

Projektabschlussbericht

MediCare

07.07.2024

Inhaltsverzeichnis

1 Organisation.....	1
2 Kurzzusammenfassung.....	1
3 Verwendete Technologien.....	2
4 Lösungsstrategie.....	2
5 Algorithmen, Datenstrukturen und Architektur.....	6
5.1 Algorithmen.....	7
5.2 Zukünftige Verbesserungen.....	13
6 Referenzen.....	13
7 Anhang A – Quellcode.....	16

1 Organisation

Autor(en)	Marie Rose, Assena Satarkulova
Eindeutige Teamnummer	9PG_T4
Namen der Teammitglieder	Marie Rose, Assena Satarkulova

2 Kurzzusammenfassung

Unser Projektziel war es, ein Medication Management System in JavaFX zu implementieren, um eine zentrale Herausforderung in Krankenhäusern und Pflegeheimen anzugehen: die effiziente und fehlerfreie Verwaltung der Medikamenteneinnahme. Das System adressiert Probleme wie die Schwierigkeit, den Überblick über die Medikamenteneinnahme mehrerer Patienten zu behalten. Ein zentraler Aspekt ist dabei die Reduzierung von Medikationsfehlern durch präzise Überwachung und Erinnerungsfunktionen und die Verbesserung der Sicherheit und Lebensqualität der Patienten.

Das entwickelte System bietet eine umfassende Lösung für diese Probleme. Es ermöglicht die Erstellung individueller Medikamentenpläne für jeden Patienten, unter Berücksichtigung der spezifischen Medikamente, Nebenwirkungen und Einnahmezeiten. Die Verwaltung von Medikamenten und Patientendaten wird durch Funktionen zum Hinzufügen und Löschen von Informationen erleichtert. Zudem bietet das System eine Erinnerungsfunktion, die das Pflegepersonal automatisch an anstehende Medikamentengaben erinnert, wodurch das Risiko von vergessenen oder verspäteten Gaben minimiert wird.

Die Anzeige relevanter Informationen zu den Nebenwirkungen der Medikamente trägt zur Patientensicherheit bei, indem sie das Pflegepersonal über mögliche Risiken informiert und eine fundierte Entscheidung über die Medikamentenverabreichung ermöglicht.

Zur Umsetzung dieser Funktionen wurden verschiedene Methoden, Verfahren und Werkzeuge eingesetzt. Die Programmiersprache Java bildete die Grundlage für die Entwicklung, während eine geeignete Entwicklungsumgebung wie IntelliJ IDEA die Programmierung und das Testen des Systems erleichterte. Zur Erstellung der grafischen Benutzeroberfläche wurde eine GUI-Bibliothek wie JavaFX verwendet, um eine intuitive und benutzerfreundliche Bedienung zu gewährleisten.

Die Speicherung und Organisation der Daten erfolgte mithilfe von Listen und benutzerdefinierten Klassen für Patienten und Medikamente. Das Designprinzip Model-View-Controller (MVC) gewährleistete eine klare Trennung von Daten, Präsentation und Steuerung, während die objektorientierte Programmierung die Strukturierung des Codes und die Wiederverwendbarkeit von Komponenten förderte.

Durch die Nutzung von Ressourcen wie Vorlesungen, Übungen und Fachliteratur konnten wir das gelernte Wissen praktisch anwenden und erfolgreich in die Entwicklung des MediCare einfließen lassen. Die iterative Entwicklung von Prototypen ermöglichte es uns, unsere Fähigkeiten und unser Verständnis für die Umsetzung kontinuierlich zu verbessern. Auf Basis dieser Prototypen konnten wir schließlich die finale Version des Systems entwickeln, in der alle Projektziele erfolgreich erfüllt wurden.

3 Verwendete Technologien

Das vollständige Programm wurde unter Verwendung der Programmiersprache Java entwickelt. Die Entwicklungsumgebung IntelliJ IDEA Community Edition 2024.1.3 (Laufzeitversion: 17.0.11+1-b1207.24 amd64) wurde eingesetzt, wobei das SDK zulu-21 (Azul Zulu Version 21.0.2) verwendet und das Sprachlevel auf 17 - Sealed Types, Always-Strict Floating-Point Semantics festgelegt wurde. Für die Erstellung der grafischen Benutzeroberfläche (GUI) wurde JavaFX 17 verwendet und für die Layout-Definition der Benutzeroberflächenkomponenten FXML, ein XML-basiertes Markup zur Definition der Benutzeroberfläche. JavaFX 17 wurde gewählt, da ein Mitglied des Teams einen älteren Computer hat und diese Version eine bessere Kompatibilität mit älteren Systemen bietet.

4 Lösungsstrategie

Um eine Lösung für die Umsetzung unseres Softwareprodukts zu finden, sind wir systematisch vorgegangen und haben verschiedene Prototypen entwickelt. In diesem Abschnitt werden wir beschreiben, wie wir dabei vorgegangen sind und welche Erkenntnisse wir mithilfe dieser Prototypen gewonnen haben.

Zu Beginn des MediCare-Projekts wurde eine umfassende Mindmap (siehe Abbildung 1) erstellt, um die gewünschten Funktionalitäten und die Struktur der Benutzeroberfläche visuell darzustellen. Diese Mindmap diente als Leitfaden für die weitere Entwicklung und half dabei, den Überblick über das Projekt zu behalten und sicherzustellen, dass keine wichtigen Aspekte übersehen wurden.

Anhand dieser detaillierten Planung konnten wir effizient mit der Entwicklung der Prototypen beginnen. Die Mindmap zeigte deutlich die Hauptbereiche des Projekts:

- **Login:** Der Einstiegspunkt in die Anwendung, der eine Benutzeroauthentifizierung erfordert.
- **Patientenliste:** Die zentrale Übersicht über alle erfassten Patienten, einschließlich ihrer Geburtstage und zugeordneten Medikamentenindizes. Hier können neue Patienten hinzugefügt oder bestehende gelöscht werden.
- **Medikamentenliste:** Eine Übersicht über alle verfügbaren Medikamente mit ihren Nebenwirkungen und Indizes. Auch hier ist das Hinzufügen und Löschen von Medikamenten möglich.
- **Erinnerungsfunktion:** Eine Funktion, die Patienten an die Einnahme ihrer Medikamente erinnert.

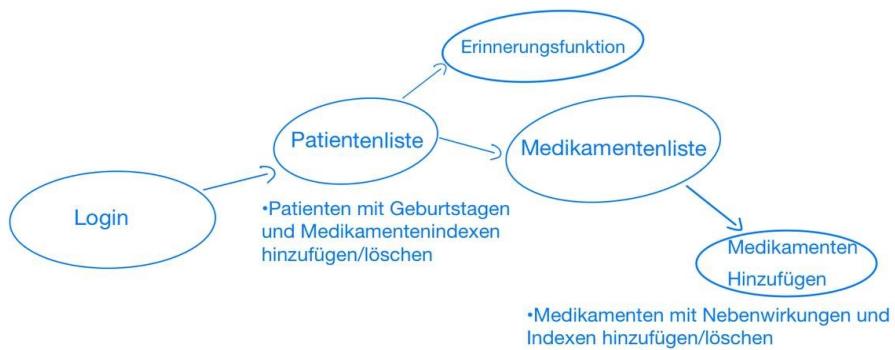


Abbildung 1: MindMap

Diese Strukturierung ermöglichte es uns, gezielt an den einzelnen Komponenten zu arbeiten und schrittweise Prototypen zu entwickeln, die die in der Mindmap festgelegten Funktionen umsetzen.

1. Prototyp

Medication Management System

Willkommen! Bitte wählen Sie Ihre/n Patient/in aus:

- Anna Braun - 1990-05-15
- Max Mustermann - 1985-03-20
- Johannes Bauer - 1975-07-30

Medikamentenliste

Neues Medikament hinzufügen

Medikamentenliste

- Paracetamol
- Aspirin
- Zostex

Abbildung 2: Prototyp 1
Projektabschlussbericht

Abbildung 3: Prototyp 1
Praktikum INF2, Sommersemester 2024

Der erste Prototyp des MediCare-Projekts konzentriert sich auf die Kernfunktionalität der Patienten- und Medikamentenverwaltung und dient als Grundlage für zukünftige Erweiterungen.

Im Zentrum des Prototyps steht die Implementierung der Model-View-Controller (MVC)-Architektur. Diese ermöglicht eine klare Trennung von Daten, Benutzeroberfläche und Steuerungslogik. Das Model, bestehend aus den Klassen *PatientModel* und *MedicationModel*, verwaltet die Patientendaten (Name, Geburtsdatum) und die verfügbaren Medikamente. Die View, repräsentiert durch die Klassen *PatientView* und *MedicationsView*, zeigt diese Daten dem Benutzer an und ermöglicht die Interaktion. Der Controller agiert als Vermittler zwischen Model und View, indem er Benutzereingaben verarbeitet, das Model aktualisiert und die View entsprechend anpasst.

Die Datenstrukturen im Prototyp sind bewusst einfach gehalten, um eine unkomplizierte Implementierung und effiziente Verwaltung der Daten zu gewährleisten. Hierbei kommen vor allem ObservableLists zum Einsatz, um die Medikamentenlisten jedes Patienten zu verwalten. Diese Datenstruktur ist besonders wertvoll in unserer JavaFX-basierten Anwendung, da sie Änderungen in der Liste automatisch an die Benutzeroberfläche weitergeben kann. Dadurch wird jede Modifikation, wie das Hinzufügen oder Entfernen eines Medikaments, sofort visuell reflektiert, ohne dass eine manuelle Aktualisierung der Benutzeroberfläche notwendig ist.

Der Ablauf des Prototyps beginnt mit der Erstellung von drei Beispielpatienten und dem Hinzufügen eines Medikaments zu einem dieser Patienten. Anschließend wird die *PatientView* angezeigt, in der der Benutzer einen Patienten auswählen kann. Durch Klicken auf eine Schaltfläche kann der Benutzer die *MedicationsView* öffnen, die die Liste der verfügbaren Medikamente anzeigt.

Obwohl der Prototyp die grundlegende Funktionalität der Patienten- und Medikamentenverwaltung abdeckt, gibt es noch einige Bereiche, die in zukünftigen Iterationen verbessert werden können. Eine wichtige Erweiterung wird die Implementierung einer persistenten Datenspeicherung wie die Serialisierung der Daten in Dateien sein. Dadurch werden die Daten beim Schließen der Anwendung nicht verloren gehen.

Darüber hinaus fehlt im Prototyp noch die Möglichkeit, Medikamente aus der *MedicationsView* einem ausgewählten Patienten hinzuzufügen sowie die Möglichkeit, die Patienten- und Medikamentenliste innerhalb der Anwendung zu verändern und zu erweitern.

Der Hauptaspekt unserer Anwendung ist die Erinnerungsfunktion. Im aktuellen Prototyp ist diese noch nicht vorhanden. Eine solche Funktion würde es ermöglichen, Erinnerungen für die Medikamenteneinnahme zu erstellen und zu verwalten, was den Hauptzweck der Anwendung unterstützt.

2. Prototyp

Abbildung 4: Login Prototyp 2

Abbildung 5: Hauptfenster Prototyp 2

Abbildung 6: Medikamentenliste Prototyp 2

Abbildung 7: Medikamenten hinzufügen
Prototyp 2

Der zweite Prototyp des MediCare-Projekts stellt eine deutliche Weiterentwicklung gegenüber dem ersten dar. Es erweitert die Kernfunktionalität um wichtige Aspekte wie ein Login-System, persistente Datenspeicherung und eine erste Version der Erinnerungsfunktion.

