src/verwaltung/BenachrichtigungsVerwaltung.java

```
package verwaltung;

import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

import model.*;

/**
 * Verwaltungsklasse für automatische Benachrichtigungen
 */
public class BenachrichtigungsVerwaltung {
    private List<Benachrichtigung> benachrichtigungen;
    private int naechsteId = 1;

    public BenachrichtigungsVerwaltung() {
        this.benachrichtigungen = new ArrayList<>();
    }

    /**
     * Erstellt automatische Erinnerungen für anstehende Fristen
     */
    public void erstelleFristerinnerungen(List<Student> studenten, List<Klausur> klausuren) {
        LocalDate heute = LocalDate.now();

        for (Klausur klausur : klausuren) {
            long tageZurFrist = ChronoUnit.DAYS.between(heute, klausur.getAnmeldefrist());

            // Warnung 7 Tage vor Fristablauf
            if (tageZurFrist == 7) {
                for (Student student : studenten) {
                    String nachricht = "■■ Die Anmeldefrist für '" + klausur.getTitel() +
                                       "' läuft in 7 Tagen ab! (Frist: " + klausur.getAnmeldefrist() + ")";

                    Benachrichtigung benachrichtigung = new Benachrichtigung(
                            "FRIST_" + naechsteId++,
                            student,
                            nachricht,
                            Benachrichtigung.BenachrichtigungsTyp.FRISTERINNERUNG
                    );

                    hinzufuegen(benachrichtigung);
                    benachrichtigung.sende();
                }
            }

            // Finale Warnung 1 Tag vor Fristablauf
            if (tageZurFrist == 1) {
                for (Student student : studenten) {
                    String nachricht = "■ LETZTE CHANCE! Anmeldefrist für '" + klausur.getTitel() +
                                       "' läuft MORGEN ab! Jetzt anmelden!";

                    Benachrichtigung benachrichtigung = new Benachrichtigung(
                            "URGENT_" + naechsteId++,
                            student,
                            nachricht,
                            Benachrichtigung.BenachrichtigungsTyp.WARNUNG
                    );

                    hinzufuegen(benachrichtigung);
                    benachrichtigung.sende();
                }
            }
        }
    }

    /**
     * Erstellt Benachrichtigung über neue Klausuranmeldung
     */
    public void benachrichtigeKlausuranmeldung(Student student, Klausur klausur) {
        String nachricht = "■ Erfolgreich für '" + klausur.getTitel() + "' angemeldet. " +
                           "Klausurtermin: " + klausur.getDatum().toLocalDate() + " in " + klausur.getRaum();

        Benachrichtigung benachrichtigung = new Benachrichtigung(
                "ANMELD_" + naechsteId++,
                student,
```

```
                nachricht,
                Benachrichtigung.BenachrichtigungsTyp.KLAUSUR_ANMELDUNG
        );

        hinzufuegen(benachrichtigung);
        benachrichtigung.sende();
    }

    /**
     * Benachrichtigt über verfügbare Noten
     */
    public void benachrichtigeNotenVerfuegbar(Student student, Versuch versuch) {
        String nachricht = "■ Note für '" + versuch.getKlausur().getTitel() + "' ist verfügbar: " +
                           versuch.getNote() + " (" + versuch.getBewertung() + ")";

        Benachrichtigung benachrichtigung = new Benachrichtigung(
                "NOTE_" + naechsteId++,
                student,
                nachricht,
                Benachrichtigung.BenachrichtigungsTyp.NOTE_VERFUEGBAR
        );

        hinzufuegen(benachrichtigung);
        benachrichtigung.sende();
    }

    /**
     * Fügt eine Benachrichtigung hinzu
     */
    public void hinzufuegen(Benachrichtigung benachrichtigung) {
        benachrichtigungen.add(benachrichtigung);
    }

    /**
     * Gibt alle Benachrichtigungen für einen Studenten zurück
     */
    public List<Benachrichtigung> getBenachrichtigungenFuerStudent(Student student) {
        return benachrichtigungen.stream()
                .filter(b -> b.getEmpfaenger().equals(student))
                .sorted((b1, b2) -> b2.getZeitpunkt().compareTo(b1.getZeitpunkt())) // Neueste zuerst
                .collect(Collectors.toList());
    }

    /**
     * Gibt ungelesene Benachrichtigungen für einen Studenten zurück
     */
    public List<Benachrichtigung> getUngelesene(Student student) {
        return benachrichtigungen.stream()
                .filter(b -> b.getEmpfaenger().equals(student) && !b.istGelesen())
                .sorted((b1, b2) -> b2.getZeitpunkt().compareTo(b1.getZeitpunkt()))
                .collect(Collectors.toList());
    }

    /**
     * Markiert alle Benachrichtigungen eines Studenten als gelesen
     */
    public void alleAlsGelesenMarkieren(Student student) {
        benachrichtigungen.stream()
                .filter(b -> b.getEmpfaenger().equals(student))
                .forEach(Benachrichtigung::markiereAlsGelesen);
    }
}
```

src/verwaltung/ErweiterteStudentenVerwaltung.java

```
package verwaltung;

import util.Database;
import java.sql.*;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;
import model.*;

/**
 * Erweiterte Studentenverwaltung mit Validierung und Duplikat-Prüfung
```

```
 */
public class ErweiterteStudentenVerwaltung {
    private List<Student> studenten;
    private Set<String> registrierteMatrikelnummern;
    private VersuchsVerwaltung versuchsVerwaltung;
    private BenachrichtigungsVerwaltung benachrichtigungsVerwaltung;

    public ErweiterteStudentenVerwaltung() {
        this.studenten = new ArrayList<>();
        this.registrierteMatrikelnummern = new HashSet<>();
        this.versuchsVerwaltung = new VersuchsVerwaltung();
        this.benachrichtigungsVerwaltung = new BenachrichtigungsVerwaltung();
        erstelleTabelleWennNichtVorhanden();
        ladeDatenAusDatenbank();
    }

    /**
     * Fügt einen neuen Studenten mit Validierung hinzu
     * @throws DuplikatException bei bereits existierender Matrikelnummer
     */
    public void hinzufuegenMitValidierung(Student student) throws DuplikatException {
        if (student == null) {
            throw new IllegalArgumentException("Student darf nicht null sein!");
        }

        // Prüfe auf Duplikate
        if (registrierteMatrikelnummern.contains(student.getMatrikelnummer())) {
            throw new DuplikatException(
                "Student mit Matrikelnummer " + student.getMatrikelnummer() +
                " existiert bereits!"
            );
        }

        // Füge hinzu
        studenten.add(student);
        registrierteMatrikelnummern.add(student.getMatrikelnummer());
        speichereInDatenbank(student);
    }

    /**
     * Fügt einen Studenten ohne Exception hinzu (für Tests und Beispieldaten)
     */
    public void hinzufuegen(Student student) {
        if (student == null) return;

        try {
            if (!registrierteMatrikelnummern.contains(student.getMatrikelnummer())) {
                studenten.add(student);
                registrierteMatrikelnummern.add(student.getMatrikelnummer());
                speichereInDatenbank(student);
            }
        } catch (Exception e) {
            // Ignoriere Fehler für Beispieldaten
        }
    }

    /**
     * Validiert eine Matrikelnummer bevor ein Student erstellt wird
     */
    public ValidationResult validiereMatrikelnummer(String matrikelnummer) {
        // Prüfe Format
        if (!Student.isValidMatrikelnummer(matrikelnummer)) {
            return new ValidationResult(false,
                "Matrikelnummer muss 5-8 Ziffern haben!");
        }

        // Prüfe Duplikate
        if (registrierteMatrikelnummern.contains(matrikelnummer)) {
            return new ValidationResult(false,
                "Matrikelnummer bereits vergeben!");
        }

        return new ValidationResult(true, "OK");
    }

    /**
     * Löscht einen Studenten anhand der Matrikelnummer
     */
    public boolean loeschen(String matrikelnummer) {
```

```
    boolean entfernt = studenten.removeIf(s -> s.getMatrikelnummer().equals(matrikelnummer));
    if (entfernt) {
        registrierteMatrikelnummern.remove(matrikelnummer);
        loescheAusDatenbank(matrikelnummer);
    }
    return entfernt;
}

/**
 * Sucht Studenten nach Nachname (Volltext)
 */
public List<Student> suchenNachName(String nachname) {
    return studenten.stream()
            .filter(s -> s.getNachname().toLowerCase().contains(nachname.toLowerCase()) ||
                         s.getVorname().toLowerCase().contains(nachname.toLowerCase()))
            .sorted()
            .collect(Collectors.toList());
}

/**
 * Sucht Studenten nach Studiengang
 */
public List<Student> suchenNachStudiengang(String studiengang) {
    return studenten.stream()
            .filter(s -> s.getStudiengang().toLowerCase().contains(studiengang.toLowerCase()))
            .sorted()
            .collect(Collectors.toList());
}

/**
 * Gibt alle Studenten sortiert zurück
 */
public List<Student> getAlleSortiert(SortierKriterium kriterium) {
    Comparator<Student> comparator;

    switch (kriterium) {
        case NACHNAME:
            comparator = Comparator.comparing(Student::getNachname)
                                   .thenComparing(Student::getVorname);
            break;
        case VORNAME:
            comparator = Comparator.comparing(Student::getVorname)
                                   .thenComparing(Student::getNachname);
            break;
        case STUDIENGANG:
            comparator = Comparator.comparing(Student::getStudiengang)
                                   .thenComparing(Student::getNachname);
            break;
        case MATRIKELNUMMER:
        default:
            comparator = Comparator.comparing(Student::getMatrikelnummer);
            break;
    }

    return studenten.stream()
            .sorted(comparator)
            .collect(Collectors.toList());
}

/**
 * Findet Student anhand Matrikelnummer
 */
public Student findeNachMatrikelnummer(String matrikelnummer) {
    return studenten.stream()
            .filter(s -> s.getMatrikelnummer().equals(matrikelnummer))
            .findFirst()
            .orElse(null);
}

/**
 * Meldet einen Studenten zu einer Klausur an
 */
public void anmeldenZuKlausur(String matrikelnummer, Klausur klausur)
        throws FristAbgelaufenException, KlausurKonfliktException {
    Student student = findeNachMatrikelnummer(matrikelnummer);
    if (student == null) {
        throw new IllegalArgumentException("Student nicht gefunden: " + matrikelnummer);
    }
```

```
    // Prüfe auf Konflikte mit bereits angemeldeten Klausuren
    for (Klausur angemeldet : student.getAngemeldeteKlausuren()) {
        if (angemeldet.konfliktMit(klausur)) {
            throw new KlausurKonfliktException("Zeitkonflikt mit " + angemeldet.getTitel());
        }
    }

    // Anmeldung durchführen
    student.anmeldenZuKlausur(klausur);
    klausur.studentHinzufuegen(student);

    // Benachrichtigung senden
    benachrichtigungsVerwaltung.benachrichtigeKlausuranmeldung(student, klausur);
}

/**
 * Registriert einen Prüfungsversuch und benachrichtigt den Studenten
 */
public void versuchEintragen(String matrikelnummer, Klausur klausur, double note, LocalDate datum) {
    Student student = findeNachMatrikelnummer(matrikelnummer);
    if (student == null) {
        throw new IllegalArgumentException("Student nicht gefunden: " + matrikelnummer);
    }

    Versuch versuch = new Versuch(student, klausur, note, datum);
    versuchsVerwaltung.versuchHinzufuegen(versuch);

    // Benachrichtigung über verfügbare Note
    benachrichtigungsVerwaltung.benachrichtigeNotenVerfuegbar(student, versuch);
}

/**
 * Gibt Studenten mit schlechten Leistungen zurück (Durchschnitt > 3.0)
 */
public List<Student> getStudentenMitSchlechtenLeistungen() {
    return studenten.stream()
            .filter(s -> {
                double durchschnitt = s.berechneNotendurchschnitt();
                return durchschnitt > 3.0 && durchschnitt > 0; // > 0 bedeutet es gibt Noten
            })
            .sorted(Comparator.comparing(Student::berechneNotendurchschnitt).reversed())
            .collect(Collectors.toList());
}

/**
 * Erstellt automatische Erinnerungen für alle Studenten
 */
public void erstelleAutomatischeErinnerungen(List<Klausur> klausuren) {
    benachrichtigungsVerwaltung.erstelleFristerinnerungen(studenten, klausuren);
}

/**
 * Gibt Statistiken über Matrikelnummern zurück
 */
public MatrikelnummerStatistik getMatrikelnummerStatistik() {
    if (studenten.isEmpty()) {
        return new MatrikelnummerStatistik(0, "", "", 0);
    }

    List<String> nummern = new ArrayList<>(registrierteMatrikelnummern);
    nummern.sort(String::compareTo);

    double avgLength = nummern.stream()
        .mapToInt(String::length)
        .average()
        .orElse(0);

    return new MatrikelnummerStatistik(
        nummern.size(),
        nummern.get(0),
        nummern.get(nummern.size() - 1),
        avgLength
    );
}

public VersuchsVerwaltung getVersuchsVerwaltung() {
    return versuchsVerwaltung;
}
```

```
public BenachrichtigungsVerwaltung getBenachrichtigungsVerwaltung() {
    return benachrichtigungsVerwaltung;
}

public enum SortierKriterium {
    MATRIKELNUMMER, NACHNAME, VORNAME, STUDIENGANG
}

// Hilfsklassen
public static class ValidationResult {
    public final boolean isValid;
    public final String message;

    public ValidationResult(boolean isValid, String message) {
        this.isValid = isValid;
        this.message = message;
    }
}

public static class MatrikelnummerStatistik {
    public final int anzahl;
    public final String niedrigste;
    public final String hoechste;
    public final double durchschnittlicheLaenge;

    public MatrikelnummerStatistik(int anzahl, String niedrigste,
                                   String hoechste, double durchschnittlicheLaenge) {
        this.anzahl = anzahl;
        this.niedrigste = niedrigste;
        this.hoechste = hoechste;
        this.durchschnittlicheLaenge = durchschnittlicheLaenge;
    }

    @Override
    public String toString() {
        return String.format("Anzahl: %d | Bereich: %s-%s | Durchschnittl. Länge: %.1f",
            anzahl, niedrigste, hoechste, durchschnittlicheLaenge);
    }
}

// Custom Exception für Duplikate
public static class DuplikatException extends Exception {
    public DuplikatException(String message) {
        super(message);
    }
}

// Datenbankoperationen
private void erstelleTabelleWennNichtVorhanden() {
    String sql = """
        CREATE TABLE IF NOT EXISTS student (
            matrikelnummer TEXT PRIMARY KEY,
            vorname TEXT NOT NULL,
            nachname TEXT NOT NULL,
            studiengang TEXT NOT NULL,
            geburtsdatum TEXT
        )
    """;

    try (Connection conn = Database.connect(); Statement stmt = conn.createStatement()) {
        stmt.execute(sql);
    } catch (SQLException e) {
        System.err.println("Fehler beim Erstellen der Student-Tabelle: " + e.getMessage());
    }
}

private void speichereInDatenbank(Student student) {
    String sql = "INSERT OR REPLACE INTO student (matrikelnummer, vorname, nachname, studiengang, geburtsdatum)
    try (Connection conn = Database.connect(); PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, student.getMatrikelnummer());
        pstmt.setString(2, student.getVorname());
        pstmt.setString(3, student.getNachname());
        pstmt.setString(4, student.getStudiengang());
        pstmt.setString(5, student.getGeburtsdatum() != null ? student.getGeburtsdatum().toString() : null);
        pstmt.executeUpdate();
    } catch (SQLException e) {
        System.err.println("Fehler beim Speichern des Studenten: " + e.getMessage());
    }
}
```

```
    private void loescheAusDatenbank(String matrikelnummer) {
        String sql = "DELETE FROM student WHERE matrikelnummer = ?";
        try (Connection conn = Database.connect(); PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, matrikelnummer);
            pstmt.executeUpdate();
        } catch (SQLException e) {
            System.err.println("Fehler beim Löschen des Studenten: " + e.getMessage());
        }
    }

    private void ladeDatenAusDatenbank() {
        String sql = "SELECT * FROM student";
        try (Connection conn = Database.connect(); Statement stmt = conn.createStatement(); ResultSet rs = stmt.exe
            while (rs.next()) {
                String geburtsdatumStr = rs.getString("geburtsdatum");
                LocalDate geburtsdatum = geburtsdatumStr != null ? LocalDate.parse(geburtsdatumStr) : null;

                try {
                    Student student = new Student(
                            rs.getString("matrikelnummer"),
                            rs.getString("vorname"),
                            rs.getString("nachname"),
                            rs.getString("studiengang"),
                            geburtsdatum
                    );
                    studenten.add(student);
                    registrierteMatrikelnummern.add(student.getMatrikelnummer());
                } catch (IllegalArgumentException e) {
                    // Ungültige Daten in DB - überspringe diesen Eintrag
                    System.err.println("Warnung: Ungültiger Student in Datenbank: " + e.getMessage());
                }
            }
        } catch (SQLException e) {
            System.err.println("Fehler beim Laden der Studenten: " + e.getMessage());
        }
    }

    /**
     * Zeigt alle Studenten auf der Konsole an
     */
    public void alleAnzeigen() {
        if (studenten.isEmpty()) {
            System.out.println("Keine Studenten vorhanden.");
            return;
        }

        System.out.println("\n=== ALLE STUDENTEN ===");
        studenten.stream()
                .sorted()
                .forEach(System.out::println);
        System.out.println("Gesamt: " + studenten.size() + " Studenten\n");
    }
}
```

