11.10.2023 bis 10.12.2023

Prozessorientierter Projektbericht

Projekt zum Thema :

Desktop-Anwendung zur Terminvergabe in einer Arztpraxis

Vorgelegt von :

Rafik Fezai

Jagdweg 39

53115 Bonn

Email : rafikfezai16@gmail.com

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API .....*Application Programming Interface*
DML.....*Data Manipulation Language*
GUI .....*graphical user interface*
JDK.....*Java Development Kit*
MVC.....*Model View Controller*
OJDBC.....Oracle JDBC (Java Database Connectivity)

1 Einleitung

Das Projekt basiert auf selbst festgelegten Anforderungen, und es ist wichtig zu betonen, dass das entstandene Produkt nicht als vollständige Anwendung konzipiert wurde. Es erfolgte kein Deployment, und die Anwendung wurde lediglich nur mittels eines Black-Box-Tests überprüft.

Der Bericht dokumentiert die Abläufe und Phasen dieses Projekts, beginnend von der Analyse bis zur Zielerreichung. Der gesamte Zeitrahmen für die Planung und Durchführung dieses Projekts betrug etwa 2 Monate, mit einem Startdatum am 11. Oktober 2023 und einem Abschlussdatum am 10. Dezember 2023.

Die entwickelte Anwendung dient der Terminverwaltung innerhalb einer fiktiven Arztpraxis.

2 Anforderungen

- Für eine Arztpraxis in Bonn soll eine plattformunabhängige Desktop-Anwendung in Java-Swing erstellt werden. Die Daten sollen in einer Oracle-Datenbank gespeichert werden. Diese Anwendung richtet sich an die Praxis-Mitarbeiter und soll ihnen helfen, digital Patiententermine zu verwalten.
- Die Öffnungszeiten der Praxis sind von 8 Uhr bis 17 Uhr. Zwischen 12 Uhr und 14 Uhr ist eine Pause für alle Mitarbeiter vorgesehen, in der keine Termine vergeben werden dürfen.
- Die Termine sollen in 15-minütigen Intervallen vergeben werden. Sie müssen an Werktagen (Montag bis Freitag) stattfinden und dürfen nicht an gesetzlichen Feiertagen in Nordrhein-Westfalen vereinbart werden.
- Es kommt häufig vor, dass Patienten ihre Termine absagen. Daher ist eine Funktion zur Terminstornierung erforderlich.
- Ein Patient darf einen Termin nur mit seinem Betreuer (Arzt) vereinbaren, da dieser die gesundheitliche Lage des Patienten kennt.
- In der Praxis werden oft neue Patienten aufgenommen. Aufgrund der hohen Anzahl an Patienten werden auch häufig neue Ärzte eingestellt, die ihren eigenen festen Arbeitsplatz (Raum) in der Praxis erhalten.
- Um die Übersicht über die Termine zu behalten, muss die Anwendung eine Ansicht für alle Termine bieten.
- Zur Sicherung der Daten muss sich der Anwender zuerst anmelden, um das Programm nutzen zu können.

3 Analysephase

3.1 Anwendungsfalldiagramm erstellen

Basierend auf den oben genannten Anforderungen handelt es sich um eine Single-User-Software. Der Hauptbenutzer ist der Praxis-Mitarbeiter, der als primärer Akteur agiert. Der Patient interagiert nicht direkt mit dem System, ist jedoch als externer Akteur involviert. Das folgende Anwendungsfalldiagramm bietet eine anschauliche und globale Visualisierung des Systems und seine unterschiedlichen Funktionalitäten.