Login-System und MVC-Erweiterung:

Der zweite Prototyp führt ein einfaches Login-System ein, das die Benutzerauthentifizierung ermöglicht. Dies erfolgt über die neuen Komponenten *LoginView.fxml* und *LoginController*,

die in die bestehende MVC-Architektur integriert werden. Der Benutzer muss nun korrekte Anmeldeinformationen eingeben, um auf die Hauptfunktionen der Anwendung zuzugreifen.

Persistente Datenspeicherung:

Eine wesentliche Verbesserung des zweiten Prototyps ist die Implementierung der persistenten Datenspeicherung. Anstatt Daten nur temporär im Speicher zu halten, werden sie nun mithilfe der Serialisierung in Dateien gespeichert (*patientSerialize*, *medicationSerialize*). Dies gewährleistet, dass die Patientendaten und Medikamentendaten beim Schließen der Anwendung nicht verloren gehen und bei einem Neustart wieder verfügbar sind.

Grundlegende Erinnerungsfunktion:

Der zweite Prototyp legt den Grundstein für die Erinnerungsfunktion, indem er den *ReminderManager* einführt. Dieser verwaltet *Reminder*-Objekte, die eine Nachricht und einen Zeitpunkt für die Erinnerung enthalten. Ein *Timer* wird verwendet, um die Erinnerungen zu planen, und eine einfache Konsolenausgabe dient als erste Form der Benachrichtigung.

Erweiterte UI-Elemente:

Die Benutzeroberfläche wurde um wichtige Elemente erweitert, die für die Erinnerungsfunktion relevant sind. Ein *DatePicker* ermöglicht die Auswahl des Datums, und zwei *ComboBoxen* dienen zur Auswahl von Stunde und Minute für die Erinnerung. Diese Elemente werden im *MainWindowController* verwaltet.

Übergang zum finalen Code:

Aufbauend auf den Erkenntnissen aus dem zweiten Prototyp wurde der finale Code entwickelt. Dieser beinhaltet entscheidende Verbesserungen in Bezug auf Datenintegrität und Benutzerfreundlichkeit. Insbesondere wurden Funktionen zur Überprüfung auf doppelte Angaben bei Patienten und Medikamenten implementiert, um sicherzustellen, dass keine redundanten Daten in das System gelangen. Darüber hinaus wurde die Möglichkeit zum Löschen von Patienten und Medikamenten hinzugefügt, um eine flexible Verwaltung der Daten zu ermöglichen.

5 Algorithmen, Datenstrukturen und Architektur

Einleitung

Das MediCare-System basiert auf einer Model-View-Controller (MVC)-Architektur, die eine klare Trennung zwischen Datenmanagement, Benutzeroberfläche und Steuerlogik ermöglicht.

Das Model ist dabei verantwortlich für die Verwaltung der Medikamenten- und Patientendaten und die Logik zur Verarbeitung dieser Daten.

Die View stellt die Benutzeroberfläche dar, über die der Nutzer mit dem System interagiert. Hierzu haben wir insgesamt vier FXML-Dateien erstellt und diese mithilfe des Screenbuilders modelliert.

Der Controller dient als Vermittler zwischen Model und View und verarbeitet Benutzereingaben. Für jede Scene haben wir einen separaten Controller erstellt.

In Abbildung 2 wird die UML-Darstellung des finalen Codes gezeigt. Die Hauptklassen und ihre Beziehungen werden im Folgenden näher erläutert.

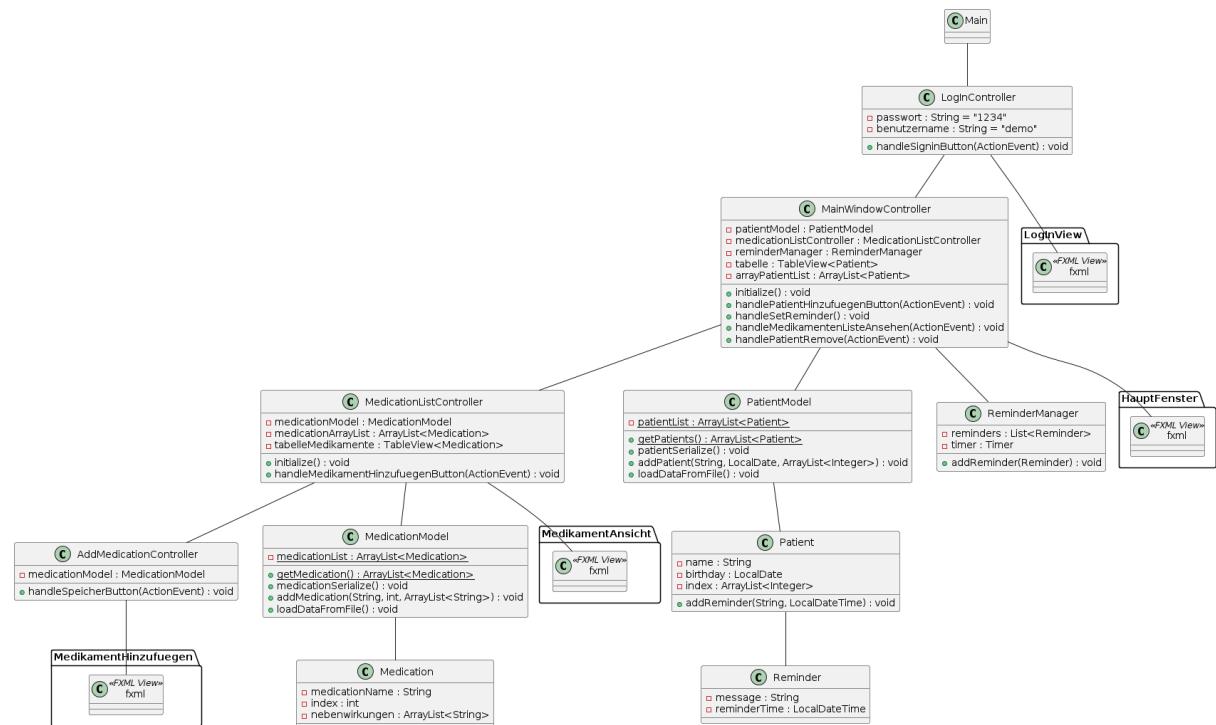


Abbildung 8: UML-Diagramm

5.1 Algorithmen

Das MediCare-Projekt implementiert verschiedene Algorithmen zur Verwaltung von Patienten, Medikamenten und Erinnerungen.

Patientenverwaltung:

Hinzufügen eines Patienten (`handlePatientHinzufuegenButton`): Dieser Algorithmus ist dafür verantwortlich, neue Patienten in das System aufzunehmen. Zunächst werden die eingegebenen Daten (Name, Geburtsdatum, Medikamentenindizes) aus den entsprechenden Textfeldern und dem DatePicker ausgelesen.

```
String name = PatientenNameTextfeld.getText();
```

```
LocalDate birthday = GeburtsdatumFeld.getValue();
```

```
String indexString = indexLabel.getText();
```

Falls kein Index eingegeben wurde, wird eine Fehlermeldung angezeigt und die Methode abgebrochen.

```
if(indexString == null || indexString.isEmpty()) {  
  
    Alert alert = new Alert(Alert.AlertType.ERROR);  
  
    alert.setTitle("Fehler");  
  
    alert.setHeaderText(null);  
  
    alert.setContentText("Bitte geben Sie einen Index ein.");  
  
    alert.showAndWait();  
  
    return;  
}
```

Anschließend wird überprüft, ob alle erforderlichen Felder ausgefüllt sind. Wenn ja, werden die Medikamentenindizes, die als kommagetrennte Zeichenkette vorliegen, in eine ArrayList umgewandelt. Dabei wird sichergestellt, dass fehlerhafte Eingaben (z.B. nicht-numerische Werte) nicht zu einem Programmabsturz führen.

```
ArrayList<Integer> medikamenteIndex = new ArrayList<>();  
  
String[] indexStrArray = indexString.split(",");  
  
for (int i = 0; i < indexStrArray.length; i++) {  
  
    try {  
  
        int index = Integer.parseInt(indexStrArray[i].trim());  
  
        medikamenteIndex.add(index);  
  
    } catch (NumberFormatException e) {  
  
        System.out.println("Fehler beim Hinzufügen des Patienten");  
  
    }  
}
```

Bevor der neue Patient hinzugefügt wird, prüft der Algorithmus, ob bereits ein Patient mit dem gleichen Namen und Geburtsdatum existiert. Falls ja, wird eine Fehlermeldung angezeigt. Andernfalls wird der neue Patient zu der *patientList* hinzugefügt, die Daten werden serialisiert, die Eingabefelder geleert und die Tabellenansicht aktualisiert.

```

if(patientModel.getPatients().stream().anyMatch(patient -> patient.getName().equals(name)
&& patient.getBirthday().equals(birthday))) {

    Alert alert = new Alert(Alert.AlertType.ERROR);

    alert.setTitle("Fehler");
    alert.setHeaderText(null);
    alert.setContentText("Patient existiert bereits.");
    alert.showAndWait();

    return;
}

patientModel.addPatient(name, birthday, medikamenteIndex);
patientModel.patientSerialize();
PatientenNameTextFeld.clear();
GeburtsdatumFeld.setValue(null);
indexLabel.clear();
updateTableViewPatient();

```

Löschen eines Patienten: Der Algorithmus entfernt den ausgewählten Patienten aus der *patientList* und aktualisiert die Tabellenansicht.

```

public void handlePatientRemove(ActionEvent actionEvent {

    Patient selectedPatient = tabelle.getSelectionModel().getSelectedItem();
    arrayPatientList.remove(selectedPatient);
    PatientModel.getPatients().remove(selectedPatient);
    patientModel.patientSerialize();
    updateTableViewPatient();
}

```

Medikamentenverwaltung: Öffnen des Fensters zum Hinzufügen eines Medikaments (*handleMedikamentHinzufuegenButton* im *MedicationListController*): Dieser Algorithmus ist dafür zuständig, ein neues Fenster zu öffnen, in dem der Benutzer die Daten für ein neues Projektabschlussbericht

Praktikum INF2, Sommersemester 2024

Medikament eingeben kann. Er lädt die FXML-Datei *MedikamentHinzufuegen.fxml*, erstellt eine neue Szene und Bühne, und zeigt das Fenster an.

```
public void handleMedikamentHinzufuegenButton(ActionEvent actionEvent) throws
IOException {

    FXMLLoader loader = new
    FXMLLoader(getClass().getResource("MedikamentHinzufuegen.fxml"));

    Parent MedikamenteHinzufuegen = loader.load();

    Scene scene = new Scene(MedikamenteHinzufuegen);

    Stage stage = new Stage();

    stage.setScene(scene);

    stage.setTitle("Medikament hinzufügen");

    AddMedicationController controller = loader.getController();

    controller.setMedicationListController(this); // Referenz zu diesem Controller
    übergeben

    stage.show();

}
```