src/verwaltung/KlausurVerwaltung.java

```
package verwaltung;

import model.*;
import util.Database;

import java.sql.*;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

/**
 * Verwaltungsklasse für Klausuren mit ArrayList-basierter Speicherung
 */
public class KlausurVerwaltung {
    private List<Klausur> klausuren;

    public KlausurVerwaltung() {
        this.klausuren = new ArrayList<>();
        erstelleTabelleWennNichtVorhanden();
    }
```

```
/**
 * Fügt eine neue Klausur hinzu
 */
public void hinzufuegen(Klausur klausur) throws KlausurKonfliktException {
    // Prüfe auf Konflikte
    for (Klausur vorhandene : klausuren) {
        if (vorhandene.konfliktMit(klausur)) {
            throw new KlausurKonfliktException("Zeitkonflikt mit " + vorhandene.getTitel());
        }
    }

    klausuren.add(klausur);
    speichereInDatenbank(klausur);
}

/**
 * Entfernt eine Klausur
 */
public boolean loeschen(String id) {
    boolean entfernt = klausuren.removeIf(k -> k.getId().equals(id));
    if (entfernt) {
        loescheAusDatenbank(id);
    }
    return entfernt;
}

/**
 * Sucht Klausuren nach Titel oder Modul
 */
public List<Klausur> suchenNachTitel(String suchtext) {
    return klausuren.stream()
            .filter(k -> k.getTitel().toLowerCase().contains(suchtext.toLowerCase()) ||
                         k.getModul().toLowerCase().contains(suchtext.toLowerCase()))
            .collect(Collectors.toList());
}

/**
 * Gibt alle Klausuren sortiert nach Datum zurück
 */
public List<Klausur> getAlleSortiert() {
    return klausuren.stream()
            .sorted()
            .collect(Collectors.toList());
}

/**
 * Findet Klausuren in einem bestimmten Zeitraum
 */
public List<Klausur> suchenNachZeitraum(LocalDate von, LocalDate bis) {
    return klausuren.stream()
            .filter(k -> {
                LocalDate klausurDatum = k.getDatum().toLocalDate();
                return !klausurDatum.isBefore(von) && !klausurDatum.isAfter(bis);
            })
            .sorted()
            .collect(Collectors.toList());
}

/**
 * Gibt kommende Klausuren zurück (ab heute)
 */
public List<Klausur> getKommendeKlausuren() {
    LocalDate heute = LocalDate.now();
    return klausuren.stream()
            .filter(k -> !k.getDatum().toLocalDate().isBefore(heute))
            .sorted()
            .collect(Collectors.toList());
}

/**
 * Findet Klausur anhand der ID
 */
public Klausur findeNachId(String id) {
    return klausuren.stream()
            .filter(k -> k.getId().equals(id))
            .findFirst()
            .orElse(null);
}
```

```
    private void erstelleTabelleWennNichtVorhanden() {
        String sql = """
            CREATE TABLE IF NOT EXISTS klausur (
                id TEXT PRIMARY KEY,
                titel TEXT NOT NULL,
                modul TEXT NOT NULL,
                datum TEXT NOT NULL,
                raum TEXT,
                max_versuche INTEGER DEFAULT 3,
                anmeldefrist TEXT NOT NULL
            )
        """;

        try (Connection conn = Database.connect(); Statement stmt = conn.createStatement()) {
            stmt.execute(sql);
        } catch (SQLException e) {
            System.err.println("Fehler beim Erstellen der Klausur-Tabelle: " + e.getMessage());
        }
    }

    private void speichereInDatenbank(Klausur klausur) {
        String sql = "INSERT INTO klausur (id, titel, modul, datum, raum, max_versuche, anmeldefrist) VALUES (?, ?,
        try (Connection conn = Database.connect(); PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, klausur.getId());
            pstmt.setString(2, klausur.getTitel());
            pstmt.setString(3, klausur.getModul());
            pstmt.setString(4, klausur.getDatum().toString());
            pstmt.setString(5, klausur.getRaum());
            pstmt.setInt(6, klausur.getMaxVersuche());
            pstmt.setString(7, klausur.getAnmeldefrist().toString());
            pstmt.executeUpdate();
        } catch (SQLException e) {
            System.err.println("Fehler beim Speichern der Klausur: " + e.getMessage());
        }
    }

    private void loescheAusDatenbank(String id) {
        String sql = "DELETE FROM klausur WHERE id = ?";
        try (Connection conn = Database.connect(); PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, id);
            pstmt.executeUpdate();
        } catch (SQLException e) {
            System.err.println("Fehler beim Löschen der Klausur: " + e.getMessage());
        }
    }
}
```

src/verwaltung/VersuchsVerwaltung.java

```
package verwaltung;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

import model.*;

/**
 * Verwaltungsklasse für Prüfungsversuche
 */
public class VersuchsVerwaltung {
    private List<Versuch> versuche;

    public VersuchsVerwaltung() {
        this.versuche = new ArrayList<>();
    }

    /**
     * Registriert einen neuen Prüfungsversuch
     */
    public void versuchHinzufuegen(Versuch versuch) {
        versuche.add(versuch);
        // Auch dem Studenten hinzufügen
        versuch.getStudent().addVersuch(versuch);
    }

    /**
     * Gibt alle Versuche eines Studenten zurück
```

```
     */
    public List<Versuch> getVersucheFuerStudent(Student student) {
        return versuche.stream()
                .filter(v -> v.getStudent().equals(student))
                .sorted()
                .collect(Collectors.toList());
    }

    /**
     * Gibt alle Versuche für eine Klausur zurück
     */
    public List<Versuch> getVersucheFuerKlausur(Klausur klausur) {
        return versuche.stream()
                .filter(v -> v.getKlausur().equals(klausur))
                .sorted()
                .collect(Collectors.toList());
    }

    /**
     * Berechnet Statistiken für eine Klausur
     */
    public KlausurStatistik berechneStatistik(Klausur klausur) {
        List<Versuch> klausurVersuche = getVersucheFuerKlausur(klausur);

        if (klausurVersuche.isEmpty()) {
            return new KlausurStatistik(0, 0.0, 0.0, 0, 0);
        }

        int gesamtVersuche = klausurVersuche.size();
        int bestandeneVersuche = (int) klausurVersuche.stream().filter(Versuch::istBestanden).count();

        double durchschnittsnote = klausurVersuche.stream()
                .filter(Versuch::istBestanden)
                .mapToDouble(Versuch::getNote)
                .average()
                .orElse(0.0);

        double bestehendenquote = (double) bestandeneVersuche / gesamtVersuche * 100;

        return new KlausurStatistik(gesamtVersuche, durchschnittsnote, bestehendenquote, bestandeneVersuche, gesamtV
    }

    /**
     * Innere Klasse für Klausur-Statistiken
     */
    public static class KlausurStatistik {
        public final int gesamtVersuche;
        public final double durchschnittsnote;
        public final double bestehendenquote;
        public final int bestanden;
        public final int nichtBestanden;

        public KlausurStatistik(int gesamtVersuche, double durchschnittsnote, double bestehendenquote, int bestanden
            this.gesamtVersuche = gesamtVersuche;
            this.durchschnittsnote = durchschnittsnote;
            this.bestehendenquote = bestehendenquote;
            this.bestanden = bestanden;
            this.nichtBestanden = nichtBestanden;
        }

        @Override
        public String toString() {
            return String.format("Versuche: %d | Bestanden: %d (%.1f%%) | Durchschnitt: %.2f",
                    gesamtVersuche, bestanden, bestehendenquote, durchschnittsnote);
        }
    }
}
```

src/model/Benachrichtigung.java

```
package model;

import java.time.LocalDateTime;

/**
 * Repräsentiert automatische Systemmeldungen
 */
public class Benachrichtigung {
    private String id;
    private Student empfaenger;
```

```
    private String nachricht;
    private LocalDateTime zeitpunkt;
    private BenachrichtigungsTyp typ;
    private boolean gelesen;

    public enum BenachrichtigungsTyp {
        FRISTERINNERUNG, KLAUSUR_ANMELDUNG, NOTE_VERFUEGBAR, WARNUNG
    }

    public Benachrichtigung(String id, Student empfaenger, String nachricht, BenachrichtigungsTyp typ) {
        this.id = id;
        this.empfaenger = empfaenger;
        this.nachricht = nachricht;
        this.typ = typ;
        this.zeitpunkt = LocalDateTime.now();
        this.gelesen = false;
    }

    // Getter/Setter
    public String getId() { return id; }
    public Student getEmpfaenger() { return empfaenger; }
    public String getNachricht() { return nachricht; }
    public LocalDateTime getZeitpunkt() { return zeitpunkt; }
    public BenachrichtigungsTyp getTyp() { return typ; }
    public boolean istGelesen() { return gelesen; }

    public void markiereAlsGelesen() {
        this.gelesen = true;
    }

    /**
     * Simuliert das Senden der Benachrichtigung
     */
    public void sende() {
        System.out.println("■ Benachrichtigung an " + empfaenger.getVorname() + " " + empfaenger.getNachname() + ":
    }

    @Override
    public String toString() {
        String status = gelesen ? "✓" : "■";
        return status + " " + typ + ": " + nachricht + " (" + zeitpunkt.toLocalDate() + ")";
    }
}
```

src/model/Dozent.java

```
package model;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

/**
 * Dozent erbt von Person
 */
public class Dozent extends Person {
    private List<Klausur> zugeordneteKlausuren;
    private DozentenRolle rolle;

    public enum DozentenRolle {
        DOZENT, PROFESSOR, LEHRBEAUFTRAGTE, ADMIN
    }

    public Dozent(String id, String vorname, String nachname, LocalDate geburtsdatum, DozentenRolle rolle) {
        super(id, vorname, nachname, geburtsdatum);
        this.rolle = rolle;
        this.zugeordneteKlausuren = new ArrayList<>();
    }

    public void klausurHinzufuegen(Klausur klausur) {
        if (!zugeordneteKlausuren.contains(klausur)) {
            zugeordneteKlausuren.add(klausur);
        }
    }

    public List<Klausur> getZugeordneteKlausuren() {
        return new ArrayList<>(zugeordneteKlausuren);
    }

    public DozentenRolle getRolle() { return rolle; }
```

```
    public void setRolle(DozentenRolle rolle) { this.rolle = rolle; }

    @Override
    public String toString() {
        return super.toString() + " - " + rolle;
    }
}
```

src/model/FristAbgelaufenException.java

```
package model;

/**
 * Exception für abgelaufene Fristen
 */
public class FristAbgelaufenException extends Exception {
    public FristAbgelaufenException(String message) {
        super(message);
    }
}
```

src/model/Klausur.java

```
package model;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

/**
 * Repräsentiert eine Prüfung/Klausur
 */
public class Klausur implements Comparable<Klausur> {
    private String id;
    private String titel;
    private String modul;
    private LocalDateTime datum;
    private String raum;
    private int maxVersuche;
    private LocalDate anmeldefrist;
    private List<Student> teilnehmendeStudenten;
    private Dozent verantwortlicherDozent;

    public Klausur(String id, String titel, String modul, LocalDateTime datum, String raum, int maxVersuche, LocalD
        this.id = id;
        this.titel = titel;
        this.modul = modul;
        this.datum = datum;
        this.raum = raum;
        this.maxVersuche = maxVersuche;
        this.anmeldefrist = anmeldefrist;
        this.teilnehmendeStudenten = new ArrayList<>();
    }

    // Getter/Setter
    public String getId() { return id; }
    public String getTitel() { return titel; }
    public String getModul() { return modul; }
    public LocalDateTime getDatum() { return datum; }
    public String getRaum() { return raum; }
    public int getMaxVersuche() { return maxVersuche; }
    public LocalDate getAnmeldefrist() { return anmeldefrist; }

    public void setDatum(LocalDateTime datum) { this.datum = datum; }
    public void setRaum(String raum) { this.raum = raum; }
    public void setAnmeldefrist(LocalDate anmeldefrist) { this.anmeldefrist = anmeldefrist; }
    public void setVerantwortlicherDozent(Dozent dozent) { this.verantwortlicherDozent = dozent; }

    /**
     * Prüft, ob die Anmeldefrist abgelaufen ist
     */
    public boolean istFristAbgelaufen() {
        return LocalDate.now().isAfter(anmeldefrist);
    }

    /**
     * Prüft, ob ein Student zur Klausur zugelassen ist
     */
    public boolean istStudentZugelassen(Student student) {
        return teilnehmendeStudenten.contains(student);
```

```
    }

    /**
     * Prüft auf zeitliche Konflikte mit einer anderen Klausur
     */
    public boolean konfliktMit(Klausur andere) {
        if (andere == null) return false;

        // Konflikte wenn Klausuren am gleichen Tag stattfinden
        return this.datum.toLocalDate().equals(andere.datum.toLocalDate());
    }

    public void studentHinzufuegen(Student student) throws FristAbgelaufenException {
        if (istFristAbgelaufen()) {
            throw new FristAbgelaufenException("Anmeldefrist ist abgelaufen!");
        }
        if (!teilnehmendeStudenten.contains(student)) {
            teilnehmendeStudenten.add(student);
        }
    }

    public List<Student> getTeilnehmendeStudenten() {
        return new ArrayList<>(teilnehmendeStudenten);
    }

    @Override
    public int compareTo(Klausur other) {
        return this.datum.compareTo(other.datum);
    }

    @Override
    public String toString() {
        return titel + " (" + modul + ") - " + datum.toLocalDate() + " in " + raum;
    }
}
```

src/model/KlausurKonfliktException.java

```
package model;

/**
 * Exception für Konflikte bei Klausuranmeldungen
 */
public class KlausurKonfliktException extends Exception {
    public KlausurKonfliktException(String message) {
        super(message);
    }
}
```

src/model/Person.java

```
package model;

import java.time.LocalDate;

/**
 * Basis-Klasse für alle Personen im System
 */
public abstract class Person {
    protected String id;
    protected String vorname;
    protected String nachname;
    protected LocalDate geburtsdatum;

    public Person(String id, String vorname, String nachname, LocalDate geburtsdatum) {
        this.id = id;
        this.vorname = vorname;
        this.nachname = nachname;
        this.geburtsdatum = geburtsdatum;
    }

    // Getter
    public String getId() { return id; }
    public String getVorname() { return vorname; }
    public String getNachname() { return nachname; }
    public LocalDate getGeburtsdatum() { return geburtsdatum; }

    // Setter
    public void setVorname(String vorname) { this.vorname = vorname; }
```

```
    public void setNachname(String nachname) { this.nachname = nachname; }

    @Override
    public String toString() {
        return vorname + " " + nachname + " (ID: " + id + ")";
    }
}
```