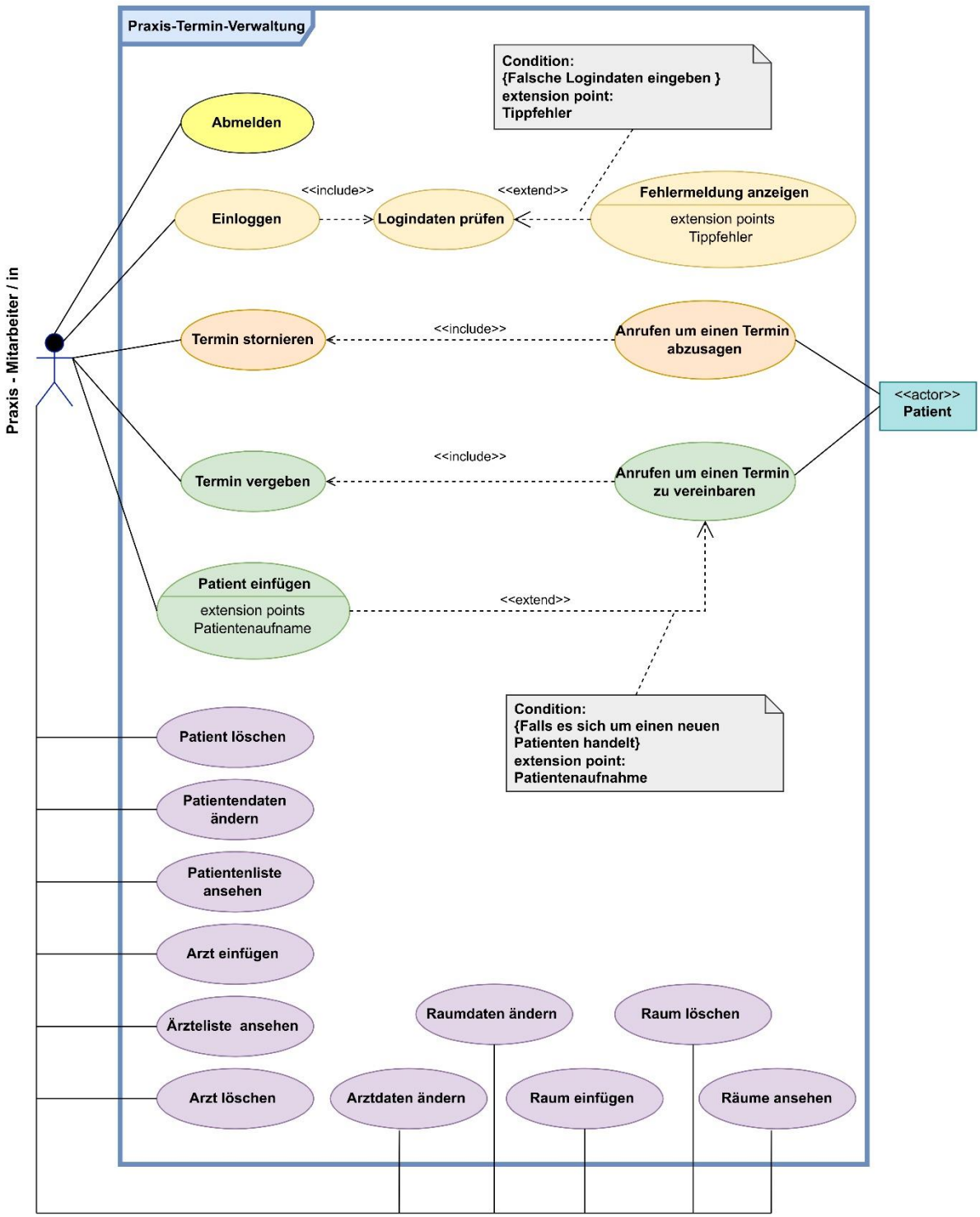

Abbildung 1: Anwendungsfalldiagramm (use case diagram)

4 Konzeptionsphase

4.1 Datenbankstruktur modellieren

4.1.1 Entity Relationship Model

Das dargestellte Entity-Relationship-Modell in Abbildung 2 enthält alle Entitätstypen, ihre zugehörigen Attribute, Beziehungen und Kardinalitäten.

Nachstehende Tabelle erläutert die Verbindungen zwischen den Entitätstypen sowie die resultierenden Kardinalitäten:

Tabelle 1: Beziehungen und Kardinalitäten im Entity-Relationship-Modell

Beziehung	**Beschreibung**	**Kardinalität**
Beziehung zwischen den Entitätstypen „Patient“ und „Termin“	Ein Patient kann beliebig viele Termine vereinbaren, wobei jeder einzelne Termin nur von einem bestimmten Patienten vereinbart werden kann.	(1:N)
Beziehung zwischen den Entitätstypen „Arzt“ und „Termin“	Einem Arzt werden beliebig viele Termine zugewiesen, wobei jedem Termin genau ein Arzt zugewiesen wird.	(1:N)
Beziehung zwischen den Entitätstypen „Patient“ und „Arzt“	Ein Arzt kann mehrere Patienten betreuen, während jeder Patient von genau einem Arzt betreut wird.	(1:N)
Beziehung zwischen den Entitätstypen „Arzt“ und „Raum“	Ein Raum ist einem Arzt zugeordnet, wobei jedem Arzt genau ein Raum zugeordnet ist.	(1:1)
Die rekursive Vertretungsbeziehung	Ein Arzt kann einen oder mehrere Vertreter haben, die ihn bei Abwesenheit vertreten. Gleichzeitig kann ein Vertreter ebenfalls als Arzt fungieren und seinerseits Vertreter haben.	(N:M)

4.1.2 Entity Relationship Diagramm

Das Entity-Relationship-Diagramm, dargestellt in Abbildung 3, umfasst alle erforderlichen Datenbanktabellen für die Datenspeicherung. Die Entitätstypen wurden in Tabellen und die Attribute in Spalten umgewandelt. Jede Tabellenspalte erhielt einen entsprechenden Datentyp und eine Länge. Zudem wurde die (N:M) Vertretungsbeziehung in eine separate Tabelle namens „Vertretung“ aufgelöst und eine neue Tabelle „Benutzer“ zur Speicherung der Logindaten modelliert. Um die Datenintegrität sicherzustellen, erhielt jede Tabelle einen Primärschlüssel sowie die notwendigen Fremdschlüssel aus anderen Tabellen, um die Beziehungen herzustellen.
Hier ist die genauere Beschreibung der Datenbanktabellen :

Die Raum-Tabelle :

Sie dient zur Speicherung von Raumdaten und enthält zwei Spalten : raum_nr und status.

- raum_nr: Eine Zeichenkette mit einer maximalen Länge von 3 Bytes, die die Raumnummer des Arztes repräsentiert. Ein Beispiel für eine gültige raum_nr wäre „R01“.

- status: Ebenfalls eine Zeichenkette mit einer maximalen Länge von 7 Bytes. Der Status kann entweder „frei“ oder „besetzt“ sein und gibt Auskunft darüber, ob der Raum einem Arzt gehört (besetzt) oder nicht (frei).

Die Arzt-Tabelle :

Sie speichert Informationen zu Ärzten und enthält 7 Spalten .