Hinzufügen eines Medikaments (*handleSpeicherButton* im *AddMedicationController*): Dieser Algorithmus wird ausgeführt, wenn der Benutzer im Fenster "Medikament hinzufügen" auf den "Speichern"-Button klickt. Er ist ähnlich aufgebaut wie das Hinzufügen eines Patienten. Es werden die Eingaben validiert, auf Duplikate geprüft und das neue Medikament der Liste hinzugefügt. Anschließend werden die Daten serialisiert, die Eingabefelder geleert und das Fenster geschlossen.

```
public void handleSpeicherButton(ActionEvent actionEvent) {

    // ... (Einlesen der Medikamentendaten, Validierung, Duplikatprüfung)

    Medication newMedication = new Medication(medikamentName, index,
    Nebenwirkungen);

    MedicationModel.addMedication(newMedication);

    medicationModel.medicamentSerialize();

    // ... (Leeren der Eingabefelder und Schließen des Fensters)

}
```

Löschen eines Medikaments: Der Algorithmus entfernt das ausgewählte Medikament aus der Liste und aktualisiert die Tabelle.

```
public void handleMedicationRemove(ActionEvent actionEvent) {  
  
    Medication selectedMedication =  
    tabelleMedikamente.getSelectionModel().getSelectedItem();  
  
    medicationList.remove(selectedMedication);  
  
    updateTableViewMedication(); }
```

Erinnerungsmanagement

Erinnerung setzen: Der MainWindowController ist für das Setzen der Erinnerungen verantwortlich. Der Benutzer wählt zunächst einen Patienten aus der Tabelle aus. Anschließend kann er mithilfe eines DatePicker das Datum und über zwei ComboBoxen die Uhrzeit für die Erinnerung auswählen. Die Methode *handleSetReminder* überprüft, ob alle erforderlichen Eingaben gemacht wurden und erstellt dann ein Reminder-Objekt. Dieses Objekt wird an den ReminderManager übergeben, der die Erinnerung plant.

```
public void handleSetReminder(ActionEvent event) {  
  
    // Eingabedaten abrufen und überprüfen  
  
    LocalDate date = datePicker.getValue();  
  
    int hour = Integer.parseInt(hourComboBox.getValue());  
  
    int minute = Integer.parseInt(minuteComboBox.getValue());  
  
    LocalDateTime reminderTime = LocalDateTime.of(date, LocalTime.of(hour,  
minute));  
  
    Patient selectedPatient = tabelle.getSelectionModel().getSelectedItem();  
  
    if (selectedPatient != null) {  
  
        String message = "Erinnerung für " + selectedPatient.getName() + ":";  
        Medikamente einnehmen";  
  
        reminderManager.addReminder(new Reminder(message, reminderTime));  
  
        // Benachrichtigung anzeigen  
  
    } else {  
  
        System.out.println("Kein Patient ausgewählt!");  
  
    }
```

```
}
```

Erinnerung auslösen: Der ReminderManager verwendet einen Timer, um Erinnerungen zu planen. Zum geplanten Zeitpunkt wird eine Benachrichtigung für den Benutzer angezeigt.

```
private void scheduleReminder(Reminder reminder) {  
  
    Timer timer = new Timer();  
  
    long delay = Duration.between(LocalDateTime.now(),  
        reminder.getTime()).toMillis();  
  
    timer.schedule(new TimerTask() {  
  
        @Override  
  
        public void run() {  
  
            Platform.runLater(() -> {  
  
                Alert alert = new Alert(Alert.AlertType.INFORMATION);  
  
                alert.setTitle("Erinnerung");  
  
                alert.setHeaderText(null);  
  
                alert.setContentText(reminder.getMessage());  
  
                alert.showAndWait();  
  
            });  
        }  
  
    }, delay);  
}
```

Login-System

Der zweite Prototyp führt ein einfaches Login-System ein. Es überprüft die eingegebenen Anmeldeinformationen (Benutzername und Passwort) in der Methode `handleSigninButton` des LoginControllers.

```
public void handleSigninButton(ActionEvent event) throws IOException {  
  
    String passwort = passwortTextField.getText();  
  
    String benutzername = benutzerNameTextField.getText();
```

```

if (benutzername.equals("demo") && passwort.equals("1234")) {

    // ... (Hauptfenster öffnen)

} else {

    // ... (Fehlermeldung anzeigen) }

}

```

5.2 Zukünftige Verbesserungen

Das entwickelte MediCare-System bietet eine solide Grundlage für die Medikamentenverwaltung. Es gibt jedoch einige Bereiche, in denen das System in Zukunft weiter verbessert werden könnte:

Passwort-Sicherheit: Die Implementierung einer sicheren Passwortverwaltung ist ein wichtiger Aspekt, der im aktuellen Prototyp noch nicht vollständig umgesetzt wurde. In zukünftigen Versionen sollte die Speicherung von Passwörtern als Hashwerte unter Verwendung eines sicheren Algorithmus erfolgen. Zusätzlich sollte eine Funktion zum Zurücksetzen des Passworts implementiert werden.

Benutzerverwaltung: Die Einführung von Benutzerkonten mit unterschiedlichen Rollen und Berechtigungen würde die Flexibilität und Sicherheit des Systems erhöhen.

Datenbank: Die Umstellung von der Serialisierung auf eine Datenbank (z.B. SQLite) würde die Skalierbarkeit, Effizienz und Zuverlässigkeit der Datenspeicherung verbessern.

Benutzeroberfläche (UI): Die Benutzeroberfläche könnte weiter optimiert werden, um die Benutzerfreundlichkeit und das Erscheinungsbild der Anwendung zu verbessern. Dazu könnte beispielsweise eine Suchfunktion für Patienten und Medikamente gehören.

6 Referenzen

Ilias Fachhochschule Dortmund

Vorlesungen (PDF) Teil 1 – Teil 15 von Prof. Dr.-Ing. Karsten Lehn

Praktika für PR2 von Prof. Dr.-Ing. Karsten Lehn

- Praktikum 1 – Praktikum 5
- ButtonPress.zip
- ButtonPressFXML.zip
- ButtonImage.zip
- LinkedListInt.zip

Zusätzliche Projekte von Dr.-Ing. Karsten Lehn

- Szenenwechsel

UML Diagramm wurde mit <https://plantuml.com/> erstellt.

“MediCare”-Logo

Shopify. (2024). Shopify Logo Maker. <https://www.shopify.com/de/tools/logo-maker>

Stackoverflow

Getting selected item from a JavaFX TableView. Stack Overflow. (2013)

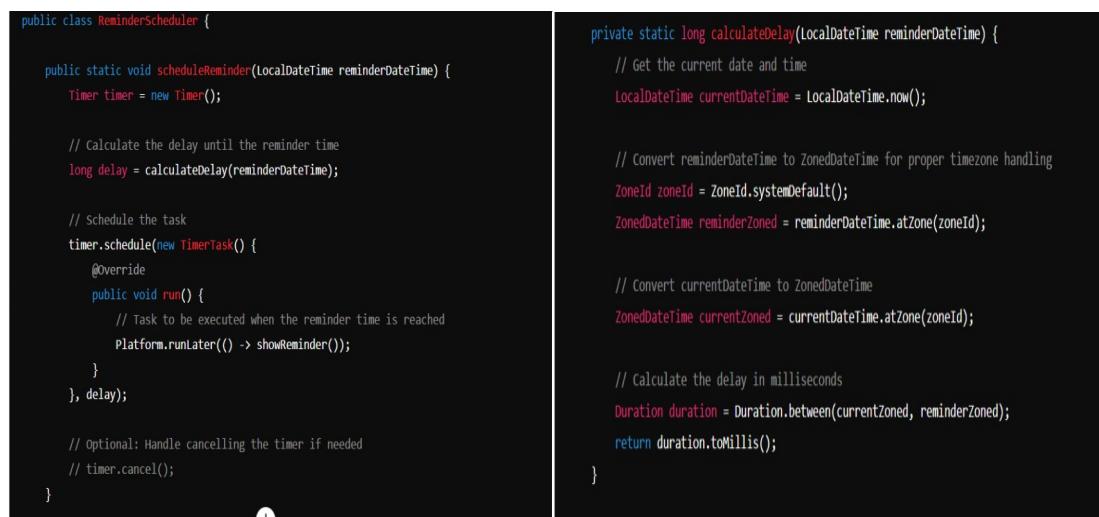
<https://stackoverflow.com/questions/17388866/getting-selected-item-from-a-javafx-tableview>

Prompts: ChatGPT URL: <https://chat.openai.com/>

Reminder-Funktion

Eingabe in ChatGPT: "Hi, I'm working on a Java application that uses JavaFX for the GUI. I need to schedule a reminder to be displayed at a specific time. Can you provide me with a method that calculates the delay until the reminder time and schedules a task to show a reminder window? I want it to be very accurate and also think about time zones"

Dazu zusätzlich: <https://stackoverflow.com/questions/32975392/jsr-310-system-currenttimemillis-vs-instant-toepochmilli-timezone>



```
public class ReminderScheduler {  
  
    public static void scheduleReminder(LocalDateTime reminderDateTime) {  
        Timer timer = new Timer();  
  
        // calculate the delay until the reminder time  
        long delay = calculateDelay(reminderDateTime);  
  
        // Schedule the task  
        timer.schedule(new TimerTask() {  
            @Override  
            public void run() {  
                // Task to be executed when the reminder time is reached  
                Platform.runLater(() -> showReminder());  
            }  
        }, delay);  
  
        // optional: Handle cancelling the timer if needed  
        // timer.cancel();  
    }  
}  
  
private static long calculateDelay(LocalDateTime reminderDateTime) {  
    // Get the current date and time  
    LocalDateTime currentDateTime = LocalDateTime.now();  
  
    // Convert reminderDateTime to ZonedDateTime for proper timezone handling  
    ZoneId zoneId = ZoneId.systemDefault();  
    ZonedDateTime reminderZoned = reminderDateTime.atZone(zoneId);  
  
    // Convert currentDateTime to ZonedDateTime  
    ZonedDateTime currentZoned = currentDateTime.atZone(zoneId);  
  
    // Calculate the delay in milliseconds  
    Duration duration = Duration.between(currentZoned, reminderZoned);  
    return duration.toMillis();  
}
```

Abbildung 9: Ausgabe ChatGPT

Abbildung 10: Ausgabe ChatGPT

SceneManager-Klasse

Eingabe in ChatGPT: "Wie kann man eine zentrale Klasse zur Verwaltung von Szenenübergängen in einer JavaFX-Anwendung umsetzen, um einfach zur vorherigen Szene zurückkehren zu können?"