src/model/Student.java

```
package model;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
import java.util.regex.Pattern;

/**
 * Student erbt von Person und erweitert um studentenspezifische Eigenschaften
 */
public class Student extends Person implements Comparable<Student> {
    // Matrikelnummer-Format: 5-8 Ziffern
    private static final Pattern MATRIKELNUMMER_PATTERN = Pattern.compile("^\\d{5,8}$");
    private static final int MIN_MATRIKELNUMMER_LENGTH = 5;
    private static final int MAX_MATRIKELNUMMER_LENGTH = 8;

    private String matrikelnummer;
    private String studiengang;
    private List<Versuch> versuche;
    private List<Klausur> angemeldeteKlausuren;

    public Student(String matrikelnummer, String vorname, String nachname, String studiengang) {
        super(matrikelnummer, vorname, nachname, LocalDate.now()); // Temporary
        validateAndInitialize(matrikelnummer, studiengang);
    }

    public Student(String matrikelnummer, String vorname, String nachname, String studiengang, LocalDate geburtsdat
        super(matrikelnummer, vorname, nachname, geburtsdatum);
        validateAndInitialize(matrikelnummer, studiengang);
    }

    private void validateAndInitialize(String matrikelnummer, String studiengang) {
        // Validiere Matrikelnummer
        if (!isValidMatrikelnummer(matrikelnummer)) {
            throw new IllegalArgumentException(
                "Ungültige Matrikelnummer! Muss " + MIN_MATRIKELNUMMER_LENGTH +
                "-" + MAX_MATRIKELNUMMER_LENGTH + " Ziffern sein. Eingegeben: " + matrikelnummer
            );
        }

        // Validiere andere Felder
        if (getVorname() == null || getVorname().trim().isEmpty()) {
            throw new IllegalArgumentException("Vorname darf nicht leer sein!");
        }
        if (getNachname() == null || getNachname().trim().isEmpty()) {
            throw new IllegalArgumentException("Nachname darf nicht leer sein!");
        }
        if (studiengang == null || studiengang.trim().isEmpty()) {
            throw new IllegalArgumentException("Studiengang darf nicht leer sein!");
        }

        this.matrikelnummer = matrikelnummer;
        this.studiengang = studiengang.trim();
        this.versuche = new ArrayList<>();
        this.angemeldeteKlausuren = new ArrayList<>();
    }

    /**
     * Validiert eine Matrikelnummer
     */
    public static boolean isValidMatrikelnummer(String matrikelnummer) {
        if (matrikelnummer == null) return false;
        return MATRIKELNUMMER_PATTERN.matcher(matrikelnummer).matches();
    }

    /**
     * Generiert eine Beispiel-Matrikelnummer für Tests
     */
    public static String generateExampleMatrikelnummer() {
        return String.format("%05d", (int)(Math.random() * 100000));
```

```
}

// Getter/Setter
public String getMatrikelnummer() { return matrikelnummer; }
public String getStudiengang() { return studiengang; }
public void setStudiengang(String studiengang) {
    if (studiengang == null || studiengang.trim().isEmpty()) {
        throw new IllegalArgumentException("Studiengang darf nicht leer sein!");
    }
    this.studiengang = studiengang.trim();
}

// Versuch-Management
public void addVersuch(Versuch versuch) {
    if (versuch != null) {
        this.versuche.add(versuch);
    }
}

public void removeVersuch(Versuch versuch) {
    this.versuche.remove(versuch);
}

public List<Versuch> getVersuche() {
    return new ArrayList<>(versuche); // Defensive copy
}

// Klausur-Anmeldung
public void anmeldenZuKlausur(Klausur klausur) throws FristAbgelaufenException {
    if (klausur.istFristAbgelaufen()) {
        throw new FristAbgelaufenException("Anmeldefrist für " + klausur.getTitel() + " ist abgelaufen!");
    }
    if (!angemeldeteKlausuren.contains(klausur)) {
        angemeldeteKlausuren.add(klausur);
    }
}

public List<Klausur> getAngemeldeteKlausuren() {
    return new ArrayList<>(angemeldeteKlausuren);
}

/**
 * Zeigt den aktuellen Prüfungsstatus für alle Klausuren
 */
public String zeigePruefungsstatus() {
    StringBuilder status = new StringBuilder();
    status.append("Prüfungsstatus für ").append(getVorname()).append(" ").append(getNachname()).append(":\n");

    for (Klausur klausur : angemeldeteKlausuren) {
        List<Versuch> klausurVersuche = getVersucheFuerKlausur(klausur);
        if (klausurVersuche.isEmpty()) {
            status.append("- ").append(klausur.getTitel()).append(": Angemeldet, noch nicht absolviert\n");
        } else {
            Versuch letzterVersuch = klausurVersuche.get(klausurVersuche.size() - 1);
            if (letzterVersuch.istBestanden()) {
                status.append("- ").append(klausur.getTitel()).append(": BESTANDEN (Note: ")
                      .append(letzterVersuch.getNote()).append(")\n");
            } else {
                status.append("- ").append(klausur.getTitel()).append(": Nicht bestanden (Versuch ")
                      .append(klausurVersuche.size()).append("/").append(klausur.getMaxVersuche()).append(")\n"
            }
        }
    }

    return status.toString();
}

/**
 * Berechnet den Notendurchschnitt aller bestandenen Prüfungen
 */
public double berechneNotendurchschnitt() {
    List<Double> bestandeneNoten = new ArrayList<>();

    for (Versuch versuch : versuche) {
        if (versuch.istBestanden()) {
            bestandeneNoten.add(versuch.getNote());
        }
    }
```

```
        if (bestandeneNoten.isEmpty()) {
            return 0.0; // Keine bestandenen Prüfungen
        }

        double summe = bestandeneNoten.stream().mapToDouble(Double::doubleValue).sum();
        return summe / bestandeneNoten.size();
    }

    private List<Versuch> getVersucheFuerKlausur(Klausur klausur) {
        return versuche.stream()
                .filter(v -> v.getKlausur().equals(klausur))
                .toList();
    }

    // Überschreibe equals und hashCode für korrekte Duplikat-Erkennung
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Student student = (Student) obj;
        return matrikelnummer.equals(student.matrikelnummer);
    }

    @Override
    public int hashCode() {
        return matrikelnummer.hashCode();
    }

    @Override
    public int compareTo(Student other) {
        return this.matrikelnummer.compareTo(other.matrikelnummer);
    }

    @Override
    public String toString() {
        return matrikelnummer + ": " + vorname + " " + nachname + " (" + studiengang + ")";
    }
}
```

src/model/Versuch.java

```
package model;

import java.time.LocalDate;

/**
 * Repräsentiert einen konkreten Prüfungsversuch
 */
public class Versuch implements Comparable<Versuch> {
    private Student student;
    private Klausur klausur;
    private double note;
    private LocalDate datum;
    private boolean erfolgreich;

    public Versuch(Student student, Klausur klausur, double note, LocalDate datum) {
        this.student = student;
        this.klausur = klausur;
        this.note = note;
        this.datum = datum;
        this.erfolgreich = istBestanden();
    }

    // Getter
    public Student getStudent() { return student; }
    public Klausur getKlausur() { return klausur; }
    public double getNote() { return note; }
    public LocalDate getDatum() { return datum; }
    public boolean isErfolgreich() { return erfolgreich; }

    /**
     * Prüft, ob der Versuch bestanden wurde (Note <= 4.0)
     */
    public boolean istBestanden() {
        return note >= 1.0 && note <= 4.0;
    }

    /**
     * Gibt eine textuelle Bewertung der Note zurück
     */
```

```
    public String getBewertung() {
        if (!istBestanden()) return "Nicht bestanden";
        if (note <= 1.3) return "Sehr gut";
        if (note <= 1.7) return "Gut";
        if (note <= 2.3) return "Gut";
        if (note <= 2.7) return "Befriedigend";
        if (note <= 3.3) return "Befriedigend";
        if (note <= 3.7) return "Ausreichend";
        if (note <= 4.0) return "Ausreichend";
        return "Nicht bestanden";
    }

    @Override
    public int compareTo(Versuch other) {
        return this.datum.compareTo(other.datum);
    }

    @Override
    public String toString() {
        return klausur.getTitel() + ": " + note + " (" + getBewertung() + ") am " + datum;
    }
}
```

src/gui/KlausurverwaltungGUI.java

```
package gui;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.Stage;
import gui.views.*;
import model.*;
import verwaltung.*;
import java.time.LocalDate;

public class KlausurverwaltungGUI extends Application {
    private ErweiterteStudentenVerwaltung studentenVerwaltung;
    private KlausurVerwaltung klausurVerwaltung;
    private BorderPane root;
    private Label statusLabel;

    @Override
    public void init() {
        // Initialisiere Verwaltungen
        studentenVerwaltung = new ErweiterteStudentenVerwaltung();
        klausurVerwaltung = new KlausurVerwaltung();
        ladeBeisspieldaten();
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Klausurverwaltungssystem v2.0");

        root = new BorderPane();
        root.setTop(createMenuBar());
        root.setCenter(createWelcomeView());
        root.setBottom(createStatusBar());

        Scene scene = new Scene(root, 1200, 700);
        scene.getStylesheets().add(getClass().getResource("/gui/resources/style.css").toExternalForm());

        primaryStage.setScene(scene);
        primaryStage.show();

        updateStatus("Willkommen im Klausurverwaltungssystem!");
    }

    private MenuBar createMenuBar() {
        MenuBar menuBar = new MenuBar();

        // Datei Menu
        Menu dateiMenu = new Menu("Datei");
        MenuItem exportItem = new MenuItem("Daten exportieren...");
        MenuItem importItem = new MenuItem("Daten importieren...");
        MenuItem beendenItem = new MenuItem("Beenden");
        beendenItem.setOnAction(e -> System.exit(0));
```

```
        dateiMenu.getItems().addAll(exportItem, importItem, new SeparatorMenuItem(), beendenItem);

        // Verwaltung Menu
        Menu verwaltungMenu = new Menu("Verwaltung");
        MenuItem studentenItem = new MenuItem("Studenten");
        studentenItem.setOnAction(e -> showStudentenVerwaltung());
        MenuItem klausurenItem = new MenuItem("Klausuren");
        klausurenItem.setOnAction(e -> showKlausurenVerwaltung());
        MenuItem anmeldungenItem = new MenuItem("Anmeldungen");
        anmeldungenItem.setOnAction(e -> showAnmeldungsVerwaltung());
        MenuItem notenItem = new MenuItem("Noten");
        notenItem.setOnAction(e -> showNotenVerwaltung());
        verwaltungMenu.getItems().addAll(studentenItem, klausurenItem, anmeldungenItem, notenItem);

        // Auswertung Menu
        Menu auswertungMenu = new Menu("Auswertung");
        MenuItem statistikItem = new MenuItem("Statistiken");
        statistikItem.setOnAction(e -> showStatistiken());
        MenuItem benachrichtigungenItem = new MenuItem("Benachrichtigungen");
        benachrichtigungenItem.setOnAction(e -> showBenachrichtigungen());
        auswertungMenu.getItems().addAll(statistikItem, benachrichtigungenItem);

        // Hilfe Menu
        Menu hilfeMenu = new Menu("Hilfe");
        MenuItem ueberItem = new MenuItem("Über...");
        ueberItem.setOnAction(e -> showAboutDialog());
        hilfeMenu.getItems().add(ueberItem);

        menuBar.getMenus().addAll(dateiMenu, verwaltungMenu, auswertungMenu, hilfeMenu);
        return menuBar;
    }

    private Node createWelcomeView() {
        VBox welcome = new VBox(20);
        welcome.setAlignment(javafx.geometry.Pos.CENTER);
        welcome.setPadding(new Insets(50));

        Label titleLabel = new Label("Klausurverwaltungssystem");
        titleLabel.setStyle("-fx-font-size: 36px; -fx-font-weight: bold;");

        Label subtitleLabel = new Label("Willkommen zur digitalen Prüfungsverwaltung");
        subtitleLabel.setStyle("-fx-font-size: 18px; -fx-text-fill: gray;");

        // Quick-Action Buttons
        HBox quickActions = new HBox(20);
        quickActions.setAlignment(javafx.geometry.Pos.CENTER);

        Button studentButton = createQuickActionButton("Studenten\nverwalten", e -> showStudentenVerwaltung());
        Button klausurButton = createQuickActionButton("Klausuren\nverwalten", e -> showKlausurenVerwaltung());
        Button anmeldungButton = createQuickActionButton("Anmeldungen\nverwalten", e -> showAnmeldungsVerwaltung())
        Button notenButton = createQuickActionButton("Noten\neintragen", e -> showNotenVerwaltung());
        Button statistikButton = createQuickActionButton("Statistiken\nanzeigen", e -> showStatistiken());

        quickActions.getChildren().addAll(studentButton, klausurButton, anmeldungButton, notenButton, statistikButt

        // Aktuelle Statistiken
        VBox stats = new VBox(10);
        stats.setAlignment(javafx.geometry.Pos.CENTER);
        stats.setPadding(new Insets(30, 0, 0, 0));

        int studentenAnzahl = studentenVerwaltung.getAlleSortiert(ErweiterteStudentenVerwaltung.SortierKriterium.NA
        int klausurenAnzahl = klausurVerwaltung.getAlleSortiert().size();
        int kommendeKlausuren = klausurVerwaltung.getKommendeKlausuren().size();

        Label statsLabel = new Label(String.format(
            "Aktuell: %d Studenten | %d Klausuren | %d kommende Prüfungen",
            studentenAnzahl, klausurenAnzahl, kommendeKlausuren
        ));
        statsLabel.setStyle("-fx-font-size: 14px;");

        stats.getChildren().add(statsLabel);

        welcome.getChildren().addAll(titleLabel, subtitleLabel, quickActions, stats);
        return welcome;
    }

    private Button createQuickActionButton(String text, javafx.event.EventHandler<javafx.event.ActionEvent> handler
        Button button = new Button();
        Label textLabel = new Label(text);
```

```
        textLabel.setStyle("-fx-font-size: 14px; -fx-font-weight: bold;");
        textLabel.setTextAlignment(javafx.scene.text.TextAlignment.CENTER);

        button.setGraphic(textLabel);
        button.setPrefSize(150, 100);
        button.setOnAction(handler);
        button.getStyleClass().add("quick-action-button");

        return button;
    }

    private HBox createStatusBar() {
        HBox statusBar = new HBox();
        statusBar.setPadding(new Insets(5, 10, 5, 10));
        statusBar.setStyle("-fx-background-color: #f0f0f0; -fx-border-color: #cccccc; -fx-border-width: 1 0 0 0;");

        statusLabel = new Label("Bereit");
        statusBar.getChildren().add(statusLabel);

        return statusBar;
    }

    private void showStudentenVerwaltung() {
        StudentenView view = new StudentenView(studentenVerwaltung, this::showNotenVerwaltung);
        root.setCenter(view);
        updateStatus("Studentenverwaltung geöffnet");
    }

    private void showKlausurenVerwaltung() {
        KlausurenView view = new KlausurenView(klausurVerwaltung);
        root.setCenter(view);
        updateStatus("Klausurverwaltung geöffnet");
    }

    private void showAnmeldungsVerwaltung() {
        AnmeldungsView view = new AnmeldungsView(studentenVerwaltung, klausurVerwaltung);
        root.setCenter(view);
        updateStatus("Anmeldungsverwaltung geöffnet");
    }

    private void showNotenVerwaltung() {
        showNotenVerwaltung(null);
    }

    private void showNotenVerwaltung(Student student) {
        NotenView view = new NotenView(studentenVerwaltung, klausurVerwaltung, this::refreshStatistikenWennOffen, s
        root.setCenter(view);
        updateStatus("Notenverwaltung geöffnet");
    }

    private void showStatistiken() {
        StatistikView view = new StatistikView(studentenVerwaltung, klausurVerwaltung);
        root.setCenter(view);
        updateStatus("Statistiken geöffnet");
    }

    private void showBenachrichtigungen() {
        BenachrichtigungsView view = new BenachrichtigungsView(studentenVerwaltung, klausurVerwaltung);
        root.setCenter(view);
        updateStatus("Benachrichtigungen geöffnet");
    }

    private void refreshStatistikenWennOffen() {
        if (root.getCenter() instanceof StatistikView) {
            ((StatistikView) root.getCenter()).refresh();
        }
    }

    private void showAboutDialog() {
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Über Klausurverwaltungssystem");
        alert.setHeaderText("Klausurverwaltungssystem v2.0");
        alert.setContentText(
            "Entwickelt für die Verwaltung von Studenten, Klausuren und Prüfungsanmeldungen.\n\n" +
            "Features:\n" +
            "• Studentenverwaltung mit Validierung\n" +
            "• Klausurplanung mit Konfliktprüfung\n" +
            "• Automatische Benachrichtigungen\n" +
            "• Umfangreiche Statistiken\n" +
```

```
            "• Persistente Datenspeicherung\n\n" +
            "© 2025 OOP-Projekt"
        );
        alert.showAndWait();
    }

    private void updateStatus(String message) {
        statusLabel.setText(message);
    }

    private void ladeBeisspieldaten() {
        try {
            // Beispiel-Studenten
            studentenVerwaltung.hinzufuegen(new Student("10001", "Max", "Mustermann",
                "Informatik", LocalDate.of(2000, 5, 15)));
            studentenVerwaltung.hinzufuegen(new Student("10002", "Anna", "Schmidt",
                "BWL", LocalDate.of(1999, 8, 22)));
            studentenVerwaltung.hinzufuegen(new Student("10003", "Tom", "Weber",
                "Informatik", LocalDate.of(2001, 3, 10)));
            studentenVerwaltung.hinzufuegen(new Student("10004", "Lisa", "Mueller",
                "Mathematik", LocalDate.of(2000, 11, 5)));
            studentenVerwaltung.hinzufuegen(new Student("10005", "Sarah", "Meyer",
                "Physik", LocalDate.of(2000, 7, 18)));

            // Beispiel-Klausuren
            klausurVerwaltung.hinzufuegen(new Klausur("OOP2025", "Objektorientierte Programmierung",
                "Informatik Grundlagen", java.time.LocalDateTime.of(2025, 7, 15, 10, 0),
                "H1", 3, LocalDate.of(2025, 7, 1)));

            klausurVerwaltung.hinzufuegen(new Klausur("MATH1", "Mathematik I",
                "Grundlagen", java.time.LocalDateTime.of(2025, 7, 20, 14, 0),
                "A101", 3, LocalDate.of(2025, 7, 5)));

            klausurVerwaltung.hinzufuegen(new Klausur("BWL1", "Grundlagen BWL",
                "Betriebswirtschaft", java.time.LocalDateTime.of(2025, 7, 18, 9, 0),
                "B205", 3, LocalDate.of(2025, 7, 3)));

        } catch (Exception e) {
            // Daten existieren bereits
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

src/gui/views/AnmeldungsView.java

