick), um problematische Formulare oder Buttons zu erkennen.
- **Navigationspfade** (Klick-Sequenzen), die eine Übersicht typischer Benutzerwege liefern.
- **Verweildauer pro Seite**, um Arbeitsaufwände und potenzielle UI-Hürden sichtbar zu machen.

4.7 Umsetzung – Backend

Dieser Abschnitt beschreibt die wichtigsten Komponenten des Backends und zeigt, wie die Daten aus der Anwendung *eDok* entgegengenommen, verarbeitet und für die Analyse zur Verfügung gestellt werden. Das Backend basiert auf **Spring Boot** und stellt **REST-Schnittstellen** bereit, mit denen das Angular-Frontend kommunizieren kann. Die Kernstücke bilden dabei die *DAO*-Schicht (Data Access Objects) und die *Controller*, die die eingehenden Anfragen steuern und die entsprechenden DAO-Methoden aufrufen.

4.8 Datenbankentitäten und Objekt-Relationales Mapping

Um die zuvor vorgestellten Tabellen (siehe Abschnitt **??**) in Java zu repräsentieren, werden im Backend **JPA-Entities** verwendet. Dadurch entsteht eine klare und *wartbare* Verbindung zwischen den Datenbankstrukturen und dem Anwendungscode. Die folgende Übersicht zeigt, wie die zentralen Tabellen in Form von Entity-Klassen abgebildet wurden.

4.8.1 Vorteile des objekt-relationalen Mappings (ORM)

- **Abstraktion:** Anstatt rohe SQL-Anweisungen zu schreiben, können Entwickler*innen mit *Java-Objekten* arbeiten. JPA (Jakarta Persistence API) oder *Hibernate* erledigen die Umwandlung (Mapping) zwischen Datenbanktabellen und Objektstrukturen.
- **Konsistente Datenmodelle:** Änderungen am Datenmodell (z. B. neue Felder) lassen sich an einer zentralen Stelle (Entity-Klasse) pflegen. Die Tabellendefinition kann (je nach Konfiguration) automatisch oder halbautomatisch aktualisiert werden.

- **Bidirektionale Beziehungen:** Relationen wie „Eine Session hat viele KlickEvents“ werden als `@OneToMany` und `@ManyToOne` annotiert, was die Navigation zwischen Objekten (Session ↔ KlickEvent) erheblich erleichtert.

4.8.2 Beispiel: `ClickEvent`

```java
@Entity
@Table(name = "click_event")
public class ClickEvent {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "usability_session_id", nullable = false)
    @JsonManagedReference
    private UsabilitySession usabilitySession;

    @Column(name = "timestamp", nullable = false)
    private LocalDateTime timestamp;

    @Column(name = "element_id")
    private String elementId;

    @Column(name = "sequence_number", nullable = false)
    private int sequenceNumber;

    @Column(name = "time_since_previous")
    private Integer timeSincePrevious;

    // Getters/Setters ...
}
```

Diese Entity-Klasse repräsentiert die „`click_event`“-Tabelle. Wichtige Aspekte sind:

- `@Entity` und `@Table`: Kennzeichnen, dass die Klasse in einer Datenbanktabelle `click_event` persistiert wird.
- **Primärschlüssel**: `id` ist als `@Id` deklariert und wird über `GenerationType.IDENTITY` automatisch generiert.
- **Fremdschlüssel**: `@ManyToOne` auf `UsabilitySession` bildet die Beziehung „Viele Klick-Events gehören zu einer Session“ ab. Der konkrete Spaltenname in der Datenbank lautet `usability_session_id`.

- `sequenceNumber` und `timeSincePrevious` erlauben es, *Reihenfolge* und *Zeitabstände* der Klicks zu analysieren.

4.8.3 Weitere Entity-Klassen

`Component` Die `Component`-Entity repräsentiert die Tabelle `component`. Über eine `@OneToMany` -Beziehung kann eine einzelne Komponente mehrere `UsabilitySession`-Objekte referenzieren (z. B. Sitzungen, die in einem bestimmten Modul der Anwendung stattfinden).

`ErrorEvent` Erfasst Fehlermeldungen (Ausnahmen oder manuell gemeldete Fehler). Hier ist ebenfalls eine `@ManyToOne`-Beziehung zu `UsabilitySession` enthalten, um die Fehler eindeutig einer Session zuzuordnen. Dies erlaubt später, *Fehlerquote pro Session* oder *Fehler pro UI-Element* zu berechnen.

`NetworkRequest` Diese Entity zeichnet Netzwerkaufrufe auf (z. B. Start- und Endzeiten, URLs, HTTP-Status). Auch hier besteht ein Bezug zu `UsabilitySession`, sodass man *Performancewerte* pro Session analysieren kann.

`UsabilitySession`

```
@Entity
@Table(name = "usability_session")
public class UsabilitySession {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "session_id")
    private Long sessionId;

    @Column(name = "session_token")
    private String sessionToken;

    @Column(name = "start_time", nullable = false)
    private LocalDateTime startTime;

    @Column(name = "end_time")
    private LocalDateTime endTime;

    @ManyToOne
    @JoinColumn(name = "component_id", nullable = false)
    @JsonBackReference
    private Component component;

    @OneToMany(mappedBy = "usabilitySession", cascade = CascadeType.
       ALL)
```

```
24      @JsonBackReference
25      private List<ClickEvent> clickEvents;
26
27      @OneToMany(mappedBy = "usabilitySession", cascade = CascadeType.
            ALL)
28      @JsonBackReference
29      private List<ErrorEvent> errorEvents;
30
31      @OneToMany(mappedBy = "usabilitySession", cascade = CascadeType.
            ALL)
32      @JsonBackReference
33      private List<NetworkRequest> networkRequests;
34  }
```

Besonderheiten:

- `sessionId` fungiert als Primärschlüssel (Spalte `session_id`).
- `sessionToken` ermöglicht eine anonyme Identifikation der Session; es kann automatisch generiert oder extern vorgegeben werden.
- Beziehungen zu `ClickEvent`, `ErrorEvent` und `NetworkRequest` sind jeweils als `@OneToMany` markiert. Hierfür kommt `mappedBy` zum Einsatz, da die Fremdschlüssel in den jeweiligen *Kind-Entitäten* (Klicks, Fehler etc.) liegen.
- `@JsonBackReference` und `@JsonManagedReference` sorgen für eine korrekte Serialisierung bei *bidirektionalen* Beziehungen (etwa wenn das Frontend JSON-Daten abfragt).