Ausgabe:

```

public class SceneManager {
    private static SceneManager instance; // Die einzige Instanz dieser Klasse
    private Scene previousScene; // Speichert die vorherige Szene

    // Privater Konstruktor verhindert die Erstellung von mehr als einer Instanz
    private SceneManager() {}

    // Statische Methode, um die einzige Instanz der Klasse zurückzugeben
    public static SceneManager getInstance() {
        if (instance == null) {
            instance = new SceneManager();
        }
        return instance;
    }

    // Methode zum Speichern der vorherigen Szene
    public void setPreviousScene(Scene scene) {
        this.previousScene = scene;
    }

    // Methode zum Abrufen der vorherigen Szene
    public Scene getPreviousScene() {
        return previousScene; ↓
    }
}

```

Abbildung 11: Ausgabe ChatGPT

StyleSheet

Eingabe in ChatGPT: “Give me an example of a StyleSheet”

Ausgabe:

```

/* General Window Styling */
.root {
    -fx-background-color: #84BCDC; /* Light Blue */
    -fx-font-family: "Segoe UI", Tahoma, Geneva, Verdana, sans-serif;
}

/* Button Styling */
.button {
    -fx-background-color: #38596C; /* Darker Blue */
    -fx-text-fill: white;
    -fx-border-radius: 5px;
}

.button:hover { /* Hover effect */
    -fx-background-color: #1565c0; /* Even darker blue on hover */
    -fx-cursor: hand;
}

/* TableView Styling */
.table-view {
    -fx-background-color: white;
    -fx-table-cell-border-color: transparent; /* Remove cell borders */
}

```

Abbildung 12: Ausgabe ChatGPT

```
.table-view .column-header-background { /* Column header background */
    -fx-background-color: black; /* Darker blue */
}

.table-view .column-header .label {
    -fx-text-fill: #1565c0;
    -fx-alignment: center;
}

/* Alternating Row Colors (Optional) */
.table-view .table-row-cell:odd {
    -fx-background-color: whitesmoke;
}

.table-view .table-row-cell:selected {
    -fx-background-color: lightblue; /* Change the background color of selected cells */
    -fx-text-fill: black;           /* Ensure text is visible (optional) */
}

.table-view .table-row-cell:odd:selected {
    -fx-background-color: lightblue; /* Change the background color of odd selected cells */
    -fx-text-fill: black;           /* Ensure text is visible (optional) */
}
```



Abbildung 13: Ausgabe ChatGPT

7 Anhang A – Quellcode

Code aus anderen Quellen

Marie Rose

Assena Satarkulova

```

1 package com.example.medicare_projekt;
2
3 import javafx.application.Application;
4 import javafx.fxml.FXMLLoader;
5 import javafx.scene.Scene;
6 import javafx.scene.layout.AnchorPane;
7 import javafx.stage.Stage;
8
9 import java.io.IOException;
10
11 /**
12  * Die Hauptklasse der Anwendung, die den Startpunkt für
13  * die JavaFX-Anwendung darstellt.
14 */
15
16 /**
17  * Die `start`-Methode wird automatisch aufgerufen,
18  * wenn die JavaFX-Anwendung gestartet wird.
19  *
20  * @param primaryStage Die Hauptbühne der Anwendung.
21  * @throws IOException Wenn ein Fehler beim Laden der
22  * FXML-Datei auftritt.
23 */
24 @Override
25 public void start(Stage primaryStage) throws
26 IOException {
27     FXMLLoader loader = new FXMLLoader(getClass().
28 getResource("LogInView.fxml"));
29     AnchorPane root = loader.load();
30
31     LogInController controller = loader.getController
32     ();
33     controller.setStage(primaryStage);
34
35     Scene scene = new Scene(root);
36     primaryStage.setTitle("Login");
37     primaryStage.setScene(scene);
38     primaryStage.show();
39 }
40
41 /**
42  * Die `main`-Methode ist der Einstiegspunkt für die

```

mithilfe des Projekts „Switch Scene“ von Prof. Dr.-Ing. Karsten Lehn

38 Anwendung.

```
39      * Sie startet die JavaFX-Anwendung.  
40      *  
41      * param args Die Befehlszeilenargumente.  
42      */  
43  public static void main(String[] args) {  
44      launch(args);  
45  }  
46 }  
47
```

```
1 package com.example.medicare_projekt;
2
3 import javafx.application.Platform;
4 import javafx.scene.control.Alert;
5 import java.io.Serializable;
6 import java.time.LocalDate;
7 import java.time.LocalDateTime;
8 import java.time.ZoneId;
9 import java.util.ArrayList;
10 import java.util.Date;
11 import java.util.List;
12 import java.util.Timer;
13 import java.util.TimerTask;
14 /**
15  * Repräsentiert einen Patienten mit Namen, Geburtsdatum,
16  * Index und einer Liste von Erinnerungen.
17 */
18 public class Patient implements Serializable {
19     private static final long serialVersionUID = 1L;
20     private String name;
21     private LocalDate birthday;
22     private ArrayList<Integer> index;
23     private List<Reminder> reminders;
24     /**
25      * Konstruktor für die Patient-Klasse.
26      *
27      * @param name Der Name des Patienten.
28      * @param birthday Das Geburtsdatum des Patienten.
29      * @param index Die Medikamenten-IDs, die dem
30      * Patienten zugeordnet sind.
31      */
32     public Patient(String name, LocalDate birthday,
33                    ArrayList<Integer> index) {
34         this.name = name;
35         this.birthday = birthday;
36         this.index = index;
37         this.reminders = new ArrayList<>();
38     }
39     // ... Getter und Setter für name, birthday, index,
40     reminders
41     public ArrayList<Integer> getIndex() {
42         return index;
43     }
44 }
```

```
File - C:\Users\FHBBook\Desktop\9PG_T4 (2)\9PG_T4\MediCare_Projekt_demo\MediCare_Projekt_demo\src\main\java\com\elb\nursing\Patient.java
41
42     public LocalDate getBirthday() {
43         return birthday;
44     }
45
46     public void setName(String name) {
47         this.name = name;
48     }
49
50     public String getName() {
51         return name;
52     }
53
54     @Override
55     public String toString() {
56         return "Patient: " + name + ", " + birthday + ", "
57         + index;
58     }
59
```

```
1 package com.example.medicare_projekt;
2
3 import java.time.LocalDateTime;
4
5 /**
6  * Stellt eine Erinnerung mit einer Nachricht und einer
7  * Erinnerungszeit dar.
8 */
9 public class Reminder {
10     private String message;
11     private LocalDateTime reminderTime;
12     /**
13      * Konstruktor für die Reminder-Klasse.
14      *
15      * @param message Die Nachricht der Erinnerung.
16      * @param reminderTime Die Zeit, zu der die Erinnerung
17      * angezeigt werden soll.
18     */
19     public Reminder(String message, LocalDateTime
20 reminderTime) {
21         this.message = message;
22         this.reminderTime = reminderTime;
23     }
24
25
26     public String getMessage() {
27         return message;
28     }
29
30     public LocalDateTime getReminderTime() {
31         return reminderTime;
32     }
33 }
```

```
1 package com.example.medicare_projekt;
2
3 import java.io.Serializable;
4 import java.util.ArrayList;
5 /**
6  * Repräsentiert ein Medikament mit Namen, Index und einer
7  * Liste von Nebenwirkungen.
8  * Diese Klasse implementiert Serializable, um die
9  * Speicherung und das Laden von Medikamentenobjekten zu
10 ermöglichen.
11 */
12 public class Medication implements Serializable {
13
14     Patient patient;
15
16     private String medicationName;
17     public String getMedicationName() {
18         return medicationName;
19     }
20
21     public void setMedicationName(String medicationName) {
22         this.medicationName = medicationName;
23     }
24
25     private int index;
26
27     public int getIndex() {
28         return index;
29     }
30
31     public void setIndex(int index) {
32         this.index = index;
33     }
34
35     private ArrayList<String> nebenwirkungen;
36     public ArrayList<String> getNebenwirkungen() {
37         return nebenwirkungen;
38     }
39     public void setNebenwirkungen(ArrayList<String>
40 nebenwirkungen) {
41         this.nebenwirkungen = nebenwirkungen;
42     }
43 }
```

```
File - C:\Users\FHBBook\Desktop\9PG_T4 (2)\9PG_T4\MediCare_Projekt_demo\MediCare_Projekt_demo\src\main\java\com\de
41
42     /**
43      * Konstruktor neues Medikament-Objekts.
44      *
45      * @param medicationName Der Name des Medikaments.
46      * @param index Ein eindeutiger Index (optional) zur Identifikation des Medikaments.
47      * @param nebenwirkungen Eine ArrayList mit den möglichen Nebenwirkungen des Medikaments.
48     */
49     public Medication(String medicationName, int index,
50                        ArrayList<String> nebenwirkungen){
51         this.medicationName = medicationName;
52         this.index = index;
53         this.nebenwirkungen = nebenwirkungen;
54     }
55     /**
56      * Gibt eine String-Darstellung des Medikaments zurück
57      , die Name, Index und Nebenwirkungen enthält.
58     */
59     @Override
60     public String toString(){
61         return "Medikament: " + medicationName + ", Index
62         : " + index + ", Nebenwirkungen: " + nebenwirkungen;
63     }
64
65 }
66
```

```
1 package com.example.medicare_projekt;
2
3 import java.io.*;
4 import java.time.LocalDate;
5 import java.util.ArrayList;
6
7 /**
8  * Verwaltet eine Liste von Patienten und bietet Methoden
9  * zum Laden, Speichern und Hinzufügen von Patienten.
10 */
11
12     // Datei zum Speichern der Patientendaten
13     private static final String patientFile = "patient.ser";
14
15     // Liste der Patienten
16     private static ArrayList<Patient> patientList = new
17     ArrayList<>();
18
19     /**
20      * Gibt die Liste der Patienten zurück.
21      *
22      * @return Eine ArrayList mit den Patienten.
23     */
24     public static ArrayList<Patient> getPatients() {
25         return patientList;
26     }
27
28     /**
29      * Serialisiert die Liste der Patienten in eine Datei.
30      * Speichert die aktuelle Liste der Patienten in der
31      * Datei 'patient.ser'.
32      */
33     public void patientSerialize() {
34         try (FileOutputStream fileOut = new
35             FileOutputStream(patientFile);
36             ObjectOutputStream out = new
37             ObjectOutputStream(fileOut)) {
38             out.writeObject(patientList);
39             System.out.println("Serialized Patient in List
40             !");
41         } catch (IOException i) {
42             i.printStackTrace();
43         }
44     }
45 }
```

```
File - C:\Users\FHBBook\Desktop\9PG_T4 (2)\9PG_T4\MediCare_Projekt_demo\MediCare_Projekt_demo\src\main\java\com\le
38
39
40
41     /**
42      * Fügt einen neuen Patienten zur Liste hinzu.
43      *
44      * @param name Der Name des Patienten.
45      * @param birthday Das Geburtsdatum des Patienten.
46      * @param index Die Medikamenten-IDs, die dem
Patienten zugeordnet sind.
47     */
48     public static void addPatient(String name, LocalDate
birthday, ArrayList<Integer> index) {
49         patientList.add(new Patient(name, birthday, index
));
50         System.out.println(patientList);
51     }
52
53     /**
54      * Lädt die Patientendaten aus der angegebenen Datei.
55      * Liest die gespeicherte Liste der Patienten aus der
Datei 'patient.ser' und lädt sie in die Anwendung.
56     */
57     public void loadDataFromFile() {
58         File file = new File(patientFile);
59         if (file.exists()) {
60             try (ObjectInputStream in = new
ObjectInputStream(new FileInputStream(file))) {
61                 patientList = (ArrayList<Patient>) in.
readObject();
62                 System.out.println("Daten geladen aus " +
patientFile);
63             } catch (IOException | ClassNotFoundException
e) {
64                 e.printStackTrace();
65             }
66         } else {
67             System.out.println("Datei nicht gefunden: " +
patientFile);
68         }
69     }
70 }
```