```
package gui.views;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import model.*;
import verwaltung.*;
import java.util.List;
import java.util.Optional;

public class AnmeldungsView extends BorderPane {
    private ErweiterteStudentenVerwaltung studentenVerwaltung;
    private KlausurVerwaltung klausurVerwaltung;
    private ComboBox<Student> studentComboBox;
    private ListView<Klausur> verfuegbareKlausurenList;
    private ListView<Klausur> angemeldeteKlausurenList;
    private TextArea statusArea;

    public AnmeldungsView(ErweiterteStudentenVerwaltung studentenVerwaltung, KlausurVerwaltung klausurVerwaltung) {
        this.studentenVerwaltung = studentenVerwaltung;
        this.klausurVerwaltung = klausurVerwaltung;

        setTop(createControlPanel());
        setCenter(createMainContent());
        setBottom(createStatusPanel());

        aktualisiereStudentenliste();
    }
```

```
private VBox createControlPanel() {
    VBox controlPanel = new VBox(10);
    controlPanel.setPadding(new Insets(10));
    controlPanel.setStyle("-fx-background-color: #f8f9fa;");

    Label titleLabel = new Label("Klausuranmeldungen verwalten");
    titleLabel.setStyle("-fx-font-size: 18px; -fx-font-weight: bold;");

    HBox studentBox = new HBox(10);
    studentBox.setAlignment(javafx.geometry.Pos.CENTER_LEFT);
    Label studentLabel = new Label("Student auswählen:");
    studentComboBox = new ComboBox<>();
    studentComboBox.setPrefWidth(300);
    studentComboBox.setOnAction(e -> aktualisiereKlausurListen());

    Button aktualisierenButton = new Button("Aktualisieren");
    aktualisierenButton.setOnAction(e -> {
        aktualisiereStudentenliste();
        aktualisiereKlausurListen();
    });

    studentBox.getChildren().addAll(studentLabel, studentComboBox, aktualisierenButton);

    controlPanel.getChildren().addAll(titleLabel, studentBox);
    return controlPanel;
}

private HBox createMainContent() {
    HBox content = new HBox(20);
    content.setPadding(new Insets(20));

    // Verfügbare Klausuren
    VBox verfuegbarBox = new VBox(10);
    Label verfuegbarLabel = new Label("Verfügbare Klausuren");
    verfuegbarLabel.setStyle("-fx-font-size: 16px; -fx-font-weight: bold;");

    verfuegbareKlausurenList = new ListView<>();
    verfuegbareKlausurenList.setPrefSize(400, 300);
    verfuegbareKlausurenList.setCellFactory(lv -> new ListCell<Klausur>() {
        @Override
        protected void updateItem(Klausur item, boolean empty) {
            super.updateItem(item, empty);
            if (empty || item == null) {
                setText(null);
                setStyle("");
            } else {
                setText(String.format("%s - %s\n  Termin: %s | Raum: %s\n  Anmeldefrist: %s",
                    item.getId(), item.getTitel(),
                    item.getDatum().format(java.time.format.DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm")),
                    item.getRaum(),
                    item.getAnmeldefrist().format(java.time.format.DateTimeFormatter.ofPattern("dd.MM.yyyy"))
                ));

                if (item.istFristAbgelaufen()) {
                    setStyle("-fx-text-fill: gray; -fx-font-style: italic;");
                    setDisable(true);
                }
            }
        }
    });

    Button anmeldenButton = new Button("→ Anmelden");
    anmeldenButton.setOnAction(e -> studentAnmelden());
    anmeldenButton.setPrefWidth(400);

    verfuegbarBox.getChildren().addAll(verfuegbarLabel, verfuegbareKlausurenList, anmeldenButton);

    // Angemeldete Klausuren
    VBox angemeldetBox = new VBox(10);
    Label angemeldetLabel = new Label("Angemeldete Klausuren");
    angemeldetLabel.setStyle("-fx-font-size: 16px; -fx-font-weight: bold;");

    angemeldeteKlausurenList = new ListView<>();
    angemeldeteKlausurenList.setPrefSize(400, 300);
    angemeldeteKlausurenList.setCellFactory(lv -> new ListCell<Klausur>() {
        @Override
        protected void updateItem(Klausur item, boolean empty) {
            super.updateItem(item, empty);
            if (empty || item == null) {
```

```
                setText(null);
            } else {
                Student selected = studentComboBox.getValue();
                if (selected != null) {
                    List<Versuch> versuche = selected.getVersuche().stream()
                        .filter(v -> v.getKlausur().equals(item))
                        .toList();

                    String status = versuche.isEmpty() ? "Angemeldet" :
                        versuche.stream().anyMatch(v -> v.istBestanden()) ? "BESTANDEN" : "Nicht bestanden";

                    setText(String.format("%s - %s\n  Status: %s\n  Termin: %s",
                        item.getId(), item.getTitel(),
                        status,
                        item.getDatum().format(java.time.format.DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm"))
                    ));

                    if (status.equals("BESTANDEN")) {
                        setStyle("-fx-text-fill: green;");
                    } else if (status.equals("Nicht bestanden")) {
                        setStyle("-fx-text-fill: orange;");
                    }
                }
            }
        }
    });

    Button abmeldenButton = new Button("← Abmelden");
    abmeldenButton.setOnAction(e -> studentAbmelden());
    abmeldenButton.setPrefWidth(400);

    angemeldetBox.getChildren().addAll(angemeldetLabel, angemeldeteKlausurenList, abmeldenButton);

    content.getChildren().addAll(verfuegbarBox, angemeldetBox);
    return content;
}

private VBox createStatusPanel() {
    VBox statusPanel = new VBox(10);
    statusPanel.setPadding(new Insets(10));
    statusPanel.setStyle("-fx-background-color: #f0f0f0;");

    Label statusLabel = new Label("Status und Meldungen:");
    statusLabel.setStyle("-fx-font-weight: bold;");

    statusArea = new TextArea();
    statusArea.setPrefRowCount(3);
    statusArea.setEditable(false);
    statusArea.setWrapText(true);

    statusPanel.getChildren().addAll(statusLabel, statusArea);
    return statusPanel;
}

private void aktualisiereStudentenliste() {
    ObservableList<Student> studenten = FXCollections.observableArrayList(
        studentenVerwaltung.getAlleSortiert(ErweiterteStudentenVerwaltung.SortierKriterium.NACHNAME)
    );
    studentComboBox.setItems(studenten);

    // Custom Cell Factory für bessere Anzeige
    studentComboBox.setCellFactory(lv -> new ListCell<Student>() {
        @Override
        protected void updateItem(Student item, boolean empty) {
            super.updateItem(item, empty);
            setText(empty || item == null ? null :
                item.getMatrikelnummer() + " - " + item.getNachname() + ", " + item.getVorname());
        }
    });

    studentComboBox.setButtonCell(new ListCell<Student>() {
        @Override
        protected void updateItem(Student item, boolean empty) {
            super.updateItem(item, empty);
            setText(empty || item == null ? null :
                item.getMatrikelnummer() + " - " + item.getNachname() + ", " + item.getVorname());
        }
    });
}
```

```
private void aktualisiereKlausurListen() {
    Student selected = studentComboBox.getValue();
    if (selected == null) {
        verfuegbareKlausurenList.getItems().clear();
        angemeldeteKlausurenList.getItems().clear();
        statusArea.setText("Bitte wählen Sie einen Studenten aus.");
        return;
    }

    // Angemeldete Klausuren
    ObservableList<Klausur> angemeldet = FXCollections.observableArrayList(
        selected.getAngemeldeteKlausuren()
    );
    angemeldeteKlausurenList.setItems(angemeldet);

    // Verfügbare Klausuren (noch nicht angemeldet)
    List<Klausur> alleKlausuren = klausurVerwaltung.getKommendeKlausuren();
    ObservableList<Klausur> verfuegbar = FXCollections.observableArrayList(
        alleKlausuren.stream()
            .filter(k -> !angemeldet.contains(k))
            .toList()
    );
    verfuegbareKlausurenList.setItems(verfuegbar);

    // Status aktualisieren
    statusArea.setText(String.format("Student: %s\nAngemeldete Klausuren: %d\nVerfügbare Klausuren: %d",
        selected.toString(), angemeldet.size(), verfuegbar.size()));
}

private void studentAnmelden() {
    Student student = studentComboBox.getValue();
    Klausur klausur = verfuegbareKlausurenList.getSelectionModel().getSelectedItem();

    if (student == null || klausur == null) {
        showError("Fehler", "Bitte wählen Sie einen Studenten und eine Klausur aus.");
        return;
    }

    try {
        studentenVerwaltung.anmeldenZuKlausur(student.getMatrikelnummer(), klausur);
        aktualisiereKlausurListen();
        statusArea.appendText(String.format("\n✓ %s erfolgreich zu %s angemeldet.",
            student.getVorname(), klausur.getTitel()));
    } catch (FristAbgelaufenException e) {
        showError("Anmeldefrist abgelaufen",
            "Die Anmeldefrist für diese Klausur ist bereits abgelaufen.");
    } catch (KlausurKonfliktException e) {
        showError("Terminkonflikt", e.getMessage());
    } catch (Exception e) {
        showError("Fehler", "Anmeldung fehlgeschlagen: " + e.getMessage());
    }
}

private void studentAbmelden() {
    Student student = studentComboBox.getValue();
    Klausur klausur = angemeldeteKlausurenList.getSelectionModel().getSelectedItem();

    if (student == null || klausur == null) {
        showError("Fehler", "Bitte wählen Sie einen Studenten und eine Klausur aus.");
        return;
    }

    // Prüfe ob bereits Versuche existieren
    boolean hatVersuche = student.getVersuche().stream()
        .anyMatch(v -> v.getKlausur().equals(klausur));

    if (hatVersuche) {
        showError("Abmeldung nicht möglich",
            "Student hat bereits an dieser Klausur teilgenommen.");
        return;
    }

    Alert confirm = new Alert(Alert.AlertType.CONFIRMATION);
    confirm.setTitle("Abmeldung bestätigen");
    confirm.setHeaderText("Wirklich abmelden?");
    confirm.setContentText(String.format("%s von %s abmelden?",
        student.getVorname() + " " + student.getNachname(), klausur.getTitel()));
```

```
        confirm.showAndWait().ifPresent(result -> {
            if (result == ButtonType.OK) {
                // Hier würde die Abmeldung implementiert
                // Momentan nur UI-Update
                aktualisiereKlausurListen();
                statusArea.appendText(String.format("\n✓ %s von %s abgemeldet.",
                    student.getVorname(), klausur.getTitel()));
            }
        });
    }

    private void showError(String title, String content) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(content);
        alert.showAndWait();
    }
}
```

src/gui/views/BenachrichtigungsView.java

```
package gui.views;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import model.*;
import verwaltung.*;
import java.time.format.DateTimeFormatter;
import java.util.List;

public class BenachrichtigungsView extends BorderPane {
    private ErweiterteStudentenVerwaltung studentenVerwaltung;
    private KlausurVerwaltung klausurVerwaltung;
    private ComboBox<Student> studentComboBox;
    private ListView<Benachrichtigung> benachrichtigungsListe;
    private TextArea detailArea;
    private Label statistikLabel;
    private CheckBox nurUngeleseneCheckBox;

    public BenachrichtigungsView(ErweiterteStudentenVerwaltung studentenVerwaltung, KlausurVerwaltung klausurVerwal
        this.studentenVerwaltung = studentenVerwaltung;
        this.klausurVerwaltung = klausurVerwaltung;

        setTop(createControlPanel());
        setCenter(createMainContent());
        setBottom(createActionPanel());

        aktualisiereStudentenliste();
    }

    private VBox createControlPanel() {
        VBox controlPanel = new VBox(10);
        controlPanel.setPadding(new Insets(10));
        controlPanel.setStyle("-fx-background-color: #f8f9fa;");

        Label titleLabel = new Label("Benachrichtigungssystem");
        titleLabel.setStyle("-fx-font-size: 18px; -fx-font-weight: bold;");

        HBox filterBox = new HBox(10);
        filterBox.setAlignment(javafx.geometry.Pos.CENTER_LEFT);

        Label studentLabel = new Label("Student:");
        studentComboBox = new ComboBox<>();
        studentComboBox.setPrefWidth(300);
        studentComboBox.setOnAction(e -> aktualisiereBenachrichtigungen());

        nurUngeleseneCheckBox = new CheckBox("Nur ungelesene");
        nurUngeleseneCheckBox.setOnAction(e -> aktualisiereBenachrichtigungen());

        Button alleStudentenButton = new Button("Alle Studenten");
        alleStudentenButton.setOnAction(e -> {
            studentComboBox.setValue(null);
            aktualisiereBenachrichtigungen();
        });

        filterBox.getChildren().addAll(studentLabel, studentComboBox, nurUngeleseneCheckBox, alleStudentenButton);
```

```
        statistikLabel = new Label();
        statistikLabel.setStyle("-fx-font-style: italic;");

        controlPanel.getChildren().addAll(titleLabel, filterBox, statistikLabel);
        return controlPanel;
    }

    private SplitPane createMainContent() {
        SplitPane splitPane = new SplitPane();

        // Linke Seite: Benachrichtigungsliste
        VBox leftBox = new VBox(10);
        leftBox.setPadding(new Insets(10));

        Label listLabel = new Label("Benachrichtigungen");
        listLabel.setStyle("-fx-font-weight: bold;");

        benachrichtigungsListe = new ListView<>();
        benachrichtigungsListe.setPrefHeight(400);
        benachrichtigungsListe.setCellFactory(lv -> new ListCell<Benachrichtigung>() {
            @Override
            protected void updateItem(Benachrichtigung item, boolean empty) {
                super.updateItem(item, empty);
                if (empty || item == null) {
                    setText(null);
                    setGraphic(null);
                    setStyle("");
                } else {
                    VBox cellContent = new VBox(2);