4.8.4 Gründe für die gewählte Struktur

- **Klares Domänenmodell:** Jede Tabelle (z. B. `click_event`, `error_event`) ist durch eine Entity repräsentiert. Dies spiegelt den Geschäftskontext (Klicks, Fehler, Sessions usw.) unmittelbar wider.
- **Flexible Analysen:** Durch die „Eltern-Kind"-Beziehungen (`UsabilitySession` als Eltern, `ClickEvent`/`ErrorEvent`/`NetworkRequest` als Kinder) lassen sich Abfragen leicht realisieren (z. B. „Gib mir alle Klicks einer Session").
- **Datenschutz**: Die `sessionToken`-Spalte stellt sicher, dass keine *personenbezogenen Daten* gespeichert werden. Jede Session ist zwar eindeutig identifiziert, verweist aber nicht auf reale Personen.
- **Erweiterbarkeit**: Neue Felder (z. B. `browserInfo`, `deviceType`) lassen sich unkompliziert hinzufügen, ohne die gesamte Anwendungslogik massiv zu ändern.

4.9 Fazit

Die Nutzung von **JPA-Entities** bietet eine saubere und leicht wartbare Möglichkeit, um die relationalen Tabellen in Objektform darzustellen. Durch die in Abschnitt 3.9 erläuterte Datenbankstruktur werden die Klick-Ereignisse (*click_event*), Fehler (*error_event*), Netzwerkaufrufe (*network_request*) und Sitzungen (*usability_session*) so verknüpft, dass *komplexe Analysen* (z. B. Heatmaps, „most common paths“) ebenso möglich sind wie grundlegende CRUD-Operationen. So erfüllt das Modell sowohl *technische* Anforderungen (z. B. Performance, Wartbarkeit) als auch *fachliche* Ziele (z. B. Identifikation fehleranfälliger Elemente in *eDok*).

4.10 DAO-Schicht

Die *DAO*-Interfaces in diesem Projekt definieren, wie auf die Entitäten (z. B. `ClickEvent`, `ErrorEvent`) zugegriffen wird. Im Folgenden werden einige ausgewählte Interfaces erläutert; sie illustrieren die grundlegende Architektur, welche alle CRUD- und Analyseoperationen (z. B. Aggregationen) kapselt.

4.10.1 ClickEventDAO

```
1  package com.example.heatmapbackendmaven.Dao;
2
3  import com.example.heatmapbackendmaven.Entities.ClickEvent;
4  import com.example.heatmapbackendmaven.dto.ClickEventCountDto;
5  import com.example.heatmapbackendmaven.dto.respones.
       ClickEventResponseDto;
6  import com.example.heatmapbackendmaven.dto.respones.
       SequenceOccurrenceDto;
7  import java.util.List;
8
9  public interface ClickEventDAO {
10
11     String findMostClickedElement();
12
13     List<ClickEventCountDto> countClicksByElementIdForComponent(Long
           componentId);
14
15     List<ClickEventResponseDto>
           countClicksByElementIdForComponentForSequence(Long componentId)
           ;
16
17     void save(ClickEvent clickEvent);
18     void delete(ClickEvent clickEvent);
19     ClickEvent findById(Long id);
```

```
20      List<ClickEvent> findAll();
21      long countClicksByElementId(String elementId);
22
23      // returns sequences of clicks that appear most frequently
24      List<SequenceOccurrenceDto> findMostOccurringSequence(int
            sequenceLength);
25
26      // fetch all ClickEvents for a specific component
27      List<ClickEvent> findByComponent(Long componentId);
28
29      long count();
30  }
```

Dieses Interface definiert verschiedene **Lese- und Schreiboperationen** für `ClickEvent` -Objekte. Besonders wichtig sind hier die *Analysen*, die unter anderem:

- **`findMostClickedElement()`**: ermitteln, welches Element (z. B. Button) am häufigsten geklickt wurde.
- **`countClicksByElementIdForComponent()`**: die Klickanzahl nach `element_id` gruppieren, um Heatmap-Daten oder ähnliche Statistiken zu erzeugen.
- **`findMostOccurringSequence()`**: häufigste Klick-Sequenzen (Navigationspfade) identifizieren.

4.10.2 Weitere DAOs

Im Projekt existieren weitere DAO-Interfaces, die jeweils auf bestimmte Entitäten zugeschnitten sind:

- **`ComponentDao`**: Verwaltung von `Component`-Objekten (z. B. Module oder Subsysteme in *eDok*).
- **`ErrorEventDao`**: Zugriff auf `ErrorEvent`-Objekte; erlaubt u. a. *Zählungen* von Fehlern pro UI-Element und das Erstellen einer Zusammenfassung („Error Summary“).
- **`NetworkDao`**: Speichert und verwaltet Informationen zu Netzwerkaufrufen (`NetworkRequest`), etwa für Performance-Analysen.
- **`SessionDao`**: Zugriff auf `UsabilitySession`, d. h. Starten, Finden, Beenden oder Löschen einzelner Sitzungen.

Diese *DAO*-Schnittstellen werden in den Controllern und Services verwendet, um Abfragen auszuführen und Daten zu speichern. Die Implementierungen („Repository“-Klassen oder JPA-Implementierungen) sind meist stark an **Spring Data JPA** angelehnt und hier aus Übersichtsgründen weggelassen oder gekürzt.

4.11 Controller-Schicht

Die Controller empfangen HTTP-Anfragen vom Frontend und verwenden anschließend die DAO-Methoden, um Daten abzufragen oder zu speichern. Nachfolgend werden Auszüge einiger bedeutender Controller gezeigt.

4.11.1 AdminDashboardController

```
@RestController
@RequestMapping("/api/admin/dashboard")
public class AdminDashboardController {

    @Inject
    private SessionDao sessionDao;

    @Inject
    private ClickEventDAO clickEventDAO;

    @Inject
    private ErrorEventDao errorEventDao;

    @Inject
    private NetworkDao networkDao;

    @GetMapping("/stats")
    public DashboardStats getDashboardStats() {
        long totalSessions = sessionDao.count();
        long totalClickEvents = clickEventDAO.count();
        long totalErrorEvents = errorEventDao.count();
        long totalNetworkRequests = networkDao.count();

        return new DashboardStats(totalSessions, totalClickEvents,
                                  totalErrorEvents,
                                     totalNetworkRequests);
    }

    @GetMapping("/most-clicked-elements")
    public List<ElementCount> getMostClickedElements() {
        List<ClickEvent> all = clickEventDAO.findAll();
        return all.stream()
                  .collect(Collectors.groupingBy(ClickEvent::
                     getElementId,
                                  Collectors.counting()))
                  .entrySet().stream()
```

```
                    .map(e -> new ElementCount(e.getKey(), e.getValue())
                        )
                    .limit(10) // top 10
                    .collect(Collectors.toList());
    }
    // ...
}
```

Funktionalität:

- `/stats`: Ruft eine Zusammenfassung der wichtigsten Kennzahlen ab (*totalSessions*, *totalClickEvents* usw.), was im Admin-Dashboard visualisiert wird.
- `/most-clicked-elements`: Eine einfache *Top-10-Liste* der geklickten Elemente, ermittelt über `ClickEvent`.