- arzt_id: Ein numerischer Wert mit einer Länge von maximal 10 Ziffern, der die eindeutige ID des Arztes repräsentiert.
- anrede: Eine Zeichenkette mit einer Länge von maximal 4 Bytes, die entweder „Herr“ oder „Frau“ als Anrede speichert.
- titel: Eine Zeichenkette mit einer Länge von maximal 11 Bytes, die den Titel des Arztes speichert. Mögliche Werte sind „Dr.med“ oder „Prof.Dr.med“.
- vorname: Eine Zeichenkette mit einer Länge von maximal 30 Bytes für den Vornamen des Arztes.
- nachname: Eine Zeichenkette mit einer Länge von maximal 30 Bytes für den Nachnamen des Arztes.
- raum_nr: Eine Zeichenkette mit einer Länge von maximal 3 Bytes, die die Raumnummer des Arztes angibt.
- beruf: Eine Zeichenkette mit einer Länge von maximal 11 Bytes, die einen der folgenden Berufe speichert:

"Hausarzt", "Facharzt", "Chirurg", "Kinderarzt", "Gynäkologe", "Augenarzt", "HNO-Arzt", "Radiologe", "Anästhesist", "Psychiater", "Onkologe", "Geriater", "Notarzt".

Die Patient-Tabelle :

Diese Tabelle enthält Informationen zu den Patienten.

- patient_id: Ein numerischer Wert mit einer Länge von maximal 10 Ziffern, der die eindeutige ID des Patienten repräsentiert.
- anrede: Eine Zeichenkette mit einer maximalen Länge von 4 Bytes, die entweder "Herr" oder "Frau" als Anrede des Patienten speichert.
- vorname: Eine Zeichenkette mit einer maximalen Länge von 30 Bytes, die den Vornamen des Patienten speichert.
- nachname: Eine Zeichenkette mit einer maximalen Länge von 30 Bytes, die den Nachnamen des Patienten speichert.
- telefon: Eine Zeichenkette mit einer Länge von 14 Bytes, die die Telefonnummer des Patienten speichert. Telefonnummern müssen mit der Vorwahl "0" beginnen und werden von 13 Ziffern gefolgt.
- geburtsdatum: Ein Datumsattribut, das das Geburtsdatum des Patienten speichert.
- arzt_id: Ein numerischer Wert mit einer Länge von maximal 10 Ziffern, der die ID des betreuenden Arztes speichert.

Die Benutzer-Tabelle :

Sie speichert Informationen zu Benutzerkonten.

- benutzername: Eine Zeichenkette mit einer maximalen Länge von 30 Bytes, die den Benutzernamen speichert.

- passwort: Eine Zeichenkette mit einer Länge von 128 Bytes, die das Passwort des Benutzers speichert. Das Passwort wird unter Verwendung des Secure Hash Algorithmus SHA-512 verschlüsselt.
  Umrechnung : 512 Bit / 8 Bit =64 Byte → Der 64-Byte-Hash wird in einer hexadezimalen Darstellung gespeichert, wodurch jeder Byte-Wert zwei hexadezimale Zeichen ergibt →die benötigte Länge 64 Byte x 2 =128 Byte

Die Termin-Tabelle :

Sie speichert Informationen zu Terminen zwischen Patienten und Ärzten.

- termin_id: Ein numerischer Wert mit einer Länge von maximal 10 Ziffern, der die eindeutige ID des Termins repräsentiert.
- datum: Ein Datumsattribut, das das Datum des Termins speichert.
- uhrzeit: Eine Zeichenkette mit einer maximalen Länge von 5 Bytes, die die Uhrzeit des Termins speichert. Eine mögliche Uhrzeit wäre "15:00".
- status: Eine Zeichenkette mit einer Länge von 2 Bytes, die den Status des Termins angibt. Mögliche Werte sind: "Vorgenommen (V)" oder "Nicht Vorgenommen (NV)".
- patient_id: Ein numerischer Wert mit einer Länge von maximal 10 Ziffern, der die ID des betreffenden Patienten speichert.
- arzt_id: Ein numerischer Wert mit einer Länge von maximal 10 Ziffern, der die ID des betreffenden Arztes speichert.

Die Vertretung-Tabelle:

Sie dient dazu, Vertretungen von Ärzten zu verwalten, wenn ein Arzt einen anderen Arzt für einen bestimmten Zeitraum vertritt.

- vertretungs_id: Ein numerischer Wert mit einer Länge von maximal 10 Ziffern, der die eindeutige ID der Vertretung repräsentiert.
- vertreter_id: Ein numerischer Wert mit einer Länge von maximal 10 Ziffern, der die ID des vertretenden Arztes speichert.
- arzt_id: Ein numerischer Wert mit einer Länge von maximal 10 Ziffern, der die ID des vertretenen Arztes speichert.
- startdatum: Ein Datumsattribut, das den Beginn des Vertretungszeitraums angibt.
- enddatum: Ein Datumsattribut, das das Ende des Vertretungszeitraums angibt.
- grund: Eine Zeichenkette mit einer maximalen Länge von 2 Bytes, die Informationen über den Grund der Vertretung speichert.
  Mögliche Gründe: "Beurlaubt (B)", "Arbeitsunfähig (AU)", "Gekündigt (GK)" oder "Rente (R)".

4.2 Aktivitätsdiagramm für die Terminvergabe erstellen

Für das Hinzufügen eines neuen Datensatzes in die Termin-Tabelle werden die Primärschlüssel des Patienten und des Arztes (Betreuers), das Termindatum sowie die Terminuhrzeit benötigt. Der Terminstatus wird vernachlässigt, da er eher irrelevant ist.
Das Aktivitätsdiagramm in Abbildung 4 dargestellt, beschreibt den Prozess der Terminvergabe, der in vier Schritte unterteilt ist. Zunächst gibt der Benutzer das Geburtsdatum des jeweiligen Patienten ein, woraufhin eine Liste aller Patienten erscheint, die an diesem Datum geboren sind. Anschließend wählt der Benutzer einen Patienten aus dieser Liste aus, wobei die IDs zwischengespeichert werden. Danach fordert das System den Benutzer auf, ein Datum für den Termin auszuwählen, um die

verfügbaren Uhrzeiten anzuzeigen und eine passende Uhrzeit zu wählen. Abschließend werden nach Bestätigung der Termindaten diese in die Datenbank gespeichert.
Während dieses Prozesses werden Fehlermeldungen angezeigt, falls die Suche nach Patienten oder die Auswahl des Datums nicht erfolgreich war