                    HBox headerBox = new HBox(10);
                    Label typLabel = new Label(getTypIcon(item.getTyp()) + " " + item.getTyp().toString());
                    typLabel.setStyle("-fx-font-weight: bold;");

                    Label zeitLabel = new Label(item.getZeitpunkt().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
                    zeitLabel.setStyle("-fx-font-size: 11px; -fx-text-fill: gray;");

                    headerBox.getChildren().addAll(typLabel, zeitLabel);

                    Label nachrichtLabel = new Label(item.getNachricht());
                    nachrichtLabel.setWrapText(true);
                    nachrichtLabel.setMaxWidth(350);

                    Label empfaengerLabel = new Label("An: " + item.getEmpfaenger().getVorname() + " " + item.getEm
                    empfaengerLabel.setStyle("-fx-font-size: 11px; -fx-text-fill: #666;");

                    cellContent.getChildren().addAll(headerBox, nachrichtLabel, empfaengerLabel);

                    setGraphic(cellContent);

                    if (!item.istGelesen()) {
                        setStyle("-fx-background-color: #e8f4fd;");
                    } else {
                        setStyle("");
                    }
                }
            }
        });

        benachrichtigungsListe.getSelectionModel().selectedItemProperty().addListener(
            (obs, oldSel, newSel) -> {
                if (newSel != null) {
                    zeigeDetails(newSel);
                }
            }
        );

        leftBox.getChildren().addAll(listLabel, benachrichtigungsListe);

        // Rechte Seite: Details
        VBox rightBox = new VBox(10);
        rightBox.setPadding(new Insets(10));

        Label detailLabel = new Label("Details");
        detailLabel.setStyle("-fx-font-weight: bold;");

        detailArea = new TextArea();
        detailArea.setEditable(false);
```

```
        detailArea.setWrapText(true);
        detailArea.setPrefRowCount(20);

        rightBox.getChildren().addAll(detailLabel, detailArea);

        splitPane.getItems().addAll(leftBox, rightBox);
        splitPane.setDividerPositions(0.6);

        return splitPane;
    }

    private HBox createActionPanel() {
        HBox actionPanel = new HBox(10);
        actionPanel.setPadding(new Insets(10));
        actionPanel.setStyle("-fx-background-color: #f0f0f0;");

        Button erinnerungenButton = new Button("Automatische Erinnerungen erstellen");
        erinnerungenButton.setOnAction(e -> erstelleErinnerungen());

        Button alsGelesenButton = new Button("Als gelesen markieren");
        alsGelesenButton.setOnAction(e -> markiereAlsGelesen());

        Button alleAlsGelesenButton = new Button("Alle als gelesen");
        alleAlsGelesenButton.setOnAction(e -> markiereAlleAlsGelesen());

        Button loeschenButton = new Button("Löschen");
        loeschenButton.setOnAction(e -> loescheBenachrichtigung());

        actionPanel.getChildren().addAll(
            erinnerungenButton,
            new Separator(javafx.geometry.Orientation.VERTICAL),
            alsGelesenButton,
            alleAlsGelesenButton,
            new Separator(javafx.geometry.Orientation.VERTICAL),
            loeschenButton
        );

        return actionPanel;
    }

    private void aktualisiereStudentenliste() {
        ObservableList<Student> studenten = FXCollections.observableArrayList(
            studentenVerwaltung.getAlleSortiert(ErweiterteStudentenVerwaltung.SortierKriterium.NACHNAME)
        );
        studentComboBox.setItems(studenten);

        studentComboBox.setCellFactory(lv -> new ListCell<Student>() {
            @Override
            protected void updateItem(Student item, boolean empty) {
                super.updateItem(item, empty);
                setText(empty || item == null ? null :
                    item.getMatrikelnummer() + " - " + item.getNachname() + ", " + item.getVorname());
            }
        });

        studentComboBox.setButtonCell(new ListCell<Student>() {
            @Override
            protected void updateItem(Student item, boolean empty) {
                super.updateItem(item, empty);
                setText(empty || item == null ? "Alle Studenten" :
                    item.getMatrikelnummer() + " - " + item.getNachname() + ", " + item.getVorname());
            }
        });
    }

    private void aktualisiereBenachrichtigungen() {
        ObservableList<Benachrichtigung> liste = FXCollections.observableArrayList();
        BenachrichtigungsVerwaltung verwaltung = studentenVerwaltung.getBenachrichtigungsVerwaltung();

        Student selectedStudent = studentComboBox.getValue();

        if (selectedStudent != null) {
            // Benachrichtigungen für einen bestimmten Studenten
            if (nurUngeleseneCheckBox.isSelected()) {
                liste.addAll(verwaltung.getUngelesene(selectedStudent));
            } else {
                liste.addAll(verwaltung.getBenachrichtigungenFuerStudent(selectedStudent));
            }
        } else {
```

```
        // Alle Benachrichtigungen
        for (Student student : studentenVerwaltung.getAlleSortiert(ErweiterteStudentenVerwaltung.SortierKriteri
            if (nurUngeleseneCheckBox.isSelected()) {
                liste.addAll(verwaltung.getUngelesene(student));
            } else {
                liste.addAll(verwaltung.getBenachrichtigungenFuerStudent(student));
            }
        }
    }

    benachrichtigungsListe.setItems(liste);

    // Statistik aktualisieren
    int ungelesen = (int) liste.stream().filter(b -> !b.istGelesen()).count();
    statistikLabel.setText(String.format("Gesamt: %d | Ungelesen: %d", liste.size(), ungelesen));

    detailArea.clear();
}

private void zeigeDetails(Benachrichtigung benachrichtigung) {
    StringBuilder details = new StringBuilder();

    details.append("=== BENACHRICHTIGUNGSDETAILS ===\n\n");
    details.append("Typ: ").append(benachrichtigung.getTyp()).append("\n");
    details.append("Empfänger: ").append(benachrichtigung.getEmpfaenger()).append("\n");
    details.append("Zeitpunkt: ").append(benachrichtigung.getZeitpunkt().format(DateTimeFormatter.ofPattern("dd
    details.append("Status: ").append(benachrichtigung.istGelesen() ? "Gelesen" : "Ungelesen").append("\n\n");
    details.append("Nachricht:\n");
    details.append("-".repeat(40)).append("\n");
    details.append(benachrichtigung.getNachricht()).append("\n");
    details.append("-".repeat(40)).append("\n");

    detailArea.setText(details.toString());

    // Markiere als gelesen
    if (!benachrichtigung.istGelesen()) {
        benachrichtigung.markiereAlsGelesen();
        aktualisiereBenachrichtigungen();
    }
}

private void erstelleErinnerungen() {
    Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
    alert.setTitle("Erinnerungen erstellen");
    alert.setHeaderText("Automatische Erinnerungen");
    alert.setContentText("Möchten Sie für alle Studenten automatische Erinnerungen für anstehende Klausurfriste

    alert.showAndWait().ifPresent(result -> {
        if (result == ButtonType.OK) {
            List<Klausur> klausuren = klausurVerwaltung.getKommendeKlausuren();
            studentenVerwaltung.erstelleAutomatischeErinnerungen(klausuren);
            aktualisiereBenachrichtigungen();

            Alert info = new Alert(Alert.AlertType.INFORMATION);
            info.setTitle("Erinnerungen erstellt");
            info.setHeaderText(null);
            info.setContentText("Automatische Erinnerungen wurden erfolgreich erstellt.");
            info.showAndWait();
        }
    });
}

private void markiereAlsGelesen() {
    Benachrichtigung selected = benachrichtigungsListe.getSelectionModel().getSelectedItem();
    if (selected != null && !selected.istGelesen()) {
        selected.markiereAlsGelesen();
        aktualisiereBenachrichtigungen();
    }
}

private void markiereAlleAlsGelesen() {
    Student selectedStudent = studentComboBox.getValue();

    Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
    alert.setTitle("Alle als gelesen markieren");
    alert.setHeaderText("Bestätigung");

    if (selectedStudent != null) {
        alert.setContentText("Alle Benachrichtigungen von " + selectedStudent.getVorname() + " " + selectedStud
```

```
        } else {
            alert.setContentText("Wirklich ALLE Benachrichtigungen als gelesen markieren?");
        }

        alert.showAndWait().ifPresent(result -> {
            if (result == ButtonType.OK) {
                BenachrichtigungsVerwaltung verwaltung = studentenVerwaltung.getBenachrichtigungsVerwaltung();

                if (selectedStudent != null) {
                    verwaltung.alleAlsGelesenMarkieren(selectedStudent);
                } else {
                    for (Student student : studentenVerwaltung.getAlleSortiert(ErweiterteStudentenVerwaltung.Sortie
                        verwaltung.alleAlsGelesenMarkieren(student);
                    }
                }

                aktualisiereBenachrichtigungen();
            }
        });
    }

    private void loescheBenachrichtigung() {
        Benachrichtigung selected = benachrichtigungsListe.getSelectionModel().getSelectedItem();
        if (selected != null) {
            Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
            alert.setTitle("Benachrichtigung löschen");
            alert.setHeaderText("Bestätigung");
            alert.setContentText("Diese Benachrichtigung wirklich löschen?");

            alert.showAndWait().ifPresent(result -> {
                if (result == ButtonType.OK) {
                    // Hier würde die Löschfunktion implementiert
                    // Momentan nur UI-Update
                    aktualisiereBenachrichtigungen();
                }
            });
        }
    }

    private String getTypIcon(Benachrichtigung.BenachrichtigungsTyp typ) {
        switch (typ) {
            case FRISTERINNERUNG:
                return "■";
            case KLAUSUR_ANMELDUNG:
                return "✍■";
            case NOTE_VERFUEGBAR:
                return "■";
            case WARNUNG:
                return "■■";
            default:
                return "■";
        }
    }
}
```

src/gui/views/KlausurenView.java

```
package gui.views;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.*;
import model.Klausur;
import model.KlausurKonfliktException;
import verwaltung.KlausurVerwaltung;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Optional;

public class KlausurenView extends BorderPane {
    private KlausurVerwaltung verwaltung;
    private TableView<Klausur> tableView;
    private ObservableList<Klausur> klausurenListe;
    private CheckBox nurKommendeCheckBox;
    private static final DateTimeFormatter DATE_TIME_FORMAT = DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm");
    private static final DateTimeFormatter DATE_FORMAT = DateTimeFormatter.ofPattern("dd.MM.yyyy");
```

```
public KlausurenView(KlausurVerwaltung verwaltung) {
    this.verwaltung = verwaltung;
    this.klausurenListe = FXCollections.observableArrayList();

    setTop(createToolBar());
    setCenter(createTableView());
    setBottom(createInfoBar());

    aktualisiereListe();
}

private ToolBar createToolBar() {
    ToolBar toolBar = new ToolBar();

    Button neuButton = new Button("Neue Klausur");
    neuButton.setOnAction(e -> zeigeKlausurDialog(null));

    Button bearbeitenButton = new Button("Bearbeiten");
    bearbeitenButton.setOnAction(e -> {
        Klausur selected = tableView.getSelectionModel().getSelectedItem();
        if (selected != null) {
            zeigeKlausurDialog(selected);
        }
    });

    Button loeschenButton = new Button("Löschen");
    loeschenButton.setOnAction(e -> klausurLoeschen());

    nurKommendeCheckBox = new CheckBox("Nur kommende Klausuren");
    nurKommendeCheckBox.setOnAction(e -> aktualisiereListe());

    Button aktualisierenButton = new Button("Aktualisieren");
    aktualisierenButton.setOnAction(e -> aktualisiereListe());

    toolBar.getItems().addAll(
        neuButton, bearbeitenButton, loeschenButton,
        new Separator(),
        nurKommendeCheckBox,
        new Separator(),
        aktualisierenButton
    );

    return toolBar;
}

private TableView<Klausur> createTableView() {
    tableView = new TableView<>();

    TableColumn<Klausur, String> idCol = new TableColumn<>("ID");
    idCol.setCellValueFactory(new PropertyValueFactory<>("id"));
    idCol.setPrefWidth(80);

    TableColumn<Klausur, String> titelCol = new TableColumn<>("Titel");
    titelCol.setCellValueFactory(new PropertyValueFactory<>("titel"));
    titelCol.setPrefWidth(250);

    TableColumn<Klausur, String> modulCol = new TableColumn<>("Modul");
    modulCol.setCellValueFactory(new PropertyValueFactory<>("modul"));
    modulCol.setPrefWidth(200);

    TableColumn<Klausur, LocalDateTime> datumCol = new TableColumn<>("Datum/Uhrzeit");
    datumCol.setCellValueFactory(new PropertyValueFactory<>("datum"));
    datumCol.setPrefWidth(150);
    datumCol.setCellFactory(col -> new TableCell<Klausur, LocalDateTime>() {
        @Override
        protected void updateItem(LocalDateTime item, boolean empty) {
            super.updateItem(item, empty);
            if (empty || item == null) {
                setText(null);
                setStyle("");
            } else {
                setText(item.format(DATE_TIME_FORMAT));
                if (item.isBefore(LocalDateTime.now())) {
                    setStyle("-fx-text-fill: gray;");
                } else if (item.isBefore(LocalDateTime.now().plusDays(7))) {
                    setStyle("-fx-text-fill: orange; -fx-font-weight: bold;");
                } else {
                    setStyle("");
```

```
                }
            }
        }
    });

    TableColumn<Klausur, String> raumCol = new TableColumn<>("Raum");
    raumCol.setCellValueFactory(new PropertyValueFactory<>("raum"));
    raumCol.setPrefWidth(80);

    TableColumn<Klausur, LocalDate> fristCol = new TableColumn<>("Anmeldefrist");
    fristCol.setCellValueFactory(new PropertyValueFactory<>("anmeldefrist"));
    fristCol.setPrefWidth(120);
    fristCol.setCellFactory(col -> new TableCell<Klausur, LocalDate>() {
        @Override
        protected void updateItem(LocalDate item, boolean empty) {
            super.updateItem(item, empty);
            if (empty || item == null) {
                setText(null);
                setStyle("");
            } else {
                setText(item.format(DATE_FORMAT));
                if (item.isBefore(LocalDate.now())) {
                    setText(item.format(DATE_FORMAT) + " (abgelaufen)");
                    setStyle("-fx-text-fill: red;");
                } else if (item.isBefore(LocalDate.now().plusDays(3))) {
                    setStyle("-fx-text-fill: orange; -fx-font-weight: bold;");
                } else {
                    setStyle("");
                }
            }
        }
    });

    TableColumn<Klausur, Integer> versucheCol = new TableColumn<>("Max. Versuche");
    versucheCol.setCellValueFactory(new PropertyValueFactory<>("maxVersuche"));
    versucheCol.setPrefWidth(100);

    tableView.getColumns().addAll(idCol, titelCol, modulCol, datumCol, raumCol, fristCol, versucheCol);
    tableView.setItems(klausurenListe);

    // Kontextmenü
    ContextMenu contextMenu = new ContextMenu();
    MenuItem teilnehmerItem = new MenuItem("Teilnehmer anzeigen");
    teilnehmerItem.setOnAction(e -> zeigeTeilnehmer());
    contextMenu.getItems().add(teilnehmerItem);
    tableView.setContextMenu(contextMenu);

    return tableView;
}

private HBox createInfoBar() {
    HBox infoBar = new HBox(20);
    infoBar.setPadding(new Insets(10));
    infoBar.setStyle("-fx-background-color: #f0f0f0;");

    Label gesamtLabel = new Label();
    Label kommendeLabel = new Label();
    Label heuteLabel = new Label();

    aktualisiereInfo(gesamtLabel, kommendeLabel, heuteLabel);

    infoBar.getChildren().addAll(
        new Label("Gesamt:"), gesamtLabel,
        new Separator(javafx.geometry.Orientation.VERTICAL),
        new Label("Kommende:"), kommendeLabel,
        new Separator(javafx.geometry.Orientation.VERTICAL),
        new Label("Diese Woche:"), heuteLabel
    );

    return infoBar;
}

private void aktualisiereInfo(Label gesamt, Label kommende, Label woche) {
    int gesamtAnzahl = verwaltung.getAlleSortiert().size();
    int kommendeAnzahl = verwaltung.getKommendeKlausuren().size();
    int wocheAnzahl = (int) verwaltung.getKommendeKlausuren().stream()
        .filter(k -> k.getDatum().isBefore(LocalDateTime.now().plusWeeks(1)))
        .count();
```