4.11.2 ClickEventController

```
@RestController
@RequestMapping("/api/click-events")
public class ClickEventController {
    @Inject
    private ClickEventDAO clickEventDAO;
    @Inject
    private SessionDao sessionDao;

    @GetMapping
    public List<ClickEvent> getAllClickEvents() {
        return clickEventDAO.findAll();
    }

    @GetMapping("/component/heatmap/{componentId}")
    public Map<String, Long> getHeatmapForComponent(@PathVariable Long
        componentId) {
        List<ClickEventCountDto> results =
            clickEventDAO.countClicksByElementIdForComponent(
               componentId);

        Map<String, Long> heatmapData = new HashMap<>();
        for (ClickEventCountDto dto : results) {
            heatmapData.put(dto.getElementId(), dto.getTotalClicks());
        }
        return heatmapData;
    }

    @PostMapping
```

```
27      public void createClickEvent(@RequestBody CreateClickEventDto
            eventDto) {
28          UsabilitySession session = sessionDao.findById(eventDto.
                getSessionId())
29                  .orElseThrow(() -> new RuntimeException("Session not
                        found"));
30
31          ClickEvent event = new ClickEvent();
32          event.setElementId(eventDto.getElementId());
33          event.setSequenceNumber(eventDto.getSequenceNumber());
34          event.setUsabilitySession(session);
35          event.setTimestamp(LocalDateTime.now());
36
37          clickEventDAO.save(event);
38      }
39      // ...
40  }
```

Funktionalität:

- `getAllClickEvents()`: Liefert eine vollständige Liste aller Klick-Ereignisse zurück (primär für Debugging oder ausführliche Analysen).
- `getHeatmapForComponent()`: Aggregiert Klicks nach `element_id` und stellt Daten für eine Heatmap zur Verfügung.
- `createClickEvent()`: Nimmt ein neues Klick-Ereignis vom Frontend entgegen (*CreateClickEventDto*) und persistiert es in der Datenbank.

4.11.3 ClickSequenceController

Dieser Controller baut auf `ClickEventDAO` auf, um *Navigationspfade* (Click-Sequenzen) zu identifizieren. Beispielsweise kann man hier die „Top 10“ Pfade abrufen oder häufige Klick-Sequenzen für eine bestimmte Komponente ermitteln.

4.11.4 ErrorEventController

Ähnlich wie beim `ClickEvent` werden hier `ErrorEvent`-Objekte angelegt, abgefragt und statistisch ausgewertet. So kann man z. B. nachvollziehen, welches `element_id` am häufigsten zu Fehlern geführt hat.

4.11.5 Weitere Controller

- **ComponentController:** Verwaltung von `Component`-Objekten (z. B. Anlegen/Löschen oder Auflisten).

- **NetworkRequestController:** Ermöglicht das Anlegen von Netzwerk-Anfragen (`NetworkRequest`), um Performance-Daten oder Ladezeiten zu erfassen.
- **UsabilitySessionController:** Erzeugt oder beendet Sitzungen und verknüpft sie mit einer `Component` (Submodul in *eDok*). Damit wird die Anonymisierung gewahrt, da keine personenbeziehbaren Daten gespeichert werden.

4.12 Zusammenfassung der Backend-Architektur

Das Spring-Boot-Backend gliedert sich somit in **DAO-Interfaces**, die die Datenzugriffsebene definieren, und **Controller**, die die einzelnen Endpunkte für das Frontend bereitstellen. Durch diese Schichtung wird eine klare Trennung von *Geschäftslogik* und *Präsentationslogik* erreicht, was die Wartbarkeit und Erweiterbarkeit des Systems erhöht. Administrator*innen oder Entwickler*innen können zudem leicht neue Analyseendpunkte hinzufügen, indem sie entsprechende Methoden in den DAOs und Controllern ergänzen.

Im nächsten Abschnitt werden die **Frontend-Komponenten** in Angular näher erläutert, insbesondere die Mechanismen zur Erfassung von Klicks und Fehlern und die Visualisierung der Daten im Admin-Dashboard.

1. **Klickhäufigkeit pro UI-Element**
   Hierbei wird gezählt, wie oft ein bestimmtes `element_id` (z. B. ein Button oder Formularfeld) angeklickt wurde. Eine einfache SQL-Abfrage sähe so aus:

   Listing 1: Beispiel: Klickhäufigkeit pro Element

```sql
SELECT element_id, COUNT(*) AS total_clicks
FROM click_event
GROUP BY element_id
ORDER BY total_clicks DESC;
```

   Diese Auswertung hilft zu erkennen, welche UI-Bereiche am häufigsten genutzt werden. Elemente mit *besonders hohem* oder *unerwartet niedrigem* Klickaufkommen können im Anschluss gezielt überprüft und ggf. neu gestaltet werden.

2. **Fehlerquote (Fehler pro Klick / Zeit)**
   Um zu untersuchen, in welchem Verhältnis Klicks und Fehleraufkommen stehen, kann man beispielsweise die *Anzahl der Fehlerereignisse* (`error_event`) ins Verhältnis zu den Klick-Ereignissen setzen. Eine vereinfachte Kennzahl (Key Performance Indicator, KPI) wäre:

$$\text{Fehlerquote} = \frac{\sum(\text{Fehler})}{\sum(\text{Klicks})} \times 100\,\%$$

oder zeitbasiert:

$$\text{Fehlerquote pro Zeiteinheit} = \frac{\sum(\text{Fehler in } t)}{\sum(\text{Klicks in } t)}.$$

Eine zugehörige Abfrage könnte beispielsweise so aussehen, wenn man pro Zeiteinheit (z. B. pro Tag) die Fehler- und Klickanzahl bestimmen möchte:

Listing 2: Beispiel: Fehler- und Klickanzahl pro Tag

```
SELECT DATE(timestamp) AS day,
       (SELECT COUNT(*)
        FROM error_event e
        WHERE DATE(e.timestamp) = DATE(c.timestamp)) AS
           total_errors,
       COUNT(*) AS total_clicks
FROM click_event c
GROUP BY DATE(timestamp)
ORDER BY day;
```

Hierdurch lassen sich Zeiträume oder UI-Bereiche identifizieren, in denen *außergewöhnlich viele* Fehler auftreten, was ein klares Indiz für Optimierungsbedarf sein kann.

3. **Navigationspfade (Häufigste Klick-Sequenzen)**
   Anhand der `sequence_number` und der Zuordnung zu `usability_session` kann das System bestimmen, welchen Pfad Nutzende *typischerweise* innerhalb einer Sitzung durchlaufen (z. B. `Startseite -> Dokument anlegen -> Einstellungen -> Speichern`). Eine mögliche Herangehensweise ist das Gruppieren nach `session_id` und Sortieren nach der Klick-Sequenz:

Listing 3: Beispiel: Pfadrekonstruktion je Session

```
SELECT session_id, element_id, sequence_number
FROM click_event
ORDER BY session_id, sequence_number ASC;
```

Anschließend können sich wiederholende *Pfad-Signaturen* identifiziert werden (z. B. „Startseite->DokumentErstellen->...“), um sogenannte *most common paths* zu bestimmen. Dies liefert Aufschluss darüber, ob bestimmte Navigationsrouten gewünscht sind oder ob Nutzende häufig „Umwege“ gehen.