Mithilfe Vorlesung „LE09 - Dynamische Datenstrukturen: Listen“

```
1 package com.example.medicare_projekt;
2
3 import javafx.scene.Scene;
4 /**
5  * Diese Klasse verwaltet die Szenenwechsel in der
6  * Anwendung.
7  * 
8  * SceneManager implementiert das Singleton-Muster, um
9  * sicherzustellen, dass nur eine einzige Instanz
10 * dieser Klasse existiert. Sie ermöglicht das Speichern
11 * und Abrufen der vorherigen Szene, was für die
12 * Navigation innerhalb der Anwendung nützlich ist (z.B.
13 * um nach dem Anzeigen eines Details zu einer
14 * vorherigen Liste zurückzukehren).
15 */
16 public class SceneManager {mithilfe von Gemini
17     /**
18      * Die einzige Instanz dieser Klasse (Singleton-Muster
19      *).
20      */
21     private static SceneManager instance;
22
23     /**
24      * Privater Konstruktor, um die Instanziierung von
25      * außen zu verhindern (Singleton-Muster).
26      */
27     private SceneManager() {}
28
29     /**
30      * Gibt die einzige Instanz des SceneManagers zurück (
31      * Singleton-Muster).
32      * Wenn noch keine Instanz existiert, wird eine neue
33      * erstellt und zurückgegeben.
34      */
35     public static SceneManager getInstance() {
36         if (instance == null) {
37             instance = new SceneManager();
38         }
39         return instance;
40     }
41 }
```

```
File - C:\Users\FHBBook\Desktop\9PG_T4 (2)\9PG_T4\MediCare_Projekt_demo\MediCare_Projekt_demo\src\main\java\com\le
36         }
37         return instance;
38     }
39     /**
40      * Setzt die vorherige Szene.
41      * Diese Methode wird aufgerufen, bevor zu einer neuen
42      * Szene gewechselt wird,
43      * um die aktuelle Szene als "vorherige Szene" zu
44      * speichern.
45      *
46      * @param scene Die aktuelle Szene, die als "vorherige
47      * Szene" gespeichert werden soll.
48     */
49     public void setPreviousScene(Scene scene) {
50         this.previousScene = scene;
51     }
52     /**
53      * Gibt die vorherige Szene zurück.
54      * Diese Methode kann verwendet werden, um zur
55      * vorherigen Szene zurückzukehren.
56      *
57      * @return Die vorherige Szene, oder null, wenn es
58      * keine vorherige Szene gibt.
59      */
60     public Scene getPreviousScene() {
61         return previousScene;
62     }
63 }
```

```
1 package com.example.medicare_projekt;
2
3 import javafx.event.ActionEvent;
4 import javafx.fxml.FXML;
5 import javafx.fxml.FXMLLoader;
6 import javafx.scene.Node;
7 import javafx.scene.Parent;
8 import javafx.scene.Scene;
9 import javafx.scene.control.Alert;
10 import javafx.scene.control.Button;
11 import javafx.scene.control.TextField;
12 import javafx.scene.image.Image;
13 import javafx.scene.image.ImageView;
14 import javafx.stage.Stage;
15
16 import java.io.IOException;
17 /**
18 * Der Controller für den Login-View.
19 * Er ist verantwortlich für die Verarbeitung der
20 * Benutzereingaben und die Navigation zum Hauptfenster.
21 */
22
23     // ... FXML Elemente
24
25     @FXML
26     private TextField benutzerNameTextField;
27
28     @FXML
29     private TextField passwortTextField;
30
31     @FXML
32     private Button signInButton;
33
34     @FXML
35     private ImageView imageView;
36
37     private Stage stage;
38
39     private Scene scene;
40
41     private String passwort = "1234";
42
43     private String benutzername = "demo";
```

```

44     /**
45      * Setzt die Stage für den Controller.
46      *
47      * @param stage Die Stage, die dem Controller
48      * zugeordnet werden soll.
49      */
50     public void setStage(Stage stage) {
51         this.stage = stage;
52     }
53
54     public void setMainScene(Scene hauptseiteScene) {
55         this.scene = scene;
56     }
57
58     /**
59      * Behandelt das Klicken auf den "Sign in"-Button.
60      * Überprüft die eingegebenen Anmeldeinformationen und
61      * öffnet das Hauptfenster bei Erfolg.
62      *
63      * @param event Das ActionEvent, das ausgelöst wurde.
64      * @throws IOException Wenn ein Fehler beim Laden der
65      * FXML-Datei für das Hauptfenster auftritt.
66      */
67     public void handleSigninButton(ActionEvent event)
68     throws IOException {
69         // ... (Logik zur Überprüfung von Benutzername und
70         // Passwort, dann Öffnen des Hauptfensters)
71
72         String passwortEingegeben = passwortTextField.
73             getText();
74
75         String benutzernameEingegeben =
76             benutzerNameTextField.getText();
77
78         if (benutzernameEingegeben.equals(benutzername)
79             && passwortEingegeben.equals(passwort)) {
80             Stage loginStage =
81                 (Stage) ((Node) event.getSource()).
82                 getScene().getWindow();
83             loginStage.close();
84
85             FXMLLoader loader = new FXMLLoader(getClass()).
86                 getResource("HauptFenster.fxml"));
87             Parent hauptFenster = loader.load();
88             Scene scene = new Scene(hauptFenster);

```

```
File - C:\Users\FHBBook\Desktop\9PG_T4 (2)\9PG_T4\MediCare_Projekt_demo\MediCare_Projekt_demo\src\main\java\com\le
78         Stage mainStage = new Stage();
79         mainStage.setScene(scene);
80         mainStage.setTitle("MediCare");
81         mainStage.show();
82
83         MainWindowController controller = loader.
84             getController();
85             controller.setStage(mainStage);
86             controller.setMainScene(scene);
87         } else {
88             Alert alert = new Alert(Alert.AlertType.ERROR
89             );
90             alert.setTitle("Login Fehler");
91             alert.setHeaderText(null);
92             alert.setContentText("Falscher Benutzername
93             oder Passwort!");
94         }
95 }
```

```
1 package com.example.medicare_projekt;
2
3 import java.io.*;
4 import java.util.ArrayList;
5
6 /**
7  * Diese Klasse repräsentiert ein Modell für die
8  * Verwaltung von Medikamentendaten.
9  * Sie lädt Medikamente aus einer Datei, ermöglicht das
10 * Hinzufügen neuer Medikamente
11 * und speichert die Daten persistent.
12 */
13
14 /**
15  * Der Dateiname, unter dem die Medikamentendaten
16  * serialisiert werden.
17 */
18
19 /**
20  * Die Liste aller Medikamente.
21 */
22
23 /**
24  * Gibt die Liste aller Medikamente zurück.
25 */
26 /**
27  * @return Die Liste der Medikamente.
28 */
29
30 /**
31  * Konstruktor: Lädt Medikamentendaten aus der Datei
32  * beim Erstellen des Modells.
33 */
34
35 /**
36  * @return MedicationModel()
37  * loadFromFile(); // Daten werden beim
38  * Erstellen des Modells geladen
39 */
```

```

38
39     /**
40      * Serialisiert die Liste der Medikamente in eine
41      * Datei.
42      * Diese Methode speichert die aktuellen
43      * Medikamentendaten persistent.
44      */
45
46     public static void medicationSerialize() {
47         try (FileOutputStream fileOut = new
48             FileOutputStream(medicationFile);
49             ObjectOutputStream out = new
50             ObjectOutputStream(fileOut)) {
51             out.writeObject(medicationList);
52             System.out.println("Serialized Medication in
53             List!"); // Erfolgsmeldung (optional)
54
55         } catch (IOException i) {
56             i.printStackTrace();
57         }
58     }
59
60     /**
61      * Fügt ein neues Medikament zur Liste hinzu und
62      * serialisiert die Liste.
63      *
64      * @param medication Das hinzuzufügende Medikament.
65      */
66
67     public static void addMedication(Medication medication
68 ) {
69     medicationList.add(medication);
70
71     System.out.println(medicationList); // Ausgabe zur
72     Kontrolle
73     medicationSerialize(); // Speichern der
74     aktualisierten Daten
75 }
76
77 /**
78  * Lädt die Medikamentendaten aus der Datei.
79  * Falls die Datei nicht existiert oder beim Lesen
80  * Fehler auftreten, wird eine leere Liste verwendet.
81  */
82
83     public void loadDataFromFile() {

```

Mithilfe Vorlesung „LE09 -
Dynamische Datenstrukturen:
Listen“

```
File - C:\Users\FHBBook\Desktop\9PG_T4 (2)\9PG_T4\MediCare_Projekt_demo\MediCare_Projekt_demo\src\main\java\com\elbsoft\MediCare\MedicationList.java
72     try (ObjectInputStream in = new ObjectInputStream
73          (new FileInputStream(medicationFile))) {
74         medicationList = (ArrayList<Medication>) in.
75             readObject();
76     } catch (IOException | ClassNotFoundException e) {
77     }
78 }
79 }
```