```
    gesamt.setText(String.valueOf(gesamtAnzahl));
    kommende.setText(String.valueOf(kommendeAnzahl));
    woche.setText(String.valueOf(wocheAnzahl));

    if (wocheAnzahl > 0) {
        woche.setStyle("-fx-text-fill: orange; -fx-font-weight: bold;");
    }
}

private void zeigeKlausurDialog(Klausur klausur) {
    Dialog<Klausur> dialog = new Dialog<>();
    dialog.setTitle(klausur == null ? "Neue Klausur" : "Klausur bearbeiten");
    dialog.setHeaderText(klausur == null ? "Neue Klausur anlegen" : "Klausurdaten bearbeiten");

    GridPane grid = new GridPane();
    grid.setHgap(10);
    grid.setVgap(10);
    grid.setPadding(new Insets(20, 150, 10, 10));

    TextField idField = new TextField(klausur != null ? klausur.getId() : "");
    idField.setPromptText("z.B. OOP2025");
    idField.setDisable(klausur != null);

    TextField titelField = new TextField(klausur != null ? klausur.getTitel() : "");
    TextField modulField = new TextField(klausur != null ? klausur.getModul() : "");
    TextField raumField = new TextField(klausur != null ? klausur.getRaum() : "");

    DatePicker datumPicker = new DatePicker(klausur != null ? klausur.getDatum().toLocalDate() : LocalDate.now(
    Spinner<Integer> stundeSpinner = new Spinner<>(0, 23, klausur != null ? klausur.getDatum().getHour() : 10);
    Spinner<Integer> minuteSpinner = new Spinner<>(0, 59, klausur != null ? klausur.getDatum().getMinute() : 0,

    HBox zeitBox = new HBox(5);
    zeitBox.getChildren().addAll(stundeSpinner, new Label(":"), minuteSpinner);

    DatePicker fristPicker = new DatePicker(klausur != null ? klausur.getAnmeldefrist() : LocalDate.now().plusW
    Spinner<Integer> versucheSpinner = new Spinner<>(1, 5, klausur != null ? klausur.getMaxVersuche() : 3);

    grid.add(new Label("ID:"), 0, 0);
    grid.add(idField, 1, 0);
    grid.add(new Label("Titel:"), 0, 1);
    grid.add(titelField, 1, 1);
    grid.add(new Label("Modul:"), 0, 2);
    grid.add(modulField, 1, 2);
    grid.add(new Label("Raum:"), 0, 3);
    grid.add(raumField, 1, 3);
    grid.add(new Label("Datum:"), 0, 4);
    grid.add(datumPicker, 1, 4);
    grid.add(new Label("Uhrzeit:"), 0, 5);
    grid.add(zeitBox, 1, 5);
    grid.add(new Label("Anmeldefrist:"), 0, 6);
    grid.add(fristPicker, 1, 6);
    grid.add(new Label("Max. Versuche:"), 0, 7);
    grid.add(versucheSpinner, 1, 7);

    dialog.getDialogPane().setContent(grid);

    ButtonType speichernButtonType = new ButtonType("Speichern", ButtonBar.ButtonData.OK_DONE);
    dialog.getDialogPane().getButtonTypes().addAll(speichernButtonType, ButtonType.CANCEL);

    dialog.setResultConverter(dialogButton -> {
        if (dialogButton == speichernButtonType) {
            try {
                LocalDateTime dateTime = datumPicker.getValue().atTime(
                    stundeSpinner.getValue(),
                    minuteSpinner.getValue()
                );

                if (klausur == null) {
                    return new Klausur(
                        idField.getText().trim(),
                        titelField.getText().trim(),
                        modulField.getText().trim(),
                        dateTime,
                        raumField.getText().trim(),
                        versucheSpinner.getValue(),
                        fristPicker.getValue()
                    );
                } else {
                    klausur.setDatum(dateTime);
```

```
                    klausur.setRaum(raumField.getText().trim());
                    klausur.setAnmeldefrist(fristPicker.getValue());
                    return klausur;
                }
            } catch (Exception e) {
                showError("Fehler", "Ungültige Eingabe: " + e.getMessage());
                return null;
            }
        }
        return null;
    });

    Optional<Klausur> result = dialog.showAndWait();
    result.ifPresent(k -> {
        try {
            if (klausur == null) {
                verwaltung.hinzufuegen(k);
                showInfo("Klausur hinzugefügt", "Klausur wurde erfolgreich angelegt.");
            } else {
                showInfo("Klausur aktualisiert", "Änderungen wurden gespeichert.");
            }
            aktualisiereListe();
        } catch (KlausurKonfliktException e) {
            showError("Konflikt", e.getMessage());
        }
    });
}

private void klausurLoeschen() {
    Klausur selected = tableView.getSelectionModel().getSelectedItem();
    if (selected == null) return;

    Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
    alert.setTitle("Klausur löschen");
    alert.setHeaderText("Klausur wirklich löschen?");
    alert.setContentText(selected.getTitel() + "\n\nAlle Anmeldungen werden ebenfalls gelöscht!");

    Optional<ButtonType> result = alert.showAndWait();
    if (result.isPresent() && result.get() == ButtonType.OK) {
        if (verwaltung.loeschen(selected.getId())) {
            aktualisiereListe();
            showInfo("Klausur gelöscht", "Klausur wurde erfolgreich entfernt.");
        }
    }
}

private void zeigeTeilnehmer() {
    Klausur selected = tableView.getSelectionModel().getSelectedItem();
    if (selected == null) return;

    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("Teilnehmer");
    alert.setHeaderText("Angemeldete Studenten für: " + selected.getTitel());

    StringBuilder content = new StringBuilder();
    var teilnehmer = selected.getTeilnehmendeStudenten();
    if (teilnehmer.isEmpty()) {
        content.append("Keine Studenten angemeldet.");
    } else {
        content.append("Anzahl: ").append(teilnehmer.size()).append("\n\n");
        teilnehmer.forEach(s -> content.append("• ").append(s).append("\n"));
    }

    alert.setContentText(content.toString());
    alert.showAndWait();
}

private void aktualisiereListe() {
    klausurenListe.clear();
    if (nurKommendeCheckBox.isSelected()) {
        klausurenListe.addAll(verwaltung.getKommendeKlausuren());
    } else {
        klausurenListe.addAll(verwaltung.getAlleSortiert());
    }

    // Info-Bar aktualisieren
    HBox infoBar = (HBox) getBottom();
    if (infoBar != null && infoBar.getChildren().size() >= 6) {
        Label gesamt = (Label) infoBar.getChildren().get(1);
```

```
            Label kommende = (Label) infoBar.getChildren().get(4);
            Label woche = (Label) infoBar.getChildren().get(7);
            aktualisiereInfo(gesamt, kommende, woche);
        }
    }

    private void showInfo(String title, String content) {
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(content);
        alert.showAndWait();
    }

    private void showError(String title, String content) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(content);
        alert.showAndWait();
    }
}
```

src/gui/views/NotenView.java

```
package gui.views;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import model.*;
import verwaltung.*;
import java.time.LocalDate;

/**
 * View zum Eintragen von Noten für bestehende Klausuren
 */
public class NotenView extends BorderPane {
    private final ErweiterteStudentenVerwaltung studentenVerwaltung;
    private final KlausurVerwaltung klausurVerwaltung;
    private final Runnable statistikUpdateCallback;

    private ComboBox<Student> studentComboBox;
    private ComboBox<Klausur> klausurComboBox;
    private TextField noteField;
    private TableView<Versuch> versucheTable;
    private TextArea statusArea;

    public NotenView(ErweiterteStudentenVerwaltung studentenVerwaltung, KlausurVerwaltung klausurVerwaltung,
                     Runnable statistikUpdateCallback) {
        this(studentenVerwaltung, klausurVerwaltung, statistikUpdateCallback, null);
    }

    public NotenView(ErweiterteStudentenVerwaltung studentenVerwaltung, KlausurVerwaltung klausurVerwaltung,
                     Runnable statistikUpdateCallback, Student vorausgewaehlterStudent) {
        this.studentenVerwaltung = studentenVerwaltung;
        this.klausurVerwaltung = klausurVerwaltung;
        this.statistikUpdateCallback = statistikUpdateCallback;

        setTop(createControlPanel());
        setCenter(createVersucheTable());
        setBottom(createStatusPanel());

        aktualisiereStudentenliste();
        if (vorausgewaehlterStudent != null) {
            studentComboBox.setValue(vorausgewaehlterStudent);
            aktualisiereKlausurliste();
            aktualisiereVersuchstabelle();
        }
    }

    private VBox createControlPanel() {
        VBox box = new VBox(10);
        box.setPadding(new Insets(10));
        box.setStyle("-fx-background-color: #f8f9fa;");

        Label title = new Label("Noten eintragen");
        title.setStyle("-fx-font-size: 18px; -fx-font-weight: bold;");
```

```
        HBox selection = new HBox(10);
        selection.setAlignment(javafx.geometry.Pos.CENTER_LEFT);

        studentComboBox = new ComboBox<>();
        studentComboBox.setPrefWidth(250);
        studentComboBox.setOnAction(e -> {
            aktualisiereKlausurliste();
            aktualisiereVersuchstabelle();
        });

        klausurComboBox = new ComboBox<>();
        klausurComboBox.setPrefWidth(250);

        noteField = new TextField();
        noteField.setPromptText("1.0 - 5.0");
        noteField.setPrefWidth(80);

        Button eintragenButton = new Button("Eintragen");
        eintragenButton.setOnAction(e -> versuchEintragen());

        selection.getChildren().addAll(
            new Label("Student:"), studentComboBox,
            new Label("Klausur:"), klausurComboBox,
            new Label("Note:"), noteField,
            eintragenButton
        );

        box.getChildren().addAll(title, selection);
        return box;
    }

    private TableView<Versuch> createVersucheTable() {
        versucheTable = new TableView<>();

        TableColumn<Versuch, String> klausurCol = new TableColumn<>("Klausur");
        klausurCol.setCellValueFactory(data ->
            new javafx.beans.property.SimpleStringProperty(data.getValue().getKlausur().getTitel())
        );
        klausurCol.setPrefWidth(250);

        TableColumn<Versuch, Double> noteCol = new TableColumn<>("Note");
        noteCol.setCellValueFactory(new javafx.scene.control.cell.PropertyValueFactory<>("note"));
        noteCol.setPrefWidth(80);

        TableColumn<Versuch, LocalDate> datumCol = new TableColumn<>("Datum");
        datumCol.setCellValueFactory(new javafx.scene.control.cell.PropertyValueFactory<>("datum"));
        datumCol.setPrefWidth(120);

        TableColumn<Versuch, String> bewertungCol = new TableColumn<>("Bewertung");
        bewertungCol.setCellValueFactory(data ->
            new javafx.beans.property.SimpleStringProperty(data.getValue().getBewertung())
        );
        bewertungCol.setPrefWidth(150);

        versucheTable.getColumns().addAll(klausurCol, noteCol, datumCol, bewertungCol);
        return versucheTable;
    }

    private VBox createStatusPanel() {
        VBox statusPanel = new VBox(10);
        statusPanel.setPadding(new Insets(10));
        statusPanel.setStyle("-fx-background-color: #f0f0f0;");

        Label label = new Label("Status:");
        label.setStyle("-fx-font-weight: bold;");

        statusArea = new TextArea();
        statusArea.setPrefRowCount(3);
        statusArea.setEditable(false);
        statusArea.setWrapText(true);

        statusPanel.getChildren().addAll(label, statusArea);
        return statusPanel;
    }

    private void aktualisiereStudentenliste() {
        ObservableList<Student> studenten = FXCollections.observableArrayList(
            studentenVerwaltung.getAlleSortiert(ErweiterteStudentenVerwaltung.SortierKriterium.NACHNAME)
```

```
        );
        studentComboBox.setItems(studenten);

        studentComboBox.setCellFactory(lv -> new ListCell<>() {
            @Override
            protected void updateItem(Student item, boolean empty) {
                super.updateItem(item, empty);
                setText(empty || item == null ? null :
                    item.getMatrikelnummer() + " - " + item.getNachname() + ", " + item.getVorname());
            }
        });

        studentComboBox.setButtonCell(new ListCell<>() {
            @Override
            protected void updateItem(Student item, boolean empty) {
                super.updateItem(item, empty);
                setText(empty || item == null ? null :
                    item.getMatrikelnummer() + " - " + item.getNachname() + ", " + item.getVorname());
            }
        });
    }

    private void aktualisiereKlausurliste() {
        Student student = studentComboBox.getValue();
        if (student == null) {
            klausurComboBox.getItems().clear();
            return;
        }
        ObservableList<Klausur> klausuren = FXCollections.observableArrayList(
            student.getAngemeldeteKlausuren()
        );
        klausurComboBox.setItems(klausuren);

        klausurComboBox.setCellFactory(lv -> new ListCell<>() {
            @Override
            protected void updateItem(Klausur item, boolean empty) {
                super.updateItem(item, empty);
                setText(empty || item == null ? null : item.getId() + " - " + item.getTitel());
            }
        });

        klausurComboBox.setButtonCell(new ListCell<>() {
            @Override
            protected void updateItem(Klausur item, boolean empty) {
                super.updateItem(item, empty);
                setText(empty || item == null ? null : item.getId() + " - " + item.getTitel());
            }
        });
    }

    private void aktualisiereVersuchstabelle() {
        Student student = studentComboBox.getValue();
        if (student == null) {
            versucheTable.setItems(FXCollections.observableArrayList());
            return;
        }
        versucheTable.setItems(FXCollections.observableArrayList(student.getVersuche()));
    }

    private void versuchEintragen() {
        Student student = studentComboBox.getValue();
        Klausur klausur = klausurComboBox.getValue();
        String noteText = noteField.getText();

        if (student == null || klausur == null || noteText == null || noteText.isEmpty()) {
            showError("Bitte wählen Sie Student, Klausur und Note aus.");
            return;
        }

        try {
            double note = Double.parseDouble(noteText.replace(',', '.'));
            studentenVerwaltung.versuchEintragen(student.getMatrikelnummer(), klausur, note, LocalDate.now());
            statusArea.appendText(String.format("\n✓ Note %.1f für %s in %s eingetragen.", note,
                student.getNachname(), klausur.getTitel()));
            noteField.clear();
            aktualisiereVersuchstabelle();
            if (statistikUpdateCallback != null) {
                statistikUpdateCallback.run();
            }
```

```
        } catch (NumberFormatException ex) {
            showError("Ungültige Note. Bitte eine Zahl zwischen 1.0 und 5.0 eingeben.");
        } catch (Exception ex) {
            showError("Fehler beim Eintragen: " + ex.getMessage());
        }
    }

    private void showError(String message) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Fehler");
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }
}
```

src/gui/views/StatistikView.java

```
package gui.views;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.geometry.Side;
import javafx.scene.chart.*;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import model.*;
import verwaltung.*;
import java.util.*;
import java.util.stream.Collectors;

public class StatistikView extends BorderPane {
    private ErweiterteStudentenVerwaltung studentenVerwaltung;
    private KlausurVerwaltung klausurVerwaltung;
    private TabPane tabPane;

    public StatistikView(ErweiterteStudentenVerwaltung studentenVerwaltung, KlausurVerwaltung klausurVerwaltung) {
        this.studentenVerwaltung = studentenVerwaltung;
        this.klausurVerwaltung = klausurVerwaltung;

        Label titleLabel = new Label("Statistiken und Auswertungen");
        titleLabel.setStyle("-fx-font-size: 20px; -fx-font-weight: bold; -fx-padding: 10;");
        setTop(titleLabel);

        tabPane = new TabPane();
        tabPane.getTabs().addAll(
            createUebersichtTab(),
            createNotenverteilungTab(),
            createStudiengangTab(),
            createKlausurstatistikTab(),
            createLeistungsTab()
        );

        setCenter(tabPane);
    }

    /**
     * Aktualisiert alle Statistik-Tabs nach Datenänderungen
     */
    public void refresh() {
        tabPane.getTabs().setAll(
            createUebersichtTab(),
            createNotenverteilungTab(),
            createStudiengangTab(),
            createKlausurstatistikTab(),
            createLeistungsTab()
        );
    }

    private Tab createUebersichtTab() {
        Tab tab = new Tab("Übersicht");
        tab.setClosable(false);