4.3 Graphische Benutzeroberfläche entwerfen

Die grafische Benutzeroberfläche wurde durch die Erstellung von zwei Mockups entworfen: einem für das Login-Fenster und einem für das Hauptfenster siehe Abbildung 5 . Beim Start der Anwendung wird das Login-Fenster angezeigt. Es besteht aus Eingabefeldern für Benutzernamen und Passwort sowie einem "Anmelden"-Button. Durch die Eingabe korrekter Zugangsdaten erfolgt der Zugriff auf das Hauptfenster.
Das Hauptfenster präsentiert eine Menüleiste, die die Navigation zwischen den verschiedenen Anwendungsfunktionen ermöglicht. Die Menüleiste umfasst fünf Menüs: Home, Raum, Patient, Arzt und Termin. Jedes Menü beinhaltet weitere Menüpunkte, die die Sichtbarkeit des Hauptfensters anpassen oder spezifische Aufgaben ausführen können.
In der folgenden Tabelle ist detailliert beschrieben, welche Funktionen den einzelnen Menüpunkten zugeordnet sind:

Tabelle 2 : Funktionalitäten der einzelnen Menüpunkte

Menü	**Menüpunkt**	**Beschreibung**
Home	Zurück zur Homepage	Rückkehr zur Startseite der Anwendung.
	Abmelden	Beendet die aktuelle Sitzung und loggt den Benutzer aus.
	Programm beenden	Schließt die Anwendung.
Termin	Termin vergeben	Öffnet die Benutzeroberfläche zum Hinzufügen neuer Termine.
	Termin stornieren	Öffnet die Benutzeroberfläche zum Löschen vorhandener Termine.
	Termin-Liste	Zeigt eine Liste aller vorhandenen Termine an.
Arzt	Arzt einfügen	Öffnet die Benutzeroberfläche zur Erfassung neuer Arztdaten.
	Arzt löschen	Öffnet die Benutzeroberfläche zum Löschen von Arztdaten.
	Arztdaten aktualisieren	Öffnet die Benutzeroberfläche zur Aktualisierung der vorhandenen Arztdaten.
	Arzt-Liste	Zeigt eine Liste aller vorhandenen Ärzte an.
Patient	Patient einfügen	Öffnet die Benutzeroberfläche zur Erfassung neuer Patientendaten.
	Patient löschen	Öffnet die Benutzeroberfläche zum Löschen von Patientendaten .
	Patientendaten ändern	Öffnet die Benutzeroberfläche zur Aktualisierung der vorhandenen Patientendaten.
	Patient-Liste	Zeigt eine Liste aller vorhandenen Patienten
Raum	Raum einfügen	Öffnet die Benutzeroberfläche zur Erfassung neuer Raumdaten.
	Raum löschen	Öffnet die Benutzeroberfläche zum Löschen von Raumdaten.
	RaumNr editieren	Öffnet die Benutzeroberfläche zur Aktualisierung der Raumnummern
	Raumliste	Zeigt eine Liste alle vorhandenen Räume an .

4.4 Schichten Architektur festlegen

Das Projekt nutzt das Architekturmuster Model View Controller (MVC), das in drei Schichten strukturiert ist, um klare Aufgabenbereiche zu definieren.
Die Model-Schicht fungiert als Bindeglied zur Datenbank. Sie ermöglicht das Speichern und Laden von Daten nach einer erfolgreichen Verbindungsherstellung.

Die View-Schicht konzentriert sich ausschließlich auf die Darstellung der Daten und die Benutzerinteraktion. Hier werden sämtliche GUI-Komponenten platziert und Fehler- sowie Erfolgsmeldungen dargestellt.
Der Controller übernimmt die Steuerung der Anwendung. Er verarbeitet und validiert Benutzereingaben, agiert als Regulator für Befehle und greift auf die von View und Model bereitgestellten Methoden zu. Dabei reagiert er auf Benutzerereignisse wie Klicks oder Auswahlvorgänge, um die grafische Benutzeroberfläche entsprechend zu aktualisieren.
Die Entscheidung für das MVC-Muster basierte auf seinen zahlreichen Vorteilen. Es ermöglicht eine klare Trennung der Schichten, wodurch bei Änderungen gezielt nur eine Schicht ausgetauscht werden kann. Zudem bietet diese Architektur ein hohes Potenzial zur Wiederverwendung von Komponente

5 Implementierungsphase

5.1 Implementierung der Datenbank

5.1.1 Datenbankumgebung und Entwicklungs-Tools

In der Entwicklung und Verwaltung der Datenbank wurde die Oracle Database Express Edition (XE) in der Version 21c als Datenbankmanagementsystem verwendet. Für die Entwicklungsumgebung kam der SQL Developer zum Einsatz. Die gesamte Datenverarbeitung erfolgte lokal auf dem eigenen Rechner, da kein Server betrieben wurde. Dies ermöglichte die lokale Speicherung und Verwaltung der Daten.