Mithilfe Vorlesung „LE09 -
Dynamische Datenstrukturen:
Listen“

```
1 package com.example.medicare_projekt;
2
3
4 import java.time.LocalDateTime;
5 import java.util.ArrayList;
6 import java.util.List;
7 import java.util.Timer;
8 import java.util.TimerTask;
9
10 import javafx.application.Platform;
11 import javafx.scene.control.Alert;
12
13 /**
14  * Verwaltet Erinnerungen und zeigt sie zu den geplanten
15  * Zeiten an.
16 */
17 public class ReminderManager {
18     private List<Reminder> reminders = new ArrayList<>();
19     private Timer timer = new Timer(true);
20
21     /**
22      * Fügt eine Erinnerung hinzu und plant sie.
23      *
24      * @param reminder Die hinzuzufügende Erinnerung.
25      */
26     public void addReminder(Reminder reminder) {
27         reminders.add(reminder);
28         scheduleReminder(reminder);
29     }
30
31     /**
32      * Plant die Ausführung einer Erinnerung.
33      *
34      * @param reminder Die zu planende Erinnerung.
35      */
36     private void scheduleReminder(Reminder reminder) {
37         long delay = reminder.getReminderTime().atZone(
38             java.time.ZoneId.systemDefault()).toInstant().toEpochMilli(
39             ) - System.currentTimeMillis();
40
41         // Planung der Erinnerung mit dem Timer
42         timer.schedule(new TimerTask() {
43             @Override
44             public void run() {
45                 // Ausführung im JavaFX-Thread, um das
46                 // Erinnerungsfenster anzuzeigen
47             }
48         }, delay);
49     }
50 }
```

```
File - C:\Users\FHBBook\Desktop\9PG_T4 (2)\9PG_T4\MediCare_Projekt_demo\MediCare_Projekt_demo\src\main\java\com\el
41          Platform.runLater(() -> showReminderWindow
42          (reminder.getMessage(), reminder.getReminderTime()));
43      }
44  }
45  /**
46   * Zeigt ein Erinnerungsfenster mit der angegebenen
47   * Nachricht und Erinnerungszeit an.
48   *
49   * @param message Die Nachricht, die im
50   * Erinnerungsfenster angezeigt werden soll.
51   * @param reminderTime Die geplante Erinnerungszeit.
52   */
53   private void showReminderWindow(String message,
54   LocalDateTime reminderTime) {
55       Alert alert = new Alert(Alert.AlertType.
56       INFORMATION);
57       alert.setTitle("Medikamenten-Erinnerung");
58       alert.setHeaderText(null);
59       alert.setContentText(message + " um " +
60       reminderTime);
61       alert.showAndWait();
62   }
63 }
```

```
1 package com.example.medicare_projekt;
2
3 import javafx.event.ActionEvent;
4 import javafx.fxml.FXML;
5 import javafx.fxml.FXMLLoader;
6 import javafx.scene.Node;
7 import javafx.scene.Parent;
8 import javafx.scene.Scene;
9 import javafx.scene.control.*;
10 import javafx.scene.control.cell.PropertyValueFactory;
11 import javafx.scene.input.KeyEvent;
12 import javafx.scene.input.MouseEvent;
13 import javafx.scene.layout.AnchorPane;
14 import javafx.stage.Stage;
15 import java.io.IOException;
16 import java.time.LocalDate;
17 import java.util.ArrayList;
18 import java.time.LocalDateTime;
19 import java.time.LocalTime;
20 /**
21  * Hauptcontroller für die Anwendung.
22  * Dieser Controller verwaltet die Hauptansicht,
23  * einschließlich der Anzeige von Patienten in
24  * einer Tabelle, dem Hinzufügen neuer Patienten, dem
25  * Setzen von Erinnerungen für Patienten,
26  * dem Anzeigen der Medikamentenliste und dem Entfernen
27  * von Patienten.
28 */
29 public class MainWindowController {
30     private Stage stage;
31     private Scene hauptScene;
32     PatientModel patientModel;
33     MedicationListController medicationListController;
34
35     private Scene scene;
36
37     public void setScene(Scene scene) {
38         this.scene = scene;
39     }
40
41     public void setStage(Stage stage) {
42         this.stage = stage;
43     }
44 }
```

```
42     private Main mainApp;  
43  
44     public void setMainApp(Main mainApp) {  
45         this.mainApp = mainApp;  
46     }  
47  
48     @FXML  
49     private Button addButton;  
50  
51     @FXML  
52     private Button viewMedicationListButton;  
53  
54     @FXML  
55     private TextField PatientenNameTextFeld;  
56  
57     @FXML  
58     private DatePicker GeburtsdatumFeld;  
59  
60     @FXML  
61     private TextField indexLabel;  
62  
63     @FXML  
64     private AnchorPane hauptFenster;  
65  
66     @FXML  
67     private TableView<Patient> tabelle;  
68  
69     @FXML  
70     private TableColumn<Patient, String> tabelleName;  
71  
72     @FXML  
73     private TableColumn<Patient, LocalDate>  
    tabelleGeburtsdatum;  
74  
75     @FXML  
76     private TableColumn<Patient, Integer> tabelleIndex;  
77  
78     private ArrayList<Patient> arrayPatientList;  
79  
80     @FXML  
81     private DatePicker datePicker;  
82  
83     @FXML  
84     private ComboBox<Integer> hourComboBox;
```

```
85
86     @FXML
87     private ComboBox<Integer> minuteComboBox;
88
89     @FXML
90     private Label reminderLabel;
91
92     private ReminderManager reminderManager = new
93         ReminderManager();
94
95     /**
96      * Initialisiert den Controller.
97      * Lädt die Patientendaten aus der Datei,
98      * initialisiert das Modell, ruft die
99      * Patientenliste ab, befüllt die Tabellenansicht und
100     * initialisiert die ComboBoxen
101     * für die Erinnerungsfunktion.
102     */
103
104     public void initialize() {
105         patientModel = new PatientModel();
106         patientModel.loadDataFromFile();
107         arrayPatientList = PatientModel.getPatients();
108
109         if (tabelleName != null) {
110             tabelleName.setCellValueFactory(new
111                 PropertyValueFactory<>("name"));
112         }
113         if (tabelleGeburtsdatum != null) {
114             tabelleGeburtsdatum.setCellValueFactory(new
115                 PropertyValueFactory<>("birthday"));
116         }
117         if (tabelleIndex != null) {
118             tabelleIndex.setCellValueFactory(new
119                 PropertyValueFactory<>("index"));
120         }
121         updateTableViewPatient();
122     }
123
124     for (int i = 0; i < 24; i++) {
125         hourComboBox.getItems().add(i);
126     }
127     for (int i = 0; i < 60; i += 1) {
128         minuteComboBox.getItems().add(i);
129     }
130 }
```

```

File - C:\Users\FHBBook\Desktop\9PG_T4 (2)\9PG_T4\MediCare_Projekt_demo\MediCare_Projekt_demo\src\main\java\com\le
123    }
124    /**
125     * Behandelt das Klicken des "Patient hinzufügen"-Buttons.
126     * Liest die Eingaben aus den Textfeldern, validiert sie, erstellt einen neuen
127     * Patienten und fügt ihn zur Liste hinzu. Speichert die Patientenliste und aktualisiert
128     * die Tabellenansicht.
129     *
130     * @param event Das auslösende ActionEvent.
131     */
132     @FXML
133     private void handlePatientHinzufuegenButton(
134         ActionEvent event) {
135         String name = PatientenNameTextFeld.getText();
136         LocalDate birthday = GeburtsdatumFeld.getValue();
137         String indexString = indexLabel.getText();
138         if(indexString == null || indexString.isEmpty()
139             ()) {
140             Alert alert = new Alert(Alert.AlertType.ERROR
141             );
142             alert.setTitle("Fehler");
143             alert.setHeaderText(null);
144             alert.setContentText("Bitte geben Sie einen
145             Index ein.");
146             alert.showAndWait();
147             return;
148         }
149         if (name != null && !name.isEmpty() && birthday
150             != null && indexString != null) {
151             ArrayList<Integer> medikamenteIndex = new
152             ArrayList<>();
153             String[] indexStrArray = indexString.split(
154                 ",");
155             for (int i = 0; i < indexStrArray.length; i
156                 ++) {
157                 try {
158                     int index = Integer.parseInt(
159                         indexStrArray[i].trim());
160                     medikamenteIndex.add(index);
161                 } catch (NumberFormatException e) {
162                     System.out.println("Fehler beim

```

```

File - C:\Users\FHBBook\Desktop\9PG_T4 (2)\9PG_T4\MediCare_Projekt_demo\MediCare_Projekt_demo\src\main\java\com\le
154 Hinzufügen des Patienten");
155     }
156 }
157
158     if(patientModel.getPatients().stream().
159     anyMatch(patient -> patient.getName().equals(name) &&
160     patient.getBirthday().equals(birthday))) {
161         Alert alert = new Alert(Alert.AlertType.
162         ERROR);
163         alert.setTitle("Fehler");
164         alert.setHeaderText(null);
165         alert.setContentText("Patient existiert
166         bereits.");
167         alert.showAndWait();
168     }
169
170     patientModel.addPatient(name, birthday,
171     medikamenteIndex);
172     patientModel.patientSerialize();
173     PatientenNameTextFeld.clear();
174     GeburtsdatumFeld.setValue(null);
175     indexLabel.clear();
176     updateTableViewPatient();
177 }
178
179 /**
180  * Setzt eine Erinnerung für den ausgewählten
181  * Patienten.
182  * Liest das Datum und die Uhrzeit aus den
183  * entsprechenden Bedienelementen, erstellt eine Erinnerung
184  * und fügt sie dem Erinnerungsmanager hinzu. Zeigt
185  * eine Bestätigung oder Fehlermeldung an.
186  *
187  * @param event Das auslösende ActionEvent.
188  */
189
190 @FXML
191 private void handleSetReminder(ActionEvent event) {
192     LocalDate date = datePicker.getValue();
193     Integer hour = hourComboBox.getValue();
194     Integer minute = minuteComboBox.getValue();
195
196     if (date == null || hour == null || minute ==
197     null) {

```

```

189         Alert errorAlert = new Alert(Alert.AlertType.
190             ERROR);
191         errorAlert.setTitle("Fehler");
192         errorAlert.setHeaderText(null);
193         errorAlert.setContentText("Bitte wählen Sie
194             einen Patienten, Datum, Stunde und Minute.");
195         errorAlert.showAndWait();
196         return;
197     }
198     LocalDateTime reminderTime = LocalDateTime.of(
199         date, LocalTime.of(hour, minute));
200     Patient selectedPatient = tabelle.
201     getSelectionModel().getSelectedItem(); mithilfe stackoverflow.com
202     if (selectedPatient != null) {
203         String message = "Erinnerung für " +
204             selectedPatient.getName() + ": Medikamente einnehmen";
205         reminderManager.addReminder(new Reminder(
206             message, reminderTime));
207         Alert alert = new Alert(Alert.AlertType.
208             INFORMATION);
209         alert.setTitle("Erinnerung");
210         alert.setHeaderText(null);
211         alert.setContentText("Erinnerung gesetzt für
212             : " + selectedPatient + ", um: " + reminderTime);
213         alert.showAndWait();
214         datePicker.setValue(null);
215         hourComboBox.setValue(null);
216         minuteComboBox.setValue(null);
217     } else {
218         Alert errorAlert = new Alert(Alert.AlertType.
219             ERROR);
220         errorAlert.setTitle("Fehler");
221         errorAlert.setHeaderText(null);
222         errorAlert.setContentText("Bitte wählen Sie
223             einen Patienten, Datum, Stunde und Minute.");
224         errorAlert.showAndWait();
225     }

```

```

223     }
224     /**
225      * Aktualisiert die Tabellenansicht der Patienten.
226      * Leert die Tabelle, fügt die aktuellen
227      * Patientendaten hinzu und gibt eine Meldung aus,
228      * falls nicht alle Felder ausgefüllt sind.
229     */
230     @FXML
231     private void updateTableViewPatient() {
232         if (tabellenName != null && tabelleGeburtsdatum
233             != null && tabelleIndex != null) {
234             tabelle.getItems().clear();
235             tabelle.getItems().addAll(PatientModel.
236                 getPatients());
237         }
238         else {
239             System.out.println("Alle Felder müssen
240             ausgefüllt sein");
241         }
242     /**
243      * Öffnet die Medikamentenliste in einem neuen
244      * Fenster.
245      *
246      * param actionEvent Das auslösende ActionEvent.
247      * throws IOException Wenn ein Fehler beim Laden der
248      * FXML-Datei auftritt.
249      */
250     public void handleMedikamentenListeAnsehen(
251         ActionEvent actionEvent) throws IOException {
252         FXMLLoader loader = new FXMLLoader(getClass().
253             getResource("MedikamentAnsicht.fxml"));
254         Parent medikamentenAnsicht = loader.load();
255         Scene scene = new Scene(medikamentenAnsicht);
256         SceneManager.getInstance().setPreviousScene(((Node) actionEvent.getSource()).getScene());
257         Stage medicationStage = new Stage();
258         medicationStage.setScene(scene);
259         medicationStage.setTitle("Medikamentenliste");
260         MedicationListController controller = loader.
261             getController();

```

mithilfe des Projekts
„Switch Scene“ von
Prof. Dr.-Ing. Karsten
Lehn

```

File - C:\Users\FHBBook\Desktop\9PG_T4 (2)\9PG_T4\MediCare_Projekt_demo\MediCare_Projekt_demo\src\main\java\com\le
257     controller.setStage(stage);
258     controller.setPreviousScene(SceneManager.
259         getInstance().getPreviousScene());
260     double mainWindowX = stage.getX();
261     double mainWindowY = stage.getY();
262     double mainStageWidth = stage.getWidth();
263
264     double medicationStageX = mainWindowX +
265         mainStageWidth + 10;
266     double medicationStageY = mainWindowY;
267     medicationStage.setX(medicationStageX);
268     medicationStage.setY(medicationStageY);
269
270     medicationStage.show();
271 }
272
273     public void setMainScene(Scene hauptseiteScene) {
274         this.scene = scene;
275     }
276
277
278     public void handleMedikamenteReminder(MouseEvent
279         mouseEvent) {
280         Patient selectedPatient = (Patient) tabelle.
281             getSelectionModel().getSelectedItem(); mithilfe von stackoverflow.com
282     }
283
284     /**
285      * Entfernt den ausgewählten Patienten aus der Liste
286      * und speichert die Änderungen.
287      *
288      * param actionEvent Das auslösende ActionEvent.
289      */
290
291     public void handlePatientRemove(ActionEvent
292         actionEvent) {
293         Patient selectedPatient = (Patient) tabelle.
294             getSelectionModel().getSelectedItem();
295         arrayPatientList.remove(selectedPatient);
296         PatientModel.getPatients().remove(selectedPatient
297     );
298         patientModel.patientSerialize();
299         updateTableViewPatient();