        GridPane grid = new GridPane();
        grid.setHgap(20);
        grid.setVgap(20);
        grid.setPadding(new Insets(20));
```

```
        // Statistik-Boxen
        VBox studentenBox = createStatBox("Studenten",
            String.valueOf(studentenVerwaltung.getAlleSortiert(ErweiterteStudentenVerwaltung.SortierKriterium.NACHN
            "Gesamt eingeschrieben");

        VBox klausurenBox = createStatBox("Klausuren",
            String.valueOf(klausurVerwaltung.getAlleSortiert().size()),
            "Im System");

        VBox kommendeBox = createStatBox("Anstehend",
            String.valueOf(klausurVerwaltung.getKommendeKlausuren().size()),
            "Kommende Klausuren");

        // Berechne Durchschnittsnote
        double avgNote = studentenVerwaltung.getAlleSortiert(ErweiterteStudentenVerwaltung.SortierKriterium.NACHNAM
            .mapToDouble(Student::berechneNotendurchschnitt)
            .filter(note -> note > 0)
            .average()
            .orElse(0.0);

        VBox durchschnittBox = createStatBox("Ø-Note",
            String.format("%.2f", avgNote),
            "Gesamtdurchschnitt");

        grid.add(studentenBox, 0, 0);
        grid.add(klausurenBox, 1, 0);
        grid.add(kommendeBox, 2, 0);
        grid.add(durchschnittBox, 3, 0);

        // Schnellübersicht Liste
        VBox quickStats = new VBox(10);
        quickStats.setPadding(new Insets(20, 0, 0, 0));

        Label quickLabel = new Label("Schnellübersicht");
        quickLabel.setStyle("-fx-font-size: 16px; -fx-font-weight: bold;");

        ListView<String> quickList = new ListView<>();
        ObservableList<String> items = FXCollections.observableArrayList();

        // Studiengänge mit Anzahl
        Map<String, Long> studiengangCount = studentenVerwaltung.getAlleSortiert(ErweiterteStudentenVerwaltung.Sort
            .stream()
            .collect(Collectors.groupingBy(Student::getStudiengang, Collectors.counting()));

        items.add("=== Studiengänge ===");
        studiengangCount.forEach((studiengang, count) ->
            items.add(String.format("%s: %d Studenten", studiengang, count))
        );

        items.add("");
        items.add("=== Klausuren diese Woche ===");
        klausurVerwaltung.getKommendeKlausuren().stream()
            .filter(k -> k.getDatum().isBefore(java.time.LocalDateTime.now().plusWeeks(1)))
            .forEach(k -> items.add(k.getTitel() + " - " + k.getDatum().format(java.time.format.DateTimeFormatter.o

        quickList.setItems(items);
        quickList.setPrefHeight(300);

        quickStats.getChildren().addAll(quickLabel, quickList);

        VBox content = new VBox(20);
        content.getChildren().addAll(grid, quickStats);

        ScrollPane scrollPane = new ScrollPane(content);
        scrollPane.setFitToWidth(true);
        tab.setContent(scrollPane);

        return tab;
    }

    private Tab createNotenverteilungTab() {
        Tab tab = new Tab("Notenverteilung");
        tab.setClosable(false);

        VBox content = new VBox(20);
        content.setPadding(new Insets(20));

        // Notenverteilung Histogram
        CategoryAxis xAxis = new CategoryAxis();
```

```
xAxis.setLabel("Notenbereiche");
NumberAxis yAxis = new NumberAxis();
yAxis.setLabel("Anzahl Studenten");

BarChart<String, Number> barChart = new BarChart<>(xAxis, yAxis);
barChart.setTitle("Notenverteilung aller Studenten");

// Daten sammeln
Map<String, Integer> notenVerteilung = new TreeMap<>();
notenVerteilung.put("1.0-1.5", 0);
notenVerteilung.put("1.6-2.0", 0);
notenVerteilung.put("2.1-2.5", 0);
notenVerteilung.put("2.6-3.0", 0);
notenVerteilung.put("3.1-3.5", 0);
notenVerteilung.put("3.6-4.0", 0);
notenVerteilung.put("> 4.0", 0);
notenVerteilung.put("Keine Note", 0);

for (Student student : studentenVerwaltung.getAlleSortiert(ErweiterteStudentenVerwaltung.SortierKriterium.NA
    double durchschnitt = student.berechneNotendurchschnitt();
    if (durchschnitt == 0) {
        notenVerteilung.merge("Keine Note", 1, Integer::sum);
    } else if (durchschnitt <= 1.5) {
        notenVerteilung.merge("1.0-1.5", 1, Integer::sum);
    } else if (durchschnitt <= 2.0) {
        notenVerteilung.merge("1.6-2.0", 1, Integer::sum);
    } else if (durchschnitt <= 2.5) {
        notenVerteilung.merge("2.1-2.5", 1, Integer::sum);
    } else if (durchschnitt <= 3.0) {
        notenVerteilung.merge("2.6-3.0", 1, Integer::sum);
    } else if (durchschnitt <= 3.5) {
        notenVerteilung.merge("3.1-3.5", 1, Integer::sum);
    } else if (durchschnitt <= 4.0) {
        notenVerteilung.merge("3.6-4.0", 1, Integer::sum);
    } else {
        notenVerteilung.merge("> 4.0", 1, Integer::sum);
    }
}

XYChart.Series<String, Number> series = new XYChart.Series<>();
series.setName("Anzahl Studenten");
notenVerteilung.forEach((bereich, anzahl) ->
    series.getData().add(new XYChart.Data<>(bereich, anzahl))
);

barChart.getData().add(series);
barChart.setPrefHeight(400);

// Statistik-Text
TextArea statsText = new TextArea();
statsText.setEditable(false);
statsText.setPrefRowCount(8);

StringBuilder stats = new StringBuilder();
stats.append("=== DETAILLIERTE NOTENSTATISTIK ===\n\n");

List<Student> mitNoten = studentenVerwaltung.getAlleSortiert(ErweiterteStudentenVerwaltung.SortierKriterium
    .stream()
    .filter(s -> s.berechneNotendurchschnitt() > 0)
    .sorted(Comparator.comparing(Student::berechneNotendurchschnitt))
    .collect(Collectors.toList());

if (!mitNoten.isEmpty()) {
    stats.append(String.format("Beste Note: %.2f (%s)\n",
        mitNoten.get(0).berechneNotendurchschnitt(),
        mitNoten.get(0).getNachname() + ", " + mitNoten.get(0).getVorname()));

    stats.append(String.format("Schlechteste Note: %.2f (%s)\n",
        mitNoten.get(mitNoten.size()-1).berechneNotendurchschnitt(),
        mitNoten.get(mitNoten.size()-1).getNachname() + ", " + mitNoten.get(mitNoten.size()-1).getVorname()

    double median = mitNoten.size() % 2 == 0 ?
        (mitNoten.get(mitNoten.size()/2-1).berechneNotendurchschnitt() +
         mitNoten.get(mitNoten.size()/2).berechneNotendurchschnitt()) / 2 :
        mitNoten.get(mitNoten.size()/2).berechneNotendurchschnitt();

    stats.append(String.format("Median: %.2f\n", median));
    stats.append(String.format("Studenten mit Noten: %d von %d\n",
        mitNoten.size(),
```

```
                studentenVerwaltung.getAlleSortiert(ErweiterteStudentenVerwaltung.SortierKriterium.NACHNAME).size()
        }

        statsText.setText(stats.toString());

        content.getChildren().addAll(barChart, statsText);

        ScrollPane scrollPane = new ScrollPane(content);
        scrollPane.setFitToWidth(true);
        tab.setContent(scrollPane);

        return tab;
    }

    private Tab createStudiengangTab() {
        Tab tab = new Tab("Studiengänge");
        tab.setClosable(false);

        VBox content = new VBox(20);
        content.setPadding(new Insets(20));

        // Pie Chart für Studiengangverteilung
        PieChart pieChart = new PieChart();
        pieChart.setTitle("Verteilung nach Studiengängen");

        Map<String, Long> studiengangCount = studentenVerwaltung.getAlleSortiert(ErweiterteStudentenVerwaltung.Sort
            .stream()
            .collect(Collectors.groupingBy(Student::getStudiengang, Collectors.counting()));

        studiengangCount.forEach((studiengang, count) -> {
            PieChart.Data slice = new PieChart.Data(studiengang + " (" + count + ")", count);
            pieChart.getData().add(slice);
        });

        pieChart.setLabelLineLength(10);
        pieChart.setLegendSide(Side.RIGHT);
        pieChart.setPrefHeight(400);

        // Tabelle mit Details pro Studiengang
        TableView<StudiengangStatistik> table = new TableView<>();

        TableColumn<StudiengangStatistik, String> studiengangCol = new TableColumn<>("Studiengang");
        studiengangCol.setCellValueFactory(data -> new javafx.beans.property.SimpleStringProperty(data.getValue().s
        studiengangCol.setPrefWidth(200);

        TableColumn<StudiengangStatistik, Integer> anzahlCol = new TableColumn<>("Anzahl");
        anzahlCol.setCellValueFactory(data -> new javafx.beans.property.SimpleIntegerProperty(data.getValue().anzah
        anzahlCol.setPrefWidth(100);

        TableColumn<StudiengangStatistik, Double> durchschnittCol = new TableColumn<>("Ø-Note");
        durchschnittCol.setCellValueFactory(data -> new javafx.beans.property.SimpleDoubleProperty(data.getValue().
        durchschnittCol.setPrefWidth(100);
        durchschnittCol.setCellFactory(col -> new TableCell<StudiengangStatistik, Double>() {
            @Override
            protected void updateItem(Double item, boolean empty) {
                super.updateItem(item, empty);
                if (empty || item == null || item == 0.0) {
                    setText("-");
                } else {
                    setText(String.format("%.2f", item));
                }
            }
        });

        TableColumn<StudiengangStatistik, Integer> mitNotenCol = new TableColumn<>("Mit Noten");
        mitNotenCol.setCellValueFactory(data -> new javafx.beans.property.SimpleIntegerProperty(data.getValue().mit
        mitNotenCol.setPrefWidth(100);

        table.getColumns().addAll(studiengangCol, anzahlCol, durchschnittCol, mitNotenCol);

        // Daten für Tabelle sammeln
        ObservableList<StudiengangStatistik> tableData = FXCollections.observableArrayList();

        studiengangCount.forEach((studiengang, count) -> {
            List<Student> studiengangStudenten = studentenVerwaltung.suchenNachStudiengang(studiengang);
            double avgNote = studiengangStudenten.stream()
                .mapToDouble(Student::berechneNotendurchschnitt)
                .filter(note -> note > 0)
                .average()
```

```
                .orElse(0.0);
            int mitNoten = (int) studiengangStudenten.stream()
                .filter(s -> s.berechneNotendurchschnitt() > 0)
                .count();

            tableData.add(new StudiengangStatistik(studiengang, count.intValue(), avgNote, mitNoten));
        });

        table.setItems(tableData);
        table.setPrefHeight(200);

        content.getChildren().addAll(pieChart, new Label("Details pro Studiengang:"), table);

        ScrollPane scrollPane = new ScrollPane(content);
        scrollPane.setFitToWidth(true);
        tab.setContent(scrollPane);

        return tab;
    }

    private Tab createKlausurstatistikTab() {
        Tab tab = new Tab("Klausurstatistiken");
        tab.setClosable(false);

        VBox content = new VBox(20);
        content.setPadding(new Insets(20));

        Label selectLabel = new Label("Klausur auswählen:");
        ComboBox<Klausur> klausurBox = new ComboBox<>();
        klausurBox.setItems(FXCollections.observableArrayList(klausurVerwaltung.getAlleSortiert()));
        klausurBox.setPrefWidth(400);
        klausurBox.setCellFactory(lv -> new ListCell<Klausur>() {
            @Override
            protected void updateItem(Klausur item, boolean empty) {
                super.updateItem(item, empty);
                setText(empty || item == null ? null : item.getId() + " - " + item.getTitel());
            }
        });
        klausurBox.setButtonCell(new ListCell<Klausur>() {
            @Override
            protected void updateItem(Klausur item, boolean empty) {
                super.updateItem(item, empty);
                setText(empty || item == null ? null : item.getId() + " - " + item.getTitel());
            }
        });

        VBox statistikBox = new VBox(10);
        statistikBox.setPadding(new Insets(20));
        statistikBox.setStyle("-fx-border-color: #ddd; -fx-border-radius: 5; -fx-background-color: #f9f9f9;");

        klausurBox.setOnAction(e -> {
            Klausur selected = klausurBox.getValue();
            if (selected != null) {
                aktualisiereKlausurStatistik(selected, statistikBox);
            }
        });

        content.getChildren().addAll(selectLabel, klausurBox, statistikBox);

        tab.setContent(content);
        return tab;
    }

    private Tab createLeistungsTab() {
        Tab tab = new Tab("Leistungsübersicht");
        tab.setClosable(false);

        VBox content = new VBox(20);
        content.setPadding(new Insets(20));

        Label titleLabel = new Label("Studenten nach Leistung");
        titleLabel.setStyle("-fx-font-size: 16px; -fx-font-weight: bold;");

        // Tabs für verschiedene Kategorien
        TabPane leistungTabs = new TabPane();

        // Beste Studenten
        Tab besteTab = new Tab("Beste Studenten");
        besteTab.setClosable(false);
```

```
        ListView<String> besteList = new ListView<>();

        List<Student> besteStudenten = studentenVerwaltung.getAlleSortiert(ErweiterteStudentenVerwaltung.SortierKri
            .stream()
            .filter(s -> s.berechneNotendurchschnitt() > 0 && s.berechneNotendurchschnitt() <= 2.0)
            .sorted(Comparator.comparing(Student::berechneNotendurchschnitt))
            .limit(10)
            .collect(Collectors.toList());

        ObservableList<String> besteItems = FXCollections.observableArrayList();
        besteStudenten.forEach(s ->
            besteItems.add(String.format("%.2f - %s (%s)",
                s.berechneNotendurchschnitt(),
                s.getNachname() + ", " + s.getVorname(),
                s.getStudiengang()))
        );

        besteList.setItems(besteItems);
        besteTab.setContent(besteList);

        // Gefährdete Studenten
        Tab gefaehrdeteTab = new Tab("Gefährdete Studenten");
        gefaehrdeteTab.setClosable(false);
        ListView<String> gefaehrdeteList = new ListView<>();

        List<Student> gefaehrdeteStudenten = studentenVerwaltung.getStudentenMitSchlechtenLeistungen();

        ObservableList<String> gefaehrdeteItems = FXCollections.observableArrayList();
        gefaehrdeteStudenten.forEach(s ->
            gefaehrdeteItems.add(String.format("%.2f - %s (%s) - WARNUNG",
                s.berechneNotendurchschnitt(),
                s.getNachname() + ", " + s.getVorname(),
                s.getStudiengang()))
        );

        gefaehrdeteList.setItems(gefaehrdeteItems);
        gefaehrdeteList.setStyle("-fx-control-inner-background: #fff5f5;");
        gefaehrdeteTab.setContent(gefaehrdeteList);

        leistungTabs.getTabs().addAll(besteTab, gefaehrdeteTab);

        content.getChildren().addAll(titleLabel, leistungTabs);

        tab.setContent(content);
        return tab;
    }

    private VBox createStatBox(String title, String value, String subtitle) {
        VBox box = new VBox(5);
        box.setAlignment(javafx.geometry.Pos.CENTER);
        box.setPadding(new Insets(20));
        box.setPrefSize(200, 100);
        box.setStyle("-fx-background-color: white; -fx-border-color: #ddd; -fx-border-radius: 5;");

        Label titleLabel = new Label(title);
        titleLabel.setStyle("-fx-font-size: 14px; -fx-text-fill: #666;");

        Label valueLabel = new Label(value);
        valueLabel.setStyle("-fx-font-size: 36px; -fx-font-weight: bold;");

        Label subtitleLabel = new Label(subtitle);
        subtitleLabel.setStyle("-fx-font-size: 12px; -fx-text-fill: #999;");

        box.getChildren().addAll(titleLabel, valueLabel, subtitleLabel);

        return box;
    }

    private void aktualisiereKlausurStatistik(Klausur klausur, VBox container) {
        container.getChildren().clear();