4. **Verweildauer pro Seite**
   Die Analyse der *Verweildauer* ist hilfreich, um *Aufwände* oder *eventuelle Probleme* zu erkennen (z. B. sehr lange Bearbeitungszeiten für ein Formular). Dazu werden z. B. `Page-Load`- und `Page-Unload`-Events (oder ein entsprechender `time_since_previous`-Wert) protokolliert. Die Differenz zwischen beiden Zeitpunkten ergibt:

$$\text{Verweildauer}(Seite_i) = \text{Endzeitpunkt} - \text{Startzeitpunkt}.$$

Um beispielsweise die durchschnittliche Verweildauer pro `usability_session` zu ermitteln, kann man die `time_since_previous` Werte summieren oder einen separaten `page_load` / `page_unload` Event-Ansatz verfolgen. Eine vereinfachte SQL-Abfrage (je nach Umsetzung) könnte so aussehen:

Listing 4: Beispiel: Durchschnittliche Verweildauer je Session

```sql
SELECT usability_session_id,
       AVG(time_since_previous) AS avg_time_spent
FROM click_event
GROUP BY usability_session_id
ORDER BY usability_session_id;
```

Seiten mit *auffallend langen* Verweilzeiten könnten zu komplex oder nicht intuitiv gestaltet sein, wohingegen *extrem kurze* Zeiten auf Desinteresse oder eine Fehlbedienung hinweisen können.

Ableitung von KPIs und weiterführenden Auswertungen.
In der Praxis werden diese Basisanalysen häufig zu komplexeren *Key Performance Indicators (KPIs)* kombiniert, etwa *Fehler pro 100 Klicks* oder *durchschnittliche Navigationszeit* zwischen zwei häufig aufgerufenen Elementen. Die in den Tabellen `click_event`, `error_event` und `usability_session` abgelegten Daten bieten somit eine *hohe Flexibilität*, um je nach Fragestellung passende Metriken zu berechnen und daraus konkrete Handlungsempfehlungen abzuleiten.

4.13 Frontend: Pfade und Endpunkte (Kurzüberblick)

Im *Angular-Frontend* werden alle benötigten **REST-Endpunkte** zentral in einem Objekt `API_PATHS` definiert. Damit liegt eine einzige *Quelle der Wahrheit* (Single Source of Truth) für die Verbindungen zum **Spring-Boot-Backend** vor. Ändern sich die Basispfade oder Ressourcen-Namen, genügt eine Anpassung in dieser Datei, statt den Code an mehreren Stellen zu überarbeiten.

```
const BASE_URL = 'http://localhost:8080/api';

export const API_PATHS = {
  SESSIONS: {
    BASE: `${BASE_URL}/sessions`,
    BY_ID: (id) => `${BASE_URL}/sessions/${id}`,
  },
  CLICK_EVENTS: {
    BASE: `${BASE_URL}/click-events`,
```

```
    BY_SESSION: (sessionId) => `${BASE_URL}/click-events/session/${
        sessionId}`,
  },
  // ... weitere Endpunkte
};
```

Beispielhafter Aufruf: In einem Angular-Service kann so auf den Endpunkt zugegriffen werden:

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import API_PATHS from '../path/to/api-paths';

@Injectable({ providedIn: 'root' })
export class ClickEventService {
  constructor(private http: HttpClient) {}

  getClickEventsBySession(sessionId: number) {
    return this.http.get(API_PATHS.CLICK_EVENTS.BY_SESSION(sessionId))
        ;
  }
}
```

Auf diese Weise bleibt die *Pfad-Definition* klar gekapselt und kann flexibel angepasst oder erweitert werden, ohne dass der restliche Code verändert werden muss.

4.14 Beispielhafter Angular-Service zur Heatmap-Visualisierung (Kurzfassung)

Nachfolgend ein *Auszug* aus einem **Angular-Service**, der Daten (z. B. Klicksequenzen) vom Backend empfängt und mithilfe des Angular-Renderers *visuelle Labels* für eine Heatmap erzeugt. Diese Kurzfassung zeigt nur die wichtigsten Methoden, um das Vorgehen zu veranschaulichen.

```
@Injectable({ providedIn: 'root' })
export class HeatmapVisualizerService implements OnDestroy {
  private subscription = new Subscription();
  private renderer: Renderer2;

  constructor(
    private http: HttpClient,
    rendererFactory: RendererFactory2,
    private heatmapCaptureService: HeatmapCaptureService,
  ) {
    this.renderer = rendererFactory.createRenderer(null, null);
  }
```

```
13
14    /**
15     * Lädt eine Klick-Sequenz von der Server-API.
16     */
17    fetchClickSequence(componentId: number) {
18      return this.http.get<ClickEventResponseDto[]>(
19        API_PATHS.CLICK_SEQUENCES.BY_COMPONENT(componentId)
20      );
21    }
22
23    /**
24     * Ruft Heatmap-Daten ab und erzeugt Label-Elemente (z. B. für
          Klickanzahl).
25     */
26    fetchHeatmapDataAndCreateLabels(componentId: number, doc: Document):
          void {
27      // (1) Vorherige Subscriptions beenden
28      this.subscription.unsubscribe();
29      // (2) Per Service die entsprechenden Daten abrufen
30      const sub = this.heatmapCaptureService
31        .getHeatmapForComponentWithSequence(componentId)
32        .subscribe(data => {
33          // (3) Daten verarbeiten und Labels im DOM platzieren
34          this.processHeatmapData(data, doc);
35        });
36      this.subscription.add(sub);
37    }
38
39    /**
40     * Verarbeitet die abgerufenen Daten (z. B. elementId,
          sequenceNumber)
41     * und erzeugt CSS-Labels am Ziel-Element.
42     */
43    private processHeatmapData(data: ClickEventResponseDto[], doc:
          Document): void {
44      data.forEach(item => {
45        // Ziel-Element über item.elementId finden
46        const targetElement = doc.querySelector(`#${item.elementId}`);
47        if (targetElement) {
48          // Label anbringen, z. B. mit Klickanzahl oder Sequenznummer
49          this.createNumberLabel(targetElement, item.clickCount, 'click-
              count');
50        }
51      });
52    }
```