```

293 }

294 }

295

```
1 package com.example.medicare_projekt;
2
3 import javafx.event.ActionEvent;
4 import javafx.fxml.FXML;
5 import javafx.scene.control.Alert;
6 import javafx.scene.control.Button;
7 import javafx.scene.control.TextField;
8 import javafx.stage.Stage;
9
10 import java.util.ArrayList;
11 import java.util.Arrays;
12
13 /**
14  * Controller für das Hinzufügen neuer Medikamente.
15  * Diese Klasse verwaltet die Benutzereingabe in den
16  * Textfeldern, validiert die Eingaben,
17  * erstellt ein neues Medikament-Objekt und fügt es zum
18  * Medikamentenmodell hinzu.
19 */
20
21 public class AddMedicationController {
22
23     // Das Medikamentenmodell zum Speichern und Laden der
24     // Medikamente
25     private MedicationModel medicationModel = new
26     MedicationModel();
27
28     // Der Controller der Medikamentenliste (wird
29     // verwendet, um die Liste nach dem Hinzufügen zu
30     // aktualisieren)
31     private MedicationListController
32     medicationListController;
33
34     /**
35      * Initialisiert den Controller.
36      * Lädt die Medikamentendaten aus der Datei.
37      */
```

```

File - C:\Users\FHBBook\Desktop\9PG_T4 (2)\9PG_T4\MediCare_Projekt_demo\MediCare_Projekt_demo\src\main\java\com\le
38     public void initialize() {
39         medicationModel = new MedicationModel(); // Modell
40         medicationModel.loadDataFromFile(); // Daten
41     }
42
43     /**
44      * Behandelt das Klicken des Speicher-Buttons.
45      * Liest die Eingaben aus den Textfeldern, validiert
46      * sie, erstellt ein neues
47      * Medikament-Objekt und fügt es zum Modell hinzu.
48      *
49      * @param actionEvent Das auslösende Ereignis.
50     */
50     public void handleSpeicherButton(ActionEvent
51     actionEvent) {
51
52         // Einlesen der Medikamentendaten aus den
53         // Textfeldern
53         String medikamentName = medikamentTextFeld.getText
54         ();
54         String indexString = indexTextFeld.getText();
55         String nebenwirkungenText = nebenwirkungenTextFeld
56         .getText();
56         String[] nebenwirkungenArray = nebenwirkungenText.
57         split(",");
57         ArrayList<String> nebenwirkungen = new ArrayList
58         <>(Arrays.asList(nebenwirkungenArray));
58
59         // Trimmen der Leerzeichen aus den Nebenwirkungen
60         for (int i = 0; i < nebenwirkungen.size(); i++) {
61             nebenwirkungen.set(i, nebenwirkungen.get(i).
62             trim());
62         }
63
64         // Validierung der Eingaben
65         if (medikamentName.isEmpty() || indexString.
66         isEmpty()) {
66             Alert alert = new Alert(Alert.AlertType.ERROR
67             );
67             alert.setTitle("Fehler");
68             alert.setContentText("Bitte füllen Sie alle
68             Felder aus.");

```

```

File - C:\Users\FHBBBook\Desktop\9PG_T4 (2)\9PG_T4\MediCare_Projekt_demo\MediCare_Projekt_demo\src\main\java\com\le
69                     alert.showAndWait();
70                     return;
71                 }
72
73             // Konvertierung des Index und Überprüfung auf
74             // doppelte Medikamente
75             try {
76                 int index = Integer.parseInt(indexString);
77
78                 if(medicationModel.getMedication().stream().
79                     anyMatch(medication -> medication.getMedicationName().
80                         equals(medikamentName) && medication.getIndex() == index
81                     )) {
82                     Alert alert2 = new Alert(Alert.AlertType.
83                         ERROR);
84                     alert2.setTitle("Fehler");
85                     alert2.setContentText("Medikament
86                     existiert bereits.");
87                     alert2.showAndWait();
88                     return;
89                 }
90
91             // Erstellen des Medikament-Objekts und
92             // Hinzufügen zum Modell
93             Medication newMedication = new Medication(
94                 medikamentName, index, nebenwirkungen);
95             medicationModel.addMedication(newMedication);
96
97             // Aktualisieren der Medikamentenliste und
98             // Schließen des Fensters
99             medicationListController.
100                refreshMedicationList(); // Liste aktualisieren
101                medikamentTextFeld.clear();
102                indexTextFeld.clear();
103                nebenwirkungenTextFeld.clear();
104
105                Stage stage = (Stage) speicherButton.getScene
106                  ().getWindow();
107                stage.close();
108
109             } catch (NumberFormatException e) {
110                 // Fehlerbehandlung für ungültigen Index
111                 Alert alert = new Alert(Alert.AlertType.ERROR
112
113             );

```

```
File - C:\Users\FHBBook\Desktop\9PG_T4 (2)\9PG_T4\MediCare_Projekt_demo\MediCare_Projekt_demo\src\main\java\com\de\hhu\medicare\MedicationListController.java
101         alert.setTitle("Fehler");
102         alert.setContentText("Bitte geben Sie eine
103             gültige Zahl für den Index ein.");
104         alert.showAndWait();
105     }
106
107     /**
108      * Setzt den Controller der Medikamentenliste.
109      *
110      * @param controller Der Controller der
111      * Medikamentenliste.
112      */
113     public void setMedicationListController(
114         MedicationListController controller) {
115         this.medicationListController = controller;
116     }
}
```

File - C:\Users\FHBBook\Desktop\9PG_T4 (2)\9PG_T4\MediCare_Projekt_demo\MediCare_Projekt_demo\src\main\java\com\el

```
1 package com.example.medicare_projekt;
2
3 import javafx.event.ActionEvent;
4 import javafx.fxml.FXML;
5 import javafx.fxml.FXMLLoader;
6 import javafx.scene.Parent;
7 import javafx.scene.Scene;
8 import javafx.scene.control.*;
9 import javafx.scene.control.cell.PropertyValueFactory;
10 import javafx.stage.Stage;
11
12 import java.io.IOException;
13 import java.util.ArrayList;
14
15 /**
16  * Controller für die Anzeige und Verwaltung einer Liste
17  * von Medikamenten.
18  *
19  * Dieser Controller ermöglicht das Anzeigen der
20  * Medikamente in einer Tabelle, das Hinzufügen
21  * neuer Medikamente über ein separates Fenster, das
22  * Aktualisieren der Liste und das Löschen
23  * von Medikamenten.
24  */
25
26 public class MedicationListController {
27
28     // Das Medikamentenmodell, das die Daten hält
29     private MedicationModel medicationModel;
30
31     // Liste der anzuzeigenden Medikamente
32     private ArrayList<Medication> medicationArrayList;
33
34     @FXML
35     private Button zurueckButton;
36
37     @FXML
38     private TableColumn<Medication, Integer>
39     tabelleMedikamentenIndex;
40
41     @FXML
42     private TableColumn<Medication, String>
43     tabelleMedikament;
44
45     @FXML
46     private TableColumn<Medication, String>
```

```

File - C:\Users\FHBBook\Desktop\9PG_T4 (2)\9PG_T4\MediCare_Projekt_demo\MediCare_Projekt_demo\src\main\java\com\le
39 tabelleNebenwirkungen;
40 @FXML
41 private TableView<Medication> tabelleMedikamente;
42
43 private Scene previousScene;
44 private Stage stage;
45
46 public void setStage(Stage stage) { this.stage = stage
47 ; }
47 public void setPreviousScene(Scene scene) { this.
48 previousScene = scene; }
48
49 /**
50      * Initialisiert den Controller.
51      *
52      * Lädt die Medikamentendaten aus der Datei,
53      * initialisiert das Modell, ruft die
54      * Medikamentenliste ab und aktualisiert die
55      * Tabellenansicht.
56      */
55 @FXML
56 public void initialize() {
57     medicationModel = new MedicationModel();
58     medicationModel.loadDataFromFile();
59     medicationArrayList = medicationModel.
60         getMedication();
61         updateTableView();
61 }
62
63 /**
64      * Behandelt das Klicken des "Medikament hinzufügen"-
65      * Buttons.
66      *
67      * Öffnet ein neues Fenster zum Hinzufügen eines
68      * Medikaments.
69      *
70      * param actionEvent Das auslösende Ereignis.
71      * throws IOException Wenn ein Fehler beim Laden der
72      * FXML-Datei auftritt.
73      */
71 public void handleMedikamentHinzufuegenButton(
72     ActionEvent actionEvent) throws IOException {
72     FXMLLoader loader = new FXMLLoader(getClass().
73         getResource("MedikamentHinzufuegen.fxml"));

```

```

73         Parent MedikamenteHinzufuegen = loader.load();
74         Scene scene = new Scene(MedikamenteHinzufuegen);
75         Stage stage = new Stage();
76         stage.setScene(scene);
77         stage.setTitle("Medikament hinzufügen");
78
79         AddMedicationController controller = loader.
80             getController();
80             controller.setMedicationListController(this); // Referenz zu diesem Controller übergeben
81             stage.show();
82     }
83
84     /**
85      * Aktualisiert die Tabellenansicht mit den aktuellen
86      * Medikamentendaten.
87      * Leert die Tabelle, füllt sie mit den Daten aus der
88      * Medikamentenliste und
89      * verknüpft die Spalten mit den entsprechenden
89      * Eigenschaften der Medikamentenobjekte.
90      */
91     private void updateTableView() {
92         tabelleMedikamente.getItems().clear();
93         tabelleMedikamente.getItems().addAll(
94             medicationArrayList);
95
96         tabelleMedikament.setCellValueFactory(new
97             PropertyValueFactory<>("medicationName"));
98         tabelleMedikamentenIndex.setCellValueFactory(new
99             PropertyValueFactory<>("index"));
100        tabelleNebenwirkungen.setCellValueFactory(new
101            PropertyValueFactory<>("nebenwirkungen"));
102
103    }
104
105    /**
106     * Aktualisiert die Medikamentenliste und die
107     * Tabellenansicht.
108     *
109     * Diese Methode wird aufgerufen, nachdem ein neues
110     * Medikament hinzugefügt wurde, um die
111     * Liste und die Anzeige zu aktualisieren.
112     */
113     public void refreshMedicationList() {
114         medicationArrayList = medicationModel.