5.1.2 Erstellung der Datenbank-Objekte

Nach der Einrichtung eines neuen Users (Schema) namens „ c##cesar“ wurden die Tabellen Benutzer, Raum, Patient, Arzt, Termin und Vertretung erstellt. Sowohl Primärschlüssel als auch Fremdschlüssel wurden im Nachhinein eingefügt, um die Datenbeziehungen zu definieren. Es wurden Unique-Constraints für die Telefonnummer in der Patiententabelle sowie für spezifische Spaltenkombinationen eingeführt, um Duplikate zu verhindern und die Datenkonsistenz zu gewährleisten. Zusätzlich wurden Sequenzen eingerichtet, um eine automatische Inkrementierung zu ermöglichen.
Ein darauf folgender Trigger wurde konfiguriert, um den Raumstatus in der Raum-Tabelle entsprechend den Änderungen in der Arzt-Tabelle zu aktualisieren (siehe Abbildung 6). Zum Abschluss wurden zwei Views erstellt. Der erste View dient der Patientensuche, während der zweite View für die Anzeige von Termininformationen genutzt wird. Beide Views aktualisieren sich automatisch bei Änderungen in den zugrunde liegenden Tabellen.
Zum Abschluss dieser Phase wurde ein Datensatz in die Benutzer-Tabelle eingefügt. Das Passwort wurde mit der Funktion "DBMS_CRYPTO.HASH" unter Verwendung des Parameters 6 eingefügt, was einem SHA512-Algorithmus entspricht.

5.2 Implementierung der Anwendung

5.2.1 Entwicklungsumgebung und externe Abhängigkeiten

Für die Entwicklung dieses Projekts wurde die IntelliJ-Entwicklungsumgebung in Verbindung mit Oracle Open JDK Version 21.0.1 genutzt. Zur Anbindung an die Datenbank wurde der Treiber OJDBC in Version 11 verwendet. Die Gestaltung der Benutzeroberfläche erfolgte mithilfe der Swing-Bibliothek. Zusätzlich wurde die Bibliothek Toedter JCalendar in Version 1.4 eingesetzt, um spezielle GUI-Komponenten wie den JDateChooser zur Auswahl von Geburts- und Termininformationen zu integrieren. Des Weiteren kam die Jollyday API in Version 0.5.10 zum Einsatz, um die Feiertage in Nordrhein-Westfalen, dem Standort der Arztpraxis in Bonn, für ein spezifisches Jahr zu ermitteln.

5.2.2 Implementierung der 3 Schichten

Während der Implementierungsphase wurden zunächst drei Packages für das View-Model und den Controller erstellt, um die Struktur des Systems festzulegen. Die View wurde mit zwei Frames gestaltet, die die erforderlichen GUI-Komponenten in verschiedenen Containern (Panels) organisierten.
Als nächster Schritt wurde das Model entwickelt, in dem Methoden zur Verbindungsherstellung sowie für DML Operationen definiert wurden.
Im Controller wurde eine Klasse für den „Anmelden"-Button erstellt, um den Zugang vom Login-Fenster zum Hauptfenster zu ermöglichen.
Angesichts der verschlüsselten Speicherung der Passwörter in der Benutzer-Tabelle wurde eine spezielle Methode entworfen, um das Entschlüsseln der Passwörter zu ermöglichen.

Anschließend erfolgte schrittweise die Umsetzung der Funktionalitäten der einzelnen Menüpunkte gemäß Tabelle 2 und der definierten Anwendungsfälle aus Abbildung 1. Diese Phase lief parallel zur Testphase und wurde erfolgreich mit der Implementierung des Terminvergabeprozesses aus Abbildung 4 abgeschlossen.

6 Testphase

Die Anwendung unterzog sich einer Phase von Black-Box-Tests, um das Verhalten der Anwendung anhand der bereitgestellten Benutzeroberfläche zu überprüfen, ohne dabei interne Code- oder Implementierungsdetails zu berücksichtigen. Diese Tests bewerten die Benutzererfahrung und stellen sicher, dass die Anwendung die erwarteten Ergebnisse liefert, unabhängig von den internen Prozessen.
Parallel zur Implementierungsphase durchgeführt, prüften diese Tests die implementierten Klassen und Methoden.
Jede Klasse wurde individuell überprüft, während auch jede Methode, die eine Eingabe verarbeitete und einen Rückgabewert erzeugte, separat getestet wurde.
Allerdings waren nicht alle Tests immer erfolgreich. Anpassungen in der Implementierung waren erforderlich, um das angestrebte Ziel zu erreichen.
Nach dem Einfügen, Löschen oder Aktualisieren von Daten wurde die Datenbank auch überprüft, um sicherzustellen, dass die vorgenommenen Änderungen erfolgreich umgesetzt wurden.

7 Dokumentationsphase

Eine Entwicklerdokumentation in deutscher Sprache wurde erstellt. Die Funktionalitäten in den einzelnen Klassen und Methoden wurden ausführlich und umfassend als Kommentare beschrieben.

8 Fazit

Durch mein erstes Projekt habe ich durch eine strukturierte Vorgehensweise und gezielte Entwicklungsschritte alle Funktionalitäten gemäß den Anforderungen erreicht. Es war eine wertvolle Erfahrung, bei der ich eine umfassende Entwicklerdokumentation erstellte, um die Wartung und Weiterentwicklung des Systems zu erleichtern.
Im Verlauf dieses Projekts sammelte ich wertvolle Erkenntnisse, besonders hinsichtlich der essenziellen Rolle einer detaillierten Planung. Ich identifizierte die Notwendigkeit einer verbesserten Benutzeroberfläche und erkannte die Bedeutung der Erstellung von Unit Tests. Diese Aspekte sollten in zukünftigen Iterationen als zentrale Schwerpunkte betrachtet werden. Obwohl ich während dieses Projekts keine Unit Tests durchgeführt habe, motiviert mich diese Erfahrung, sie in kommenden Projekten als einen unverzichtbaren Bestandteil meiner Arbeitsweise zu integrieren.