```
53
54    /**
55     * Erstellt ein kleines Label-Element mit CSS-Stil
56     */
57    private createNumberLabel(target: HTMLElement, number: number,
          cssClass: string): void {
58      // ...
59    }
60
61    ngOnDestroy(): void {
62      this.subscription.unsubscribe();
63    }
64  }
```

Kommentar:

- *fetchClickSequence(...) und fetchHeatmapDataAndCreateLabels(...)* zeigen den Kern:
  1. **Abruf** der gewünschten Daten via `HttpClient` oder einem weiteren Service.
  2. **Erzeugung** von CSS-basierten Overlay-Labels, die im DOM platziert werden, um die Klick- oder Sequenz-Informationen zu visualisieren.
- *processHeatmapData(...)* illustriert, wie jedes `elementId` im Dokument gesucht und mit **Labels** versehen wird.
- Der volle Quellcode (Methodenimplementierung, Error-Handling, CSS-Stile etc.) kann je nach Projektumfang wesentlich detaillierter ausfallen.

Diese Kurzfassung verdeutlicht das Prinzip: Die *Heatmap-Daten* (z. B. Klickhäufigkeiten) werden serverseitig gesammelt und per REST abgerufen; anschließend *ergänzt* der Service das Angular-DOM um visuell hervorstechende Komponenten (Labels), sodass Administratoren oder UX-Verantwortliche die **Nutzerinteraktionen** unmittelbar „auf dem Screen“ erkennen können.

4.15 Kurz gefasster Code-Ausschnitt (HeatmapCaptureService)

Untenstehend ein *reduzierter* Auszug, der nur die **zentralen** Methoden und Felder aus dem originalen `HeatmapCaptureService` zeigt, ohne semantische Änderungen vorzunehmen. Dabei sind alle weiterführenden Details (etwa Error-Handling, Logging-Ausgaben) weggelassen, um den *Kernablauf* übersichtlich darzustellen.

```
1  @Injectable({ providedIn: 'root' })
2  export class HeatmapCaptureService {
3    private sessionIdSubject = new BehaviorSubject<number | null>(null);
4    public sessionId$ = this.sessionIdSubject.asObservable();
5    private clickSubscription?: Subscription;
```

```
6   private sequenceNumber = 0;
7
8   constructor(
9     private http: HttpClient,
10    private sessionTokenService: SessionTokenService,
11    private requestLogService: RequestLogService
12  ) {}
13
14  // 1) Startet eine neue Usability-Session
15  startNewSession(componentId: number): Observable<UsabilitySession> {
16    const payload = { componentId };
17    return this.http.post<UsabilitySession>(API_PATHS.SESSIONS.BASE,
        payload);
18  }
19
20  // 2) Setzt das 'Click'-Abonnement auf und speichert die sessionId
21  startRecording(componentId: number): Observable<number> {
22    return new Observable<number>(observer => {
23      this.startNewSession(componentId).subscribe({
24        next: session => {
25          this.sessionIdSubject.next(session.sessionId);
26          this.sessionTokenService.setToken(session.sessionToken);
27          this.clickSubscription = fromEvent<MouseEvent>(document, '
              click')
28            .subscribe(event => this.recordClickEvent(event,
                componentId));
29          observer.next(session.sessionId);
30          observer.complete();
31        },
32        error: err => observer.error(err),
33      });
34    });
35  }
36
37  // 3) Stoppt die Aufzeichnung von Klick-Ereignissen
38  stopRecording(): void {
39    this.clickSubscription?.unsubscribe();
40    this.clickSubscription = undefined;
41  }
42
43  // 4) Nimmt jeden Klick und sendet ihn an den RequestLogService
44  private recordClickEvent(event: MouseEvent, componentId: number):
      void {
45    const sessionId = this.sessionIdSubject.value;
46    if (!sessionId) return;
```

```
47
48      const elementId = (event.target as HTMLElement).id || '';
49      if (!elementId) return;
50      this.sequenceNumber++;
51
52      const clickEventLog = {
53        sessionId,
54        requestUrl: API_PATHS.CLICK_EVENTS.BASE,
55        requestMethod: 'POST',
56        requestStartTime: new Date(),
57        requestEndTime: new Date(),
58        responsePayload: JSON.stringify({ elementId, componentId,
              sequenceNumber: this.sequenceNumber, sessionId })
59      };
60
61      this.requestLogService.addLog(clickEventLog).subscribe();
62    }
63  }
```

Erläuterung (Kurzform):

- *startNewSession(...)*: Erzeugt eine neue `UsabilitySession` im Backend (z. B. via `POST`).
- *startRecording(...)*: Aktiviert ein „Klick-Abo“ auf dem `document` und speichert die `sessionId`.
- *recordClickEvent(...)*: Baut die Daten des Klicks (`elementId`, `sequenceNumber`) zusammen und übergibt sie an `requestLogService`.

Durch diesen minimalen Ausschnitt wird deutlich, *wie* Klick-Ereignisse aufgezeichnet und mit einer laufenden Session verknüpft werden, ohne alle Logger-Details oder Error-Handling-Zweige anzuzeigen.

4.16 Beispielhafter `RequestLogService` (Kurzfassung)

Dieser **Angular-Service** protokolliert und versendet Netzwerk-Logs an das Backend. Jeder Eintrag enthält Informationen wie *Request-URL*, *Statuscode*, *Zeitstempel* und eine *Session-ID*. Im folgenden, reduzierten Codeausschnitt sind nur die zentralen Methoden und Felder dargestellt:

```
1  @Injectable({ providedIn: 'root' })
2  export class RequestLogService {
3    private logs: NetworkLogEntry[] = [];
4
5    constructor(private http: HttpClient) {}
```

```
6
7     /**
8      * Fügt einen Log-Eintrag hinzu und sendet ihn direkt zum Backend.
9      */
10    addLog(logData: NetworkLogEntry): Observable<any> {
11      return this.http.post(API_PATHS.NETWORK_REQUESTS.BASE, [logData]);
12    }
13
14    /**
15     * Beispiel eines optionalen Puffers:
16     * Sendet alle gespeicherten Log-Einträge (flush) an das Backend.
17     */
18    flush(): void {
19      if (this.logs.length === 0) return;
20      this.http.post(API_PATHS.NETWORK_REQUESTS.BASE, this.logs).
            subscribe({
21        next: () => {
22          console.log('Flush erfolgreich, Einträge gesendet.');
23          this.logs = [];
24        },
25        error: err => {
26          console.error('Fehler beim Flush:', err);
27        }
28      });
29    }
30  }
```

Kurze Erläuterung:

- *addLog(...)*: Nimmt einen einzelnen Log-Eintrag entgegen und sendet ihn umgehend an den entsprechenden *API_PATH* (siehe `NETWORK_REQUESTS.BASE`). Bei Bedarf könnten mehrere Einträge auch gepuffert werden (Batching).
- *flush(...)*: Zeigt ein mögliches Batch-Szenario, in dem mehrere Log-Einträge *gesammelt* und anschließend in einem Rutsch an das Backend gesendet werden.

Zeit- oder Eventbasierter Flush Neben dem manuellen Aufruf kann `flush()` auch in *Zeitintervallen* (z. B. per `setInterval`) oder automatisch im `ngOnDestroy` (Lebenszyklus-Hook) aufgerufen werden, um die Daten *sauber* beim Entladen der Komponente zu senden. Eine solche *Batch-Strategie* kann die Performance verbessern, indem unnötige Einzelaufrufe reduziert werden. Das Konzept lässt sich bei Bedarf um *Retry-Logiken*, *Caching* oder *umfangreiche Error-Handling-Routinen* erweitern, um eine noch robustere und effizientere Protokollierung zu erreichen.

4.17 HTTP-Interceptor zum Timing und Loggen von Requests

Um die **Performance** jedes HTTP-Aufrufs zu messen und automatisch an das Backend zu protokollieren, wird in diesem Beispiel ein *TimingInterceptor* eingesetzt. Angular erlaubt die Registrierung beliebig vieler Interceptor-Instanzen über die `HTTP_INTERCEPTORS` -Provider.

4.17.1 Registrierung des Interceptors