```

```
File - C:\Users\FHBBook\Desktop\9PG_T4 (2)\9PG_T4\MediCare_Projekt_demo\MediCare_Projekt_demo\src\main\java\com\de\hhu\medicare\MedicationController.java
105     getMedication();
106     updateTableView();
107 }
108
109 /**
110  * Behandelt das Löschen eines Medikaments.
111  *
112  * @param actionEvent Das auslösende Ereignis.
113  */
114 public void handleMedicationDelete(ActionEvent
115     actionEvent) {
116     Medication selectedMedication =
117         tabelleMedikamente.getSelectionModel().getSelectedItem();
118     if (selectedMedication != null) {
119         medicationArrayList.remove(selectedMedication
120 );
121         MedicationModel.getMedication().remove(
122             selectedMedication);
123         MedicationModel.medicationSerialize();
124         updateTableView();
125     }
126 }
127 }
```

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import javafx.scene.control.Button?>
4 <?import javafx.scene.control.Label?>
5 <?import javafx.scene.control.TextField?>
6 <?import javafx.scene.image.Image?>
7 <?import javafx.scene.image.ImageView?>
8 <?import javafx.scene.layout.AnchorPane?>
9 <?import javafx.scene.text.Font?>
10
11 <AnchorPane fx:id="LogInFenster" maxHeight="-Infinity"
12   maxWidth="-Infinity" minHeight="-Infinity" minWidth="-
13   Infinity" prefHeight="400.0" prefWidth="600.0" stylesheets
14   ="@../../style.css" xmlns="http://javafx.com/javafx/19"
15   xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.
16   example.medicare_projekt.LogInController">
17   <children>
18     <TextField fx:id="benutzerNameTextField" layoutX="
19       206.0" layoutY="175.0" promptText="Benutzername" />
20     <TextField fx:id="passwortTextField" layoutX="206.0"
21       layoutY="218.0" promptText="Passwort" />
22     <Label fx:id="LogInLabel" layoutX="206.0" layoutY="
23       118.0" prefHeight="48.0" prefWidth="64.0" text="Login"
24       textAlignment="CENTER">
25       <font>
26         <Font size="22.0" />
27       </font>
28     </Label>
29     <Button fx:id="signInButton" layoutX="247.0" layoutY
30       ="263.0" mnemonicParsing="false" onAction="#
31       handleSigninButton" prefHeight="40.0" prefWidth="84.0"
32       text="Sign in " />
33     <ImageView fitHeight="118.0" fitWidth="334.0"
34       layoutX="125.0" layoutY="14.0" pickOnBounds="true"
35       preserveRatio="true">
36       <image>
37         <Image url="@../../logo_transparent.png" />
38       </image>
39     </ImageView>
40   </children>
41 </AnchorPane>
```

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import javafx.scene.control.Button?>
4 <?import javafx.scene.control.ComboBox?>
5 <?import javafx.scene.control.DatePicker?>
6 <?import javafx.scene.control.Label?>
7 <?import javafx.scene.control.TableColumn?>
8 <?import javafx.scene.control.TableView?>
9 <?import javafx.scene.control.TextField?>
10 <?import javafx.scene.layout.AnchorPane?>
11 <?import javafx.scene.layout.HBox?>
12 <?import javafx.scene.layout.VBox?>
13 <?import javafx.scene.text.Font?>
14
15 <AnchorPane fx:id="HauptFenster" maxHeight="-Infinity"
   maxWidth="-Infinity" minHeight="-Infinity" minWidth="-
   Infinity" prefHeight="484.0" prefWidth="600.0" stylesheets
   ="@../../style.css" xmlns="http://javafx.com/javafx/19"
   xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.
   example.medicare_projekt.MainWindowController">
16     <children>
17         <Label fx:id="welcomeText" layoutX="23.0" layoutY
   ="23.0" prefHeight="17.0" prefWidth="447.0" text="Hallo!
   Bitte wählen Sie Ihre/n Patient/in:">
18         <font>
19             <Font size="22.0" />
20         </font>
21     </Label>
22     <TableView fx:id="tabelle" layoutX="11.0" layoutY
   ="56.0" onMouseClicked="#handleMedikamenteReminder"
   prefHeight="286.0" prefWidth="579.0">
23         <columns>
24             <TableColumn fx:id="tabelleName" prefWidth
   ="218.0" text="Name" />
25             <TableColumn fx:id="tabelleGeburtsdatum"
   minWidth="0.0" prefWidth="277.0" text="Geburtstag" />
26             <TableColumn fx:id="tabelleIndex" maxWidth
   ="178.0" prefWidth="83.0" text="Index" />
27         </columns>
28     </TableView>
29     <Button fx:id="addButton" layoutX="469.0" layoutY
   ="351.0" mnemonicParsing="false" onAction="#
   handlePatientHinzufuegenButton" prefHeight="17.0"
   prefWidth="114.0" text="Hinzufügen" />
```

File - C:\Users\FHBBook\Desktop\9PG_T4 (2)\9PG_T4\MediCare_Projekt_demo\MediCare_Projekt_demo\src\main\resources\

```

30      <TextField fx:id="PatientenNameTextFeld" layoutX="14.0" layoutY="351.0" promptText="Nachname, Vorname" />
31      <DatePicker fx:id="GeburtsdatumFeld" layoutX="193.0" layoutY="351.0" promptText="Geburtsdatum" />
32      <Button fx:id="viewMedicationListButton" layoutX="14.0" layoutY="387.0" mnemonicParsing="false" onAction="#handleMedikamentenListeAnsehen" prefHeight="4.0" prefWidth="214.0" text="Medikamentenliste ansehen" />
33      <TextField fx:id="indexLabel" alignment="CENTER" layoutX="399.0" layoutY="351.0" prefHeight="25.0" prefWidth="60.0" promptText="Index" />
34
35
36      <VBox alignment="CENTER" layoutX="14.0" layoutY="420.0" spacing="10" />
37      <DatePicker fx:id="datePicker" layoutX="332.0" layoutY="387.0" promptText="Datum" />
38      <HBox layoutX="332.0" layoutY="428.0" spacing="10" >
39          <ComboBox fx:id="hourComboBox" />
40          <ComboBox fx:id="minuteComboBox" />
41      </HBox>
42      <Button fx:id="reminderButton" layoutX="475.0" layoutY="428.0" onAction="#handleSetReminder" text="Alarm stellen" />
43      <Button fx:id="removePatient" layoutX="14.0" layoutY="441.0" mnemonicParsing="false" onAction="#handlePatientRemove" text="Patient löschen" />
44  </children>
45 </AnchorPane>
46

```

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import javafx.scene.control.Button?>
4 <?import javafx.scene.control.TableColumn?>
5 <?import javafx.scene.control.TableView?>
6 <?import javafx.scene.layout.AnchorPane?>
7
8 <AnchorPane maxHeight="-Infinity" maxWidth="-Infinity"
  minHeight="-Infinity" minWidth="-Infinity" prefHeight="458
  .0" prefWidth="456.0" stylesheets="@../../style.css"
  xmlns="http://javafx.com/javafx/19" xmlns:fx="http://
  javafx.com/fxml/1" fx:controller="com.example.
  medicare_projekt.MedicationListController">
9   <children>
10    <TableView fx:id="tabelleMedikamente" layoutX="6.0
  " layoutY="14.0" prefHeight="396.0" prefWidth="444.0">
11      <columns>
12        <TableColumn fx:id="
  tabelleMedikamentenIndex" prefWidth="59.0" text="Index" />
13        <TableColumn fx:id="tabelleMedikament"
  maxWidth="335.0" prefWidth="101.0" text="Medikament" />
14        <TableColumn fx:id="tabelleNebenwirkungen"
  maxWidth="292.0" prefWidth="292.0" text="Nebenwirkungen"
  />
15      </columns>
16    </TableView>
17    <Button fx:id="bmedikamentHinzufuegen" layoutX="14
  .0" layoutY="419.0" mnemonicParsing="false" onAction="#
  handleMedikamentHinzufuegenButton" prefHeight="25.0"
  prefWidth="179.0" text="Medikament hinzufügen" />
18    <Button fx:id="medicationDelete" layoutX="211.0"
  layoutY="419.0" mnemonicParsing="false" onAction="#
  handleMedicationDelete" text="Medikament löschen" />
19  </children>
20 </AnchorPane>
21
```

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import javafx.scene.control.Button?>
4 <?import javafx.scene.control.Label?>
5 <?import javafx.scene.control.TextField?>
6 <?import javafx.scene.layout.AnchorPane?>
7 <?import javafx.scene.text.Font?>
8
9 <AnchorPane maxHeight="-Infinity" maxWidth="-Infinity"
  minHeight="-Infinity" minWidth="-Infinity" prefHeight="295.0"
  prefWidth="553.0" stylesheets="@../../style.css"
  xmlns="http://javafx.com/javafx/19" xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.example.
  medicare_projekt.AddMedicationController">
10   <children>
11     <TextField fx:id="medikamentTextFeld" layoutX="236.0"
  " layoutY="34.0" />
12     <Label layoutX="37.0" layoutY="34.0" prefHeight="25.0"
  prefWidth="199.0" text="Medikamentenname:>
13       <font>
14         <Font size="19.0" />
15       </font>
16     </Label>
17     <TextField fx:id="indexTextFeld" layoutX="236.0"
  layoutY="91.0" prefHeight="25.0" prefWidth="73.0" />
18     <Label layoutX="37.0" layoutY="91.0" prefHeight="25.0"
  prefWidth="65.0" text="Index: ">
19       <font>
20         <Font size="19.0" />
21       </font>
22     </Label>
23     <Label layoutX="37.0" layoutY="149.0" prefHeight="38.0"
  prefWidth="167.0" text="Nebenwirkungen: ">
24       <font>
25         <Font size="19.0" />
26       </font>
27     </Label>
28     <TextField fx:id="nebenwirkungenTextFeld" alignment
  ="TOP_LEFT" layoutX="230.0" layoutY="156.0" prefHeight="79.0"
  prefWidth="292.0" />
29     <Button fx:id="speicherButton" layoutX="442.0"
  layoutY="256.0" mnemonicParsing="false" onAction="#
  handleSpeicherButton" text="Speichern" />
30   </children>
```

31 </AnchorPane>

32

```
1 /* General Window Styling */  
2 .root {  
3     -fx-background-color: #84BCDC; /* Light Blue */  
4     -fx-font-family: "Segoe UI", Tahoma, Geneva, Verdana,  
5     sans-serif;  
6 }  
7  
8 /* Button Styling */  
9 .button {  
10     -fx-background-color: #38596C; /* Darker Blue */  
11     -fx-text-fill: white;  
12     -fx-border-radius: 5px;  
13 }  
14 .button:hover { /* Hover effect */  
15     -fx-background-color: #1565c0; /* Even darker blue on  
16     hover */  
17     -fx-cursor: hand;  
18 }  
19  
20 /* TableView Styling */  
21 .table-view {  
22     -fx-background-color: white;  
23     -fx-table-cell-border-color: transparent; /* Remove  
24     cell borders */  
25 }  
26  
27 .table-view .column-header-background { /* Column header  
28     background */  
29     -fx-background-color: black; /* Darker blue */  
30 }  
31  
32 .table-view .column-header .label {  
33     -fx-text-fill: #1565c0;  
34     -fx-alignment: center;  
35 }  
36  
37 /* Alternating Row Colors (Optional) */  
38 .table-view .table-row-cell:odd {  
39     -fx-background-color: whitesmoke;  
40 }  
41  
42 .table-view .table-row-cell:selected {  
43     -fx-background-color: lightblue; /* Change the
```

mithilfe von Gemini

File - C:\Users\FHBBook\Desktop\9PG_T4 (2)\9PG_T4\MediCare_Projekt_demo\MediCare_Projekt_demo\src\main\resources\

```
40 background-color of selected cells */
41     -fx-text-fill: black;             /* Ensure text is
42 }                                         visible (optional) */
43
44 .table-view .table-row-cell:odd:selected {
45     -fx-background-color: lightblue; /* Change the
46                                         background color of odd selected cells */
47     -fx-text-fill: black;           /* Ensure text is
48                                         visible (optional) */
49 }
50                                         visible (optional) */
51 }
52
53 .table-row-cell:even {
54     -fx-background-color: white;    /* Set background
55                                         color for even rows */
56 }
```