Anhang

Abbildung 2: Entity Relationship Model (ERM)

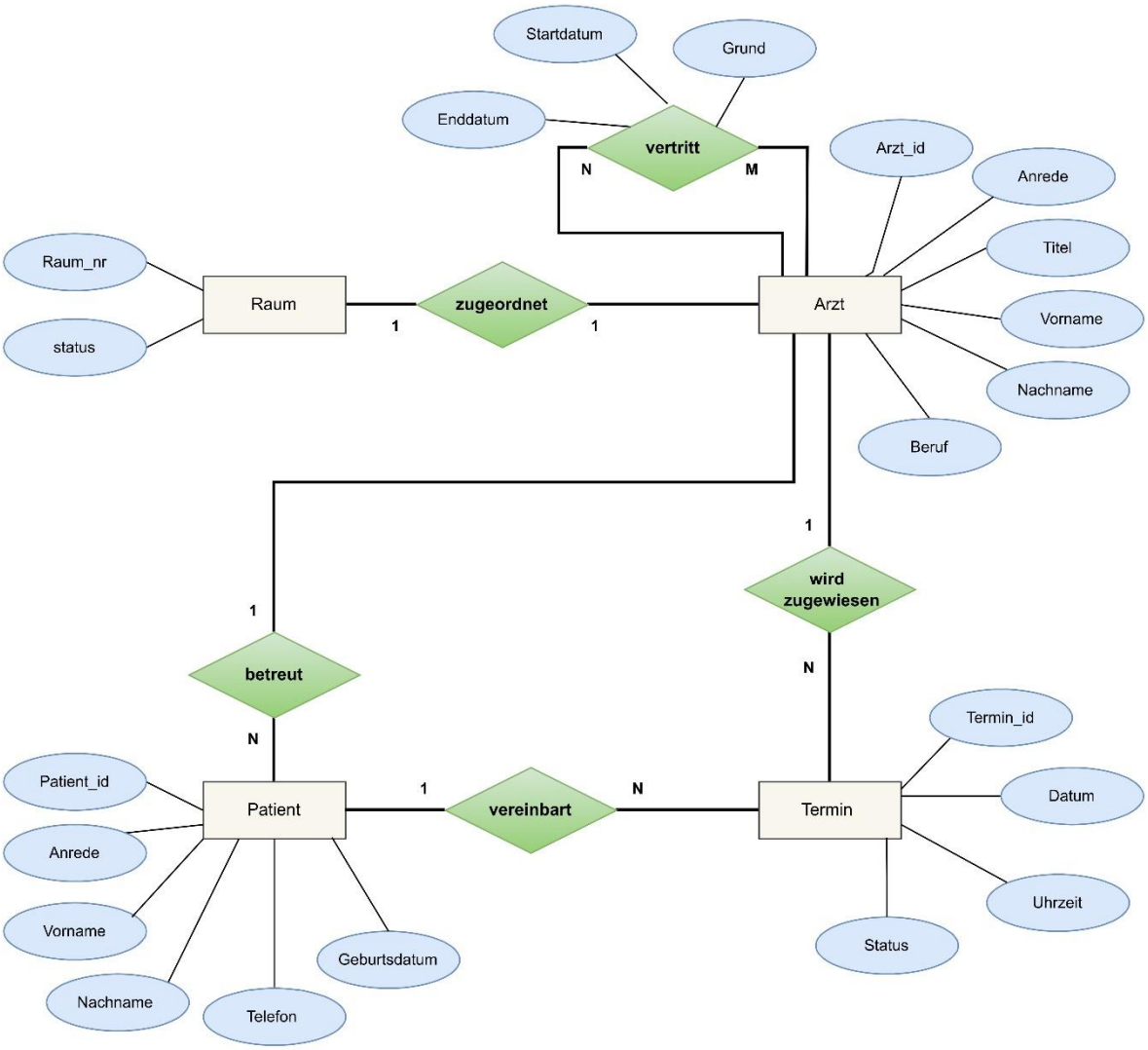

Abbildung 3: Entity Relationship Diagramm (ERD)