```
// httpInterceptorProviders.ts
import { HTTP_INTERCEPTORS } from '@angular/common/http';
import { TimingInterceptor } from './Services/TimingInterceptor';

export const httpInterceptorProviders = [
  { provide: HTTP_INTERCEPTORS, useClass: TimingInterceptor, multi:
     true },
];
```

Die Konfiguration `multi: true` sorgt dafür, dass dieser Interceptor *zusätzlich* zu eventuell vorhandenen Interceptors registriert wird (statt diese zu ersetzen).

4.17.2 Funktionsweise (Kurzfassung)

Der `TimingInterceptor` setzt zwei wesentliche Schritte um:

1. **Messung der Request-Dauer:** Vor dem Aufruf wird `startTime = performance.now()` erfasst. Nach einer erfolgreichen oder fehlerhaften Antwort wird die Zeitdifferenz `endTime - startTime` berechnet.

2. **Erfassung wichtiger Request-Daten:** Mithilfe von `logRequestTiming(...)` werden *Request-URL*, *Methode*, *Statuscode* und *Fehlernachricht* (falls vorhanden) in einem Logobjekt festgehalten. Dieses Objekt wird *asynchron* an das Backend gesendet, um *Endlosschleifen* zu verhindern (daher die Prüfung „wenn `req.url.includes(API_PATHS.NETWORK_REQUESTS.BASE)` dann skippe Logging“).

4.17.3 Beispielcode (Auszüge)

```
@Injectable()
export class TimingInterceptor implements HttpInterceptor {
  private sessionId: number;

  constructor(
    private requestLogService: RequestLogService,
    private heatmapService: HeatmapCaptureService
  ) {
```

```
9       // Auslesen der Session-ID aus dem HeatmapService (z.B. via
            sessionId$)
10      this.heatmapService.sessionId$.subscribe({
11        next: id => this.sessionId = id
12      });
13    }
14
15    intercept(req: HttpRequest<any>, next: HttpHandler): Observable<
          HttpEvent<any>> {
16      // (1) Bestimmte Endpunkte vom Logging ausnehmen
17      if (req.url.includes(API_PATHS.NETWORK_REQUESTS.BASE)) {
18        return next.handle(req);
19      }
20
21      const startTime = performance.now();
22      return next.handle(req).pipe(
23        tap(event => {
24          if (event instanceof HttpResponse) {
25            const duration = performance.now() - startTime;
26            this.logRequestTiming(req.url, req.method, duration, event.
                  status, 'Success', null);
27          }
28        }),
29        catchError(error => {
30          const duration = performance.now() - startTime;
31          this.logRequestTiming(req.url, req.method, duration, error.
                 status, 'Failed', error.message);
32          return throwError(() => error);
33        })
34      );
35    }
36
37    private logRequestTiming(
38      url: string,
39      method: string,
40      duration: number,
41      status: number,
42      statusText: string,
43      errorMessage: string | null
44    ): void {
45      // Minimales Log-Objekt
46      const logData = {
47        sessionId: this.sessionId,
48        requestUrl: url,
49        requestMethod: method,
```

```
50        durationMs: Math.round(duration),
51        statusCode: status,
52        requestStartTime: new Date(performance.timeOrigin + performance.
              now() - duration),
53        requestEndTime: new Date(),
54        responsePayload: errorMessage || statusText
55      };
56
57      // Übergabe an den RequestLogService (sammelt/sendet Logs an das
            Backend)
58      this.requestLogService.addLog(logData);
59    }
60  }
```

4.17.4 Leistungsaspekte und Erweiterungen

- **Vermeidung von Endlosschleifen:** Indem `req.url.includes(API_PATHS.NETWORK_REQUESTS.BASE)` geprüft wird, *werden* die eigenen Log-Anfragen *nicht* erneut geloggt. Das verhindert unendliche Schleifen („Interceptor ruft sich selbst auf“).
- **Asynchrones Logging:** Das *Log-Objekt* wird *nicht* direkt blockierend in die Datenbank geschrieben, sondern über `this.requestLogService` ggf. gepuffert oder sofort per `POST` gesendet. So wird die Latenz des eigentlichen Nutzer-Requests minimal erhöht.
- **Mögliche Performance-Verbesserungen:**
  1. *Batch- bzw. Timerkonzept*: Statt jeden Request einzeln ins Backend zu schreiben, könnten Logeinträge zeitverzögert gesammelt werden (`flush()` nach x Sekunden oder im `ngOnDestroy`).
  2. *Filterung*: Man könnte Requests, die sehr häufig auftreten (z. B. Ressourcen wie Icons), vollständig vom Timing ausschließen.
  3. *Retry-Mechanismen*: Bei fehlerhaften Log-Anfragen könnte man (optional) eine kleine Queue implementieren, um die Daten später erneut zu senden.
- **Fehleranalyse**: Neben der gemessenen Dauer werden ggf. *Fehlertexte* (etwa aus `HttpErrorResponse`) im Log vermerkt, was bei Debugging oder Statistikzwecken wertvoll ist.

Zusammengefasst ermöglicht der `TimingInterceptor` eine **zentralisierte Messung** und Protokollierung sämtlicher HTTP-Aufrufe, ohne die eigentliche Applikationslogik anpassen zu müssen. Dies sorgt für *Transparenz* über die Performance und erleichtert spätere Analysen (z. B. bei Latenzproblemen oder häufigen Fehlschlägen).

4.18 Globaler Error-Handler und manuelles Fehlermanagement

In diesem Ansatz werden sowohl *unbehandelte* Ausnahmen (über einen **GlobalErrorHandler**) als auch *manuell* gemeldete Fehler (via **ErrorTestComponent**) an den Server gesendet. Damit entsteht eine zentrale Möglichkeit, Fehler zu erfassen, ohne dass jede Komponente einzeln ein eigenes Logikgerüst aufbauen muss.

4.18.1 GlobalErrorHandler (Auszug)