        VersuchsVerwaltung.KlausurStatistik stats =
            studentenVerwaltung.getVersuchsVerwaltung().berechneStatistik(klausur);

        Label klausurLabel = new Label(klausur.getTitel());
        klausurLabel.setStyle("-fx-font-size: 18px; -fx-font-weight: bold;");

        GridPane grid = new GridPane();
        grid.setHgap(20);
```

```
        grid.setVgap(10);

        grid.add(new Label("Gesamtversuche:"), 0, 0);
        grid.add(new Label(String.valueOf(stats.gesamtVersuche)), 1, 0);

        grid.add(new Label("Bestanden:"), 0, 1);
        grid.add(new Label(stats.bestanden + " (" + String.format("%.1f%%", stats.bestehendenquote) + ")"), 1, 1);

        grid.add(new Label("Nicht bestanden:"), 0, 2);
        grid.add(new Label(String.valueOf(stats.nichtBestanden)), 1, 2);

        grid.add(new Label("Durchschnittsnote:"), 0, 3);
        Label avgLabel = new Label(stats.durchschnittsnote > 0 ? String.format("%.2f", stats.durchschnittsnote) : "
        if (stats.durchschnittsnote > 0 && stats.durchschnittsnote <= 2.0) {
            avgLabel.setStyle("-fx-text-fill: green; -fx-font-weight: bold;");
        } else if (stats.durchschnittsnote > 3.0) {
            avgLabel.setStyle("-fx-text-fill: red; -fx-font-weight: bold;");
        }
        grid.add(avgLabel, 1, 3);

        grid.add(new Label("Angemeldete Studenten:"), 0, 4);
        grid.add(new Label(String.valueOf(klausur.getTeilnehmendeStudenten().size())), 1, 4);

        container.getChildren().addAll(klausurLabel, grid);

        if (stats.gesamtVersuche > 0) {
            // Notenverteilung für diese Klausur
            BarChart<String, Number> notenChart = createKlausurNotenChart(klausur);
            notenChart.setPrefHeight(300);
            container.getChildren().add(notenChart);
        }
    }

    private BarChart<String, Number> createKlausurNotenChart(Klausur klausur) {
        CategoryAxis xAxis = new CategoryAxis();
        xAxis.setLabel("Note");
        NumberAxis yAxis = new NumberAxis();
        yAxis.setLabel("Anzahl");

        BarChart<String, Number> chart = new BarChart<>(xAxis, yAxis);
        chart.setTitle("Notenverteilung");
        chart.setLegendVisible(false);

        Map<String, Integer> notenCount = new TreeMap<>();

        studentenVerwaltung.getVersuchsVerwaltung().getVersucheFuerKlausur(klausur)
            .forEach(v -> {
                String noteStr = String.format("%.1f", v.getNote());
                notenCount.merge(noteStr, 1, Integer::sum);
            });

        XYChart.Series<String, Number> series = new XYChart.Series<>();
        notenCount.forEach((note, count) ->
            series.getData().add(new XYChart.Data<>(note, count))
        );

        chart.getData().add(series);

        return chart;
    }

    // Hilfsklasse für Studiengangstatistik
    private static class StudiengangStatistik {
        final String studiengang;
        final int anzahl;
        final double durchschnitt;
        final int mitNoten;

        StudiengangStatistik(String studiengang, int anzahl, double durchschnitt, int mitNoten) {
            this.studiengang = studiengang;
            this.anzahl = anzahl;
            this.durchschnitt = durchschnitt;
            this.mitNoten = mitNoten;
        }
    }
}
```

src/gui/views/StudentenView.java

```
package gui.views;
```

```
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.*;
import model.Student;
import verwaltung.ErweiterteStudentenVerwaltung;
import verwaltung.ErweiterteStudentenVerwaltung.DuplikatException;
import verwaltung.ErweiterteStudentenVerwaltung.ValidationResult;
import java.time.LocalDate;
import java.util.Optional;
import java.util.function.Consumer;

public class StudentenView extends BorderPane {
    private ErweiterteStudentenVerwaltung verwaltung;
    private TableView<Student> tableView;
    private ObservableList<Student> studentenListe;
    private TextField suchfeld;
    private ComboBox<String> suchkriteriumBox;
    private Label statistikLabel;
    private final Consumer<Student> notenHandler;

    public StudentenView(ErweiterteStudentenVerwaltung verwaltung, Consumer<Student> notenHandler) {
        this.verwaltung = verwaltung;
        this.notenHandler = notenHandler;
        this.studentenListe = FXCollections.observableArrayList();

        setTop(createToolBar());
        setCenter(createTableView());
        setBottom(createStatistikBar());

        aktualisiereListe();
    }

    private ToolBar createToolBar() {
        ToolBar toolBar = new ToolBar();

        // Buttons
        Button neuButton = new Button("Neuer Student");
        neuButton.setOnAction(e -> zeigeStudentDialog(null));

        Button bearbeitenButton = new Button("Bearbeiten");
        bearbeitenButton.setOnAction(e -> {
            Student selected = tableView.getSelectionModel().getSelectedItem();
            if (selected != null) {
                zeigeStudentDialog(selected);
            }
        });

        Button loeschenButton = new Button("Löschen");
        loeschenButton.setOnAction(e -> studentLoeschen());

        Button notenButton = new Button("Note eintragen");
        notenButton.setOnAction(e -> {
            Student selected = tableView.getSelectionModel().getSelectedItem();
            if (selected != null && notenHandler != null) {
                notenHandler.accept(selected);
            }
        });

        // Suche
        suchfeld = new TextField();
        suchfeld.setPromptText("Suchen...");
        suchfeld.textProperty().addListener((obs, alt, neu) -> sucheStudenten());

        suchkriteriumBox = new ComboBox<>();
        suchkriteriumBox.getItems().addAll("Name", "Matrikelnummer", "Studiengang");
        suchkriteriumBox.setValue("Name");
        suchkriteriumBox.setOnAction(e -> sucheStudenten());

        Button aktualisierenButton = new Button("Aktualisieren");
        aktualisierenButton.setOnAction(e -> aktualisiereListe());

        toolBar.getItems().addAll(
            neuButton, bearbeitenButton, loeschenButton, notenButton,
            new Separator(),
            new Label("Suche:"), suchkriteriumBox, suchfeld,
```

```
        new Separator(),
        aktualisierenButton
    );

    return toolBar;
}

private TableView<Student> createTableView() {
    tableView = new TableView<>();

    // Spalten definieren
    TableColumn<Student, String> matrikelCol = new TableColumn<>("Matrikelnummer");
    matrikelCol.setCellValueFactory(new PropertyValueFactory<>("matrikelnummer"));
    matrikelCol.setPrefWidth(120);

    TableColumn<Student, String> vornameCol = new TableColumn<>("Vorname");
    vornameCol.setCellValueFactory(new PropertyValueFactory<>("vorname"));
    vornameCol.setPrefWidth(150);

    TableColumn<Student, String> nachnameCol = new TableColumn<>("Nachname");
    nachnameCol.setCellValueFactory(new PropertyValueFactory<>("nachname"));
    nachnameCol.setPrefWidth(150);

    TableColumn<Student, String> studiengangCol = new TableColumn<>("Studiengang");
    studiengangCol.setCellValueFactory(new PropertyValueFactory<>("studiengang"));
    studiengangCol.setPrefWidth(200);

    TableColumn<Student, LocalDate> geburtsdatumCol = new TableColumn<>("Geburtsdatum");
    geburtsdatumCol.setCellValueFactory(new PropertyValueFactory<>("geburtsdatum"));
    geburtsdatumCol.setPrefWidth(120);

    TableColumn<Student, Double> durchschnittCol = new TableColumn<>("Notendurchschnitt");
    durchschnittCol.setCellValueFactory(cellData ->
        new javafx.beans.property.SimpleDoubleProperty(cellData.getValue().berechneNotendurchschnitt()).asObjec
    );
    durchschnittCol.setPrefWidth(130);
    durchschnittCol.setCellFactory(col -> new TableCell<Student, Double>() {
        @Override
        protected void updateItem(Double item, boolean empty) {
            super.updateItem(item, empty);
            if (empty || item == null || item == 0.0) {
                setText("-");
            } else {
                setText(String.format("%.2f", item));
                if (item > 3.0) {
                    setStyle("-fx-text-fill: red;");
                } else if (item < 2.0) {
                    setStyle("-fx-text-fill: green;");
                } else {
                    setStyle("");
                }
            }
        }
    });

    tableView.getColumns().addAll(matrikelCol, vornameCol, nachnameCol, studiengangCol, geburtsdatumCol, durchs
    tableView.setItems(studentenListe);

    // Doppelklick zum Bearbeiten
    tableView.setRowFactory(tv -> {
        TableRow<Student> row = new TableRow<>();
        row.setOnMouseClicked(event -> {
            if (event.getClickCount() == 2 && !row.isEmpty()) {
                zeigeStudentDialog(row.getItem());
            }
        });
        return row;
    });

    return tableView;
}

private HBox createStatistikBar() {
    HBox statistikBar = new HBox(10);
    statistikBar.setPadding(new Insets(10));
    statistikBar.setStyle("-fx-background-color: #f0f0f0;");

    statistikLabel = new Label();
    aktualisiereStatistik();
```

```
    statistikBar.getChildren().add(statistikLabel);
    return statistikBar;
}

private void zeigeStudentDialog(Student student) {
    Dialog<Student> dialog = new Dialog<>();
    dialog.setTitle(student == null ? "Neuer Student" : "Student bearbeiten");
    dialog.setHeaderText(student == null ? "Neuen Studenten anlegen" : "Studentendaten bearbeiten");

    // Form
    GridPane grid = new GridPane();
    grid.setHgap(10);
    grid.setVgap(10);
    grid.setPadding(new Insets(20, 150, 10, 10));

    TextField matrikelField = new TextField(student != null ? student.getMatrikelnummer() : "");
    matrikelField.setPromptText("5-8 Ziffern");
    matrikelField.setDisable(student != null); // Matrikelnummer nicht änderbar

    TextField vornameField = new TextField(student != null ? student.getVorname() : "");
    TextField nachnameField = new TextField(student != null ? student.getNachname() : "");
    TextField studiengangField = new TextField(student != null ? student.getStudiengang() : "");
    DatePicker geburtsdatumPicker = new DatePicker(student != null ? student.getGeburtsdatum() : null);

    Label validierungLabel = new Label();
    validierungLabel.setStyle("-fx-text-fill: red;");

    // Validierung bei Eingabe
    matrikelField.textProperty().addListener((obs, alt, neu) -> {
        if (student == null && !neu.isEmpty()) {
            ValidationResult result = verwaltung.validiereMatrikelnummer(neu);
            if (!result.isValid) {
                validierungLabel.setText(result.message);
                matrikelField.setStyle("-fx-border-color: red;");
            } else {
                validierungLabel.setText("");
                matrikelField.setStyle("");
            }
        }
    });

    grid.add(new Label("Matrikelnummer:"), 0, 0);
    grid.add(matrikelField, 1, 0);
    grid.add(new Label("Vorname:"), 0, 1);
    grid.add(vornameField, 1, 1);
    grid.add(new Label("Nachname:"), 0, 2);
    grid.add(nachnameField, 1, 2);
    grid.add(new Label("Studiengang:"), 0, 3);
    grid.add(studiengangField, 1, 3);
    grid.add(new Label("Geburtsdatum:"), 0, 4);
    grid.add(geburtsdatumPicker, 1, 4);
    grid.add(validierungLabel, 0, 5, 2, 1);

    dialog.getDialogPane().setContent(grid);

    // Buttons
    ButtonType speichernButtonType = new ButtonType("Speichern", ButtonBar.ButtonData.OK_DONE);
    dialog.getDialogPane().getButtonTypes().addAll(speichernButtonType, ButtonType.CANCEL);

    // Result converter
    dialog.setResultConverter(dialogButton -> {
        if (dialogButton == speichernButtonType) {
            try {
                if (student == null) {
                    // Neuer Student
                    return new Student(
                        matrikelField.getText().trim(),
                        vornameField.getText().trim(),
                        nachnameField.getText().trim(),
                        studiengangField.getText().trim(),
                        geburtsdatumPicker.getValue()
                    );
                } else {
                    // Bestehender Student bearbeiten
                    student.setVorname(vornameField.getText().trim());
                    student.setNachname(nachnameField.getText().trim());
                    student.setStudiengang(studiengangField.getText().trim());
                    return student;
```

```
                }
            } catch (IllegalArgumentException e) {
                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Fehler");
                alert.setHeaderText("Ungültige Eingabe");
                alert.setContentText(e.getMessage());
                alert.showAndWait();
                return null;
            }
        }
        return null;
    });

    Optional<Student> result = dialog.showAndWait();
    result.ifPresent(s -> {
        if (student == null) {
            // Neuen Studenten hinzufügen
            try {
                verwaltung.hinzufuegenMitValidierung(s);
                aktualisiereListe();
                showInfo("Student hinzugefügt", "Student wurde erfolgreich angelegt.");
            } catch (DuplikatException e) {
                showError("Fehler", e.getMessage());
            }
        } else {
            // Student wurde bearbeitet
            aktualisiereListe();
            showInfo("Student aktualisiert", "Änderungen wurden gespeichert.");
        }
    });
}

private void studentLoeschen() {
    Student selected = tableView.getSelectionModel().getSelectedItem();
    if (selected == null) return;

    Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
    alert.setTitle("Student löschen");
    alert.setHeaderText("Student wirklich löschen?");
    alert.setContentText(selected.toString() + "\n\nDieser Vorgang kann nicht rückgängig gemacht werden!");

    Optional<ButtonType> result = alert.showAndWait();
    if (result.isPresent() && result.get() == ButtonType.OK) {
        if (verwaltung.loeschen(selected.getMatrikelnummer())) {
            aktualisiereListe();
            showInfo("Student gelöscht", "Student wurde erfolgreich entfernt.");
        }
    }
}

private void sucheStudenten() {
    String suchtext = suchfeld.getText().trim();
    if (suchtext.isEmpty()) {
        aktualisiereListe();
        return;
    }

    studentenListe.clear();
    String kriterium = suchkriteriumBox.getValue();

    switch (kriterium) {
        case "Name":
            studentenListe.addAll(verwaltung.suchenNachName(suchtext));
            break;
        case "Matrikelnummer":
            Student student = verwaltung.findeNachMatrikelnummer(suchtext);
            if (student != null) {
                studentenListe.add(student);
            }
            break;
        case "Studiengang":
            studentenListe.addAll(verwaltung.suchenNachStudiengang(suchtext));
            break;
    }

    aktualisiereStatistik();
}

private void aktualisiereListe() {
```

```
        studentenListe.clear();
        studentenListe.addAll(verwaltung.getAlleSortiert(ErweiterteStudentenVerwaltung.SortierKriterium.NACHNAME));
        aktualisiereStatistik();
    }

    private void aktualisiereStatistik() {
        var stats = verwaltung.getMatrikelnummerStatistik();
        statistikLabel.setText(String.format(
            "Studenten: %d | Matrikelnummern: %s - %s | Durchschnittliche Länge: %.1f",
            studentenListe.size(),
            stats.niedrigste.isEmpty() ? "keine" : stats.niedrigste,
            stats.hoechste.isEmpty() ? "keine" : stats.hoechste,
            stats.durchschnittlicheLaenge
        ));
    }

    private void showInfo(String title, String content) {
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(content);
        alert.showAndWait();
    }

    private void showError(String title, String content) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(content);
        alert.showAndWait();
    }
}
```

src/util/Database.java

```
package util;

import java.sql.*;

public class Database {
    private static final String DB_URL = "jdbc:sqlite:klausurverwaltung.db";

    static {
        try {
            // Lade den SQLite Treiber explizit
            Class.forName("org.sqlite.JDBC");
            System.out.println("■ SQLite Treiber geladen!");
        } catch (ClassNotFoundException e) {
            System.err.println("■ SQLite JDBC Treiber nicht gefunden!");
            System.err.println("Stelle sicher, dass sqlite-jdbc.jar im Classpath ist.");
        }
    }

    public static Connection connect() throws SQLException {
        return DriverManager.getConnection(DB_URL);
    }
}
```