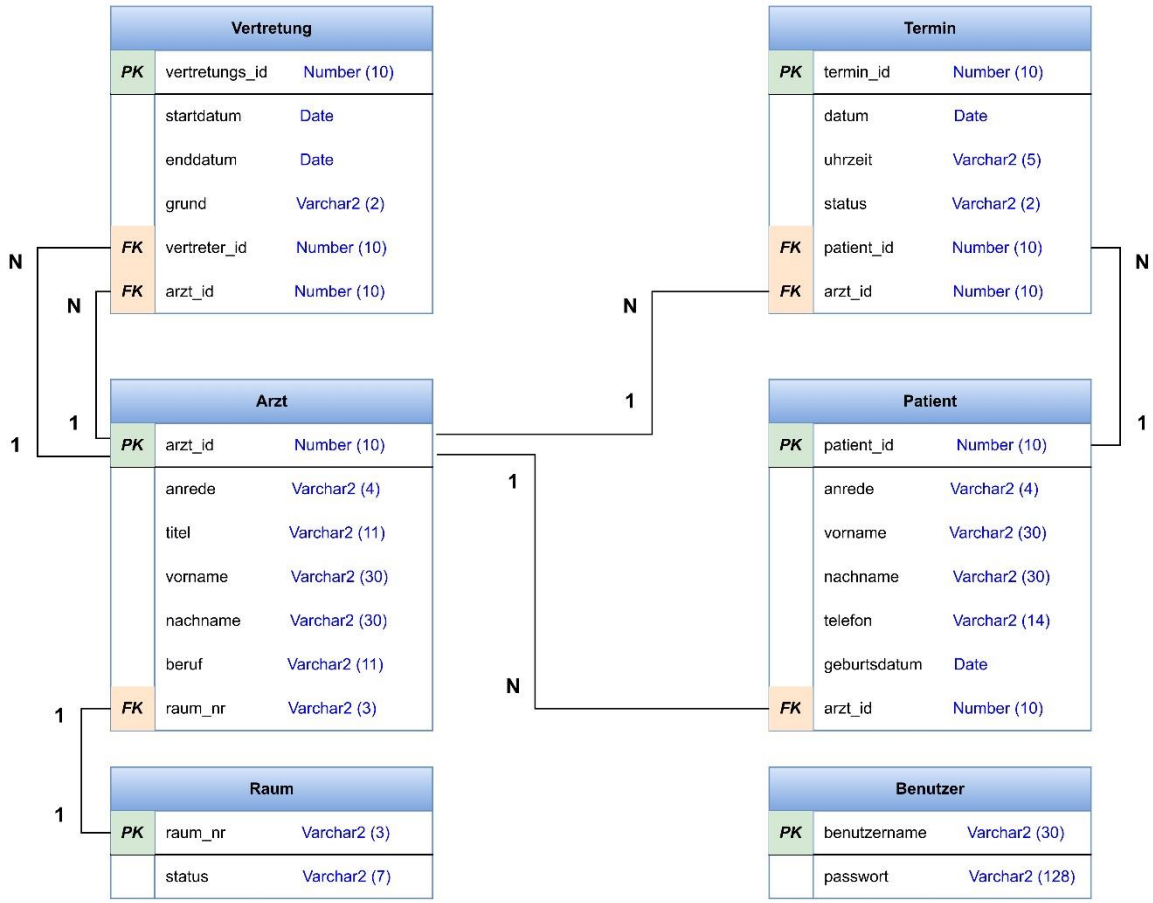

Abbildung 4: Aktivitätsdiagramm (activity diagram)