```
@Injectable()
export class GlobalErrorHandler implements ErrorHandler {
  constructor(
    private injector: Injector,
    private sessionTokenService: SessionTokenService
  ) {}

  handleError(error: any): void {
    const errorEventService = this.injector.get(ErrorEventService);
    const token = this.sessionTokenService.getToken();

    const errorEvent: ErrorEventDto = {
      componentId: 500,  // Beispiel: könnte dynamisch gesetzt werden
      elementId: 'unknown',
      message: error.message || error.toString(),
      stackTrace: error.stack || null,
      timestamp: new Date().toISOString(),
      usabilitySessionTocken: token
    };

    errorEventService.logError(errorEvent).subscribe();
  }
}
```

Funktionsweise (Kurzfassung):

- *ErrorHandler*-Schnittstelle von Angular: Wird automatisch aufgerufen, wenn eine Exception geworfen wird, die *nicht* innerhalb eines eigenen `try/catch` oder `Observable` abgefangen wurde.
- *ErrorEventService*: Sendet die Daten (Fehlermeldung, Stacktrace, `sessionToken`) an den *Back-End-Endpunkt* `/error-events/log`.
- *componentId=500, elementId='unknown'*: In diesem Beispiel statisch gesetzt; in einer realen Lösung könnte man dynamisch „Ort“ oder „UI-Element“ ermitteln.

4.18.2 Manuelles Fehlermanagement in der `ErrorTestComponent`

```
// Ausschnitt: ErrorTestComponent
export class ErrorTestComponent {
  manualError: ErrorEventDto = {
    elementId: '',
    componentId: 560, // beispielhafter Wert
    message: '',
    stackTrace: '',
    timestamp: new Date().toISOString(),
    usabilitySessionTocken: null
  };

  constructor(
    private errorEventService: ErrorEventService,
    private sessionTokenService: SessionTokenService
  ) {}

  // Methode, um einen selbst definierten Fehler zu erfassen
  logManualError(): void {
    this.manualError.timestamp = new Date().toISOString();
    this.manualError.usabilitySessionTocken = this.sessionTokenService
        .getToken();

    this.errorEventService.logError(this.manualError).subscribe({
      next: () => {
        console.log('Manueller Fehler erfolgreich gemeldet');
      },
      error: err => {
        console.error('Fehler beim Melden:', err);
      }
    });
  }

  // Beispiel: absichtlich Fehler werfen, um GlobalErrorHandler zu
     testen
  triggerUncaughtError(): void {
    throw new Error('Test uncaught error');
  }
}
```

Manueller Workflow (Kurzfassung):

1. Nutzer*in gibt *Error-Daten* ins Formular (`elementId`, `message`) ein.
2. `logManualError()` sendet den erfassten Fehler an *ErrorEventService*, der wiederum per `POST` zum Backend schreibt.

3. Rückmeldung (Erfolg/Fehler) wird im UI angezeigt.

4.18.3 ErrorEventService (Auszug)

```
@Injectable({ providedIn: 'root' })
export class ErrorEventService {
  constructor(private http: HttpClient) {}