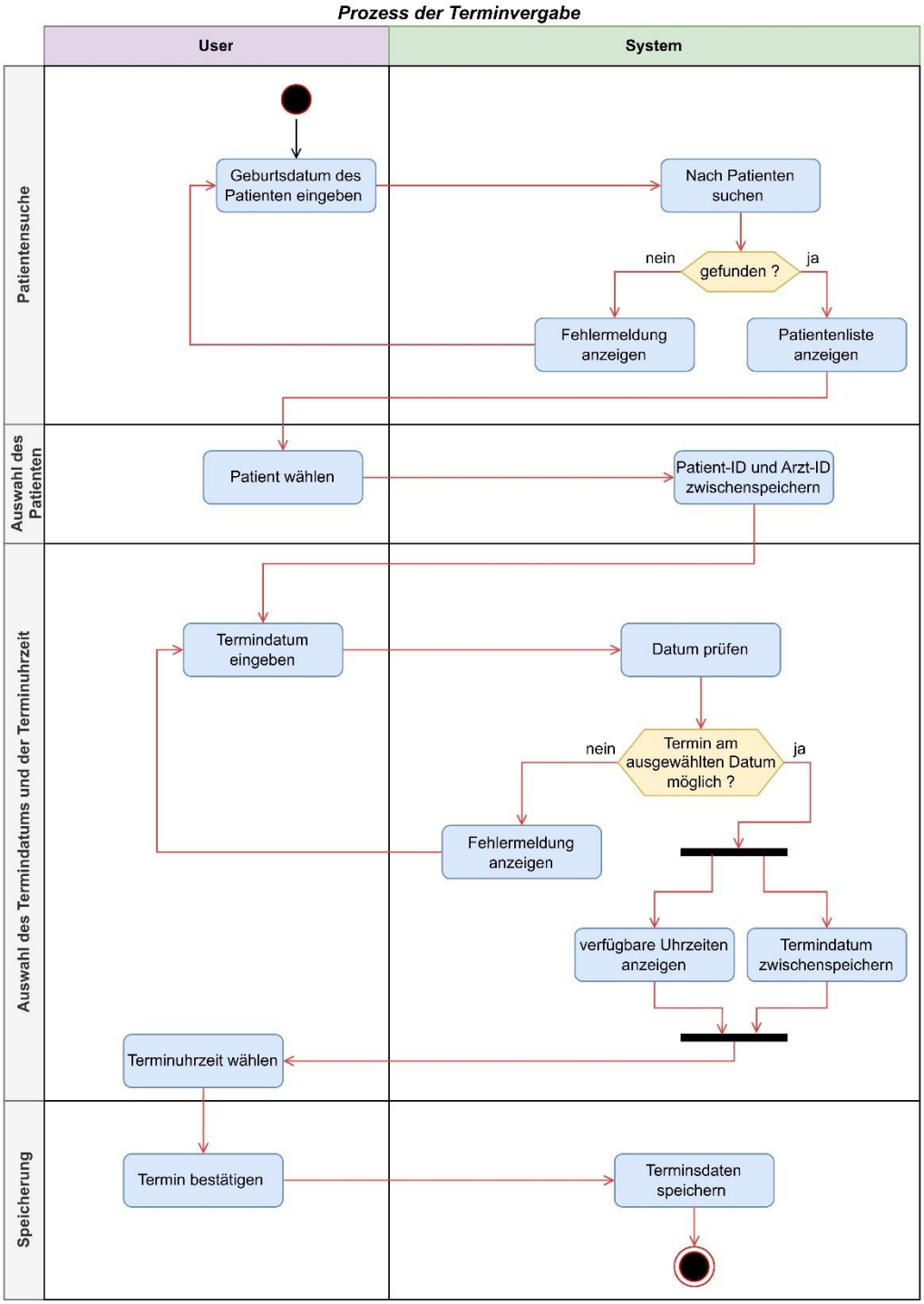

Abbildung 5: Entwurf der graphischen Benutzeroberfläche

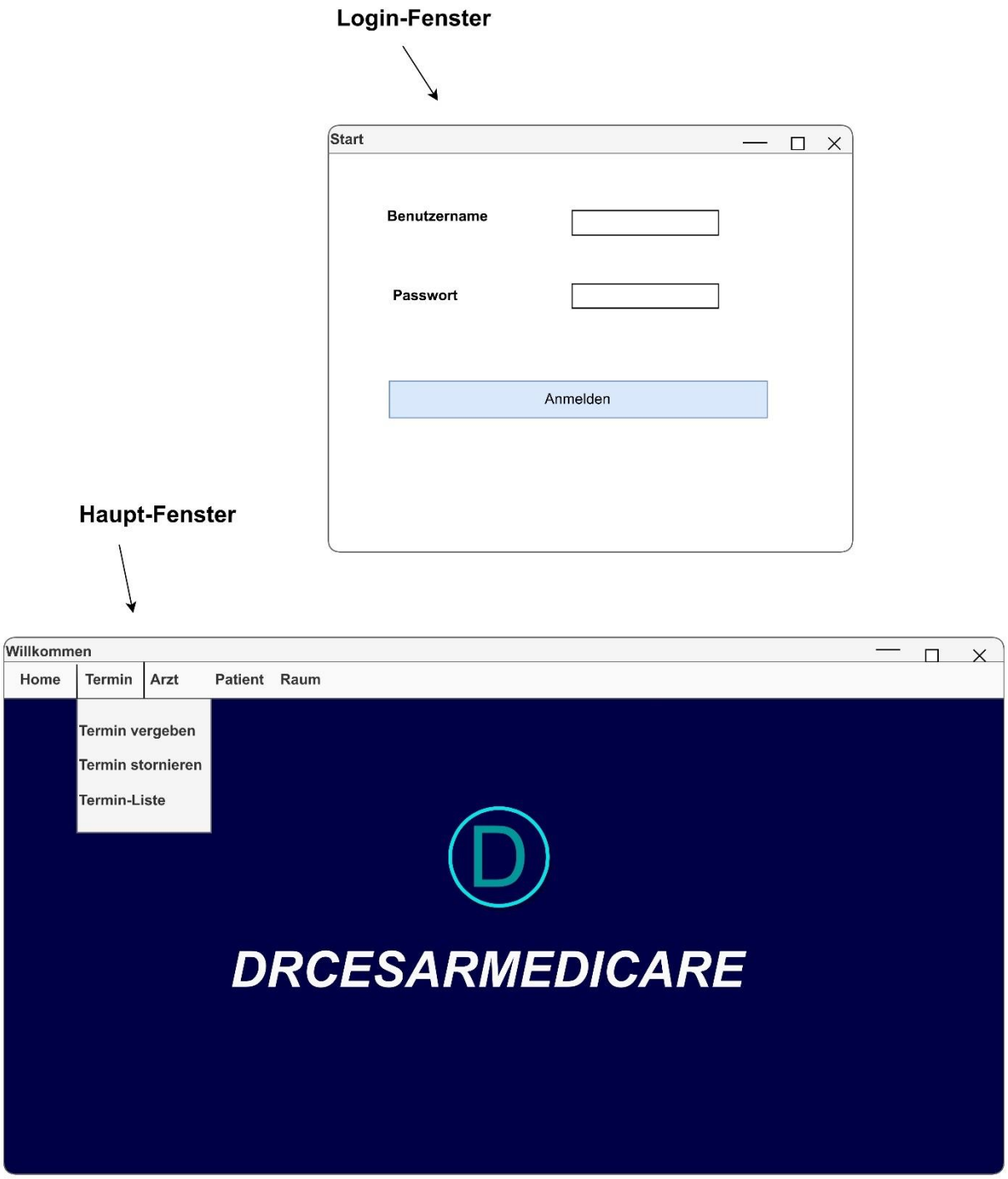

Abbildung 6 : Trigger (Raumstatuswechseln)

```
/**
Der Trigger "Raumstatuswechseln" aktualisiert den Raumstatus in der
"raum"-Tabelle basierend auf Änderungen in der "arzt"-Tabelle.
Beim Einfügen markiert er Räume als 'besetzt', beim Löschen als 'frei'
und bei
einer Raumnummeränderung aktualisiert er den entsprechenden Status.
*/
CREATE OR REPLACE TRIGGER   Raumstatuswechseln
AFTER INSERT OR DELETE OR UPDATE OF raum_nr ON arzt
FOR EACH ROW
DECLARE
--Keine Variablen erforderlich
BEGIN
    IF INSERTING THEN

        IF :NEW.raum_nr IS NOT NULL THEN
            UPDATE raum SET raum.status= 'besetzt'
            WHERE raum.raum_nr=:NEW.raum_nr;
        END IF ;

    ELSIF DELETING THEN

        IF :OLD.raum_nr IS NOT NULL THEN
            UPDATE raum SET raum.status= 'frei'
            WHERE raum.raum_nr=:OLD.raum_nr;
        END IF;


     ELSIF UPDATING ('raum_nr') THEN
        IF (:OLD.raum_nr IS NOT NULL) AND (:NEW.raum_nr IS NOT NULL) AND
                            (:OLD.raum_nr != :NEW.raum_nr) THEN

            UPDATE raum SET raum.status = 'frei'
            WHERE raum.raum_nr = :OLD.raum_nr;
            UPDATE raum SET raum.status = 'besetzt'
            WHERE raum.raum_nr = :NEW.raum_nr;

        END IF;
    END IF;
END;
```