  logError(errorEvent: ErrorEventDto): Observable<void> {
    return this.http.post<void>(API_PATHS.ERROR_EVENTS.LOG, errorEvent
        );
  }
}
```

Anmerkungen zur Performance:

- **Fehlermeldungen sind in der Regel seltener als Klick-Ereignisse**: Die sofortige Übertragung an den Server verursacht deshalb wenig Overhead.
- **Batching** könnte optional implementiert werden, wenn sehr viele Fehler gleichzeitig auftreten. Ähnlich wie beim `RequestLogService` kann man eine *Warteschlange* anlegen und nur alle x Sekunden *flushen*.
- **Vermeiden von Endlosschleifen**: Der **GlobalErrorHandler** protokolliert den Fehler nur *einmal*, man sollte jedoch darauf achten, *keine* Requests erneut in den selben Codepfad zu leiten (z. B. Ausnahmen innerhalb `logError(...)`).

Somit deckt das System sowohl *unbehandelte* Fehler (etwa Laufzeitfehler im UI) als auch *bewusst gemeldete* Ausnahmesituationen ab. Dies verschafft dem Entwicklungsteam einen umfassenden Überblick, welche Probleme in der Anwendung auftreten, in welcher `Komponente` sie geschehen und wie häufig sie vorkommen.

4.19 Bezug zu den Zielen (Kapitel 1)

In **Kapitel 1** wurde als Hauptziel dieser Arbeit definiert, die *User Experience (UX)* in der Anwendung *eDok* systematisch zu erfassen, zu analysieren und durch konkrete Maßnahmen zu verbessern. Dabei sollten folgende Aspekte besonders berücksichtigt werden:

- **Kontinuierliche Datenerfassung ohne Mehraufwand für Nutzende:** Das automatisierte *Event Logging* (z. B. in `click_event`) ermöglicht eine lückenlose Erfassung der Interaktionen, ohne dass Benutzer zusätzliche Aktionen ausführen müssen.

- **Anonymisierung und Datenschutz:** Durch die Abstraktion über die Tabelle `usability_sessio` bleibt die Identität der Nutzenden geschützt, da statt personenbezogener Daten lediglich pseudonymisierte Tokens gespeichert werden.
- **Übersichtliche Visualisierung und Trendanalysen:** Die in `click_event`, `network_request` und `error_event` gespeicherten Daten lassen sich zu aussagekräftigen *Heatmaps* und *Fehlerstatistiken* zusammenführen, sodass Administratoren im Dashboard einen klaren Überblick gewinnen.
- **Gezielte Optimierung von UI- und Nutzungsprozessen:** Anhand der gesammelten `sequence_number`-Informationen kann z. B. die Reihenfolge von Klicks analysiert werden, um häufige Pfade (oder Stolpersteine) zu identifizieren und das Benutzererlebnis weiterzuentwickeln.

Unterstützung durch die Tabellenstruktur. Die Datenbanktabellen (`usability_session`, `click_event`, `error_event` etc.) implementieren die oben genannten Ziele wie folgt:

1. `click_event` erfasst alle relevanten Interaktionen (Element-ID, Zeitstempel, Reihenfolge), die für *Heatmaps* und Klickstatistiken benötigt werden.
2. `usability_session` gewährleistet eine saubere *Session-Verfolgung*, ohne Rückschlüsse auf konkrete Personen zuzulassen. So kann z. B. die Verweildauer oder die Anzahl der Fehler pro Session analysiert werden.
3. `error_event` und `network_request` ermöglichen eine *tiefergehende Performance- und Fehleranalyse*, um systematische Probleme aufzudecken.

Erreichte Ziele. Die Arbeit konnte die gesteckten Ziele im Wesentlichen erfüllen:

- Die Datenerfassung und -speicherung funktioniert *kontinuierlich und anonymisiert*, sodass *eDok* unter realen Nutzungsbedingungen beobachtet werden kann, ohne datenschutzrechtliche Bedenken.
- Das *Dashboard* stellt die gesammelten Daten visuell dar und bietet Administrator*innen eine solide Basis, um entscheidungsrelevante Informationen (z. B. häufige Klickpfade, Fehlerhotspots) auf einen Blick zu erkennen.
- *Iterative Verbesserungen* lassen sich auf Grundlage der gewonnenen Erkenntnisse realisieren, indem problematische UI-Bereiche oder ineffiziente Interaktionsmuster konkret adressiert werden.

Nicht vollständig abgedeckte Ziele. Trotz der erfolgreichen Implementierung bleiben bestimmte Bereiche *unberücksichtigt*:

- **Keine subjektiven UX-Daten:** Da keinerlei Umfragen, Interviews oder Beobachtungen einbezogen wurden, fehlen tiefergehende Einblicke in Emotionen, Motivation und Zufriedenheit der Nutzenden.

Ungeachtet dieser Einschränkungen bildet das entwickelte Modell eine solide Grundlage, um die *Usability* von *eDok* kontinuierlich zu überwachen und gezielt zu verbessern. Zukünftige Arbeiten könnten um qualitative Methoden oder weiterführende Analysetechniken ergänzt werden, um ein noch umfassenderes Bild der *User Experience* zu gewinnen.

5 Diskussion

In diesem Abschnitt werden zentrale Aspekte des vorgestellten Modells und der Implementierung kritisch betrachtet. Dabei stehen sowohl die Stärken der gewählten Lösung als auch ihre Grenzen und mögliche Erweiterungspotenziale im Fokus.

5.1 Stärken der Umsetzung

- **Umfassende Datenerfassung**: Das System erfasst ein breites Spektrum an Informationen – von *Klick-Interaktionen* (Heatmap- und Sequenzdaten) über *Netzwerkperformance* (Timing-Interceptor) bis hin zu *Fehlerereignissen* (Globaler Error-Handler). Dadurch entsteht ein ganzheitliches Bild der aktuellen Nutzungsweise und Stabilität der Anwendung.

- **Lose Kopplung von Frontend und Backend**: Durch die klar definierten REST-Endpunkte (vgl. Abschnitt 4.13) und Services wie den `RequestLogService` können einzelne Bausteine (z. B. Session-Management, Klick-Logging) unabhängig voneinander weiterentwickelt oder ausgetauscht werden.

- **Datenschutzkonformer Ansatz**: Nutzeraktionen werden zwar aufgezeichnet, aber nur in *anonymisierter Form* (z. B. Session-Token anstelle von echten User-IDs). Damit werden Persönlichkeitsrechte der Anwender gewahrt und Datenschutzauflagen eingehalten.

- **Erweiterbarkeit**: Die gewählte Architektur erlaubt es, zusätzliche Metriken (z. B. Mausbewegungen, Scroll-Verhalten) oder Data-Mining-Techniken (z. B. Assoziationsanalyse) relativ leicht zu integrieren. Ebenso sind Batching-Mechanismen für das Logging bereits angedacht (vgl. `flush()`-Methoden).

5.2 Einschränkungen und mögliche Verbesserungen

- **Qualitative UX-Daten**: Trotz umfangreicher *quantitativer* Erfassungsmethoden (z. B. Klickhäufigkeiten, Fehlerquoten) werden *emotionale* Faktoren (Motivation, Zufrie-

denheit) oder die *Hintergründe* für bestimmtes Nutzerverhalten nicht erfasst. Qualitative Methoden (Interviews, Usability-Tests mit Beobachtung) könnten das Gesamtergebnis ergänzen.

- **Skalierung bei hohem Datenaufkommen**: Sollte die Anwendung in großem Umfang genutzt werden, könnten die *Einzelrequests* für das Logging (z. B. Klicks und Timings) die Systemlast erhöhen. Ein *systematisches Batching* oder zeitgesteuertes `flush()` würde helfen, Performance-Engpässe zu mindern.
- **Eingeschränkte Mobile-Perspektive**: Das aktuelle Konzept fokussiert sich auf Desktop-Interaktionen („Clicks“ statt „Touches“). Bei steigender Bedeutung mobiler Geräte wäre eine *Touch-Event-Erfassung* oder ein responsives Metriksystem sinnvoll.
- **Abhängigkeiten und Wartung**: Die Lösung setzt auf diverse Services (z. B. `HeatmapCaptureSer` `GlobalErrorHandler`, `TimingInterceptor`), die sinnvoll koordiniert werden müssen. Eine einheitliche Dokumentation und klare Verantwortlichkeiten verhindern hier potenziellen *Wartungs-Overhead*.

5.3 Ausblick und Weiterentwicklung

- **Erweiterte Datenanalysen**: Auf Grundlage der vorliegenden Daten könnte ein *Data-Mining-Prozess* (z. B. Clustering, Anomalie-Erkennung) echte Mehrwerte liefern. So ließen sich verborgene Muster in der Navigation oder Fehlerhotspots identifizieren.
- **Automatisierte Handlungsempfehlungen**: Ein nächster Schritt wäre, die erfassten Metriken (z. B. *Fehler pro 100 Klicks*) automatisiert zu bewerten und *Alarme* oder *Ticket-Einträge* zu generieren, sobald Schwellwerte überschritten werden.
- **Umfassendes Session- bzw. Nutzerflow-Reporting**: Die aktuell implementierte Logging-Architektur kann relativ einfach genutzt werden, um *komplexe Reports* zu generieren (z. B. „Wo brechen Nutzer am häufigsten ab?“). Das Admin-Dashboard ließe sich hierzu um *Trenddiagramme* oder *Zeitreihen* erweitern.
- **Plattformunabhängigkeit**: Eine portierte Variante der Tracking-Services (etwa für Android/iOS) würde eine *einheitliche* UX-Analyse über verschiedene Endgeräte ermöglichen. Das erfordert allerdings den Umbau von *Klick-* auf *Touch*-Events und Anpassungen beim Layout.

5.4 Fazit zur Diskussion

Die hier vorgestellte Lösung deckt den **Großteil** der *quantitativen* UX- und Usability-Metriken ab, die für das *eDok*-System relevant sind (Heatmaps, Fehlermeldungen, Per-

formance). Gleichzeitig zeigt sich, dass die *tieferen* UX-Aspekte (z. B. subjektive Zufriedenheit, emotionales Empfinden) weiterhin *ausgeklammert* sind. Zudem erfordert die konstante Datenerfassung im Live-Betrieb stets ein wachsames Auge auf *Performance* und *Datenschutz*.

Dennoch bietet das Modell eine solide Grundlage, um die **Nutzerinteraktion** in *eDok* zu verstehen, problematische Stellen frühzeitig zu erkennen und systematisch Anpassungen abzuleiten. Dies kann zu einer verbesserten Usability und letztlich zu einem reibungsloseren Arbeitsalltag für die Nutzer*innen führen.