# Gradify: A Student Grading Application

## 1. Introduction

### 1.1 Project Overview

Gradify is a desktop student grading application built using the Tauri framework with a React/TypeScript frontend and Rust backend. It provides a solution for educators to track student performance across classes and assignment types. The application uses SQLite for data storage and implements a complete model for students, classes, assignments, and grades.

### 1.2 Purpose and Objectives

Gradify aims to streamline grade management by:

- Managing student information and class enrollments
- Categorizing assignments as homework or tests
- Recording scores with validation
- Calculating overall grades with percentages and letter grades
- Identifying at-risk students

Technical objectives included implementing OOP principles, creating a cross-platform application, designing an intuitive interface, ensuring data persistence, and developing a streamlined installation process.

### 1.3 Implementation Summary

The application is built on:

Frontend: React with TypeScript
Backend: Rust with SQLx and Tauri
Database: SQLite
Installation: NSIS for Windows deployment

Key features include CRUD operations for all entities, grade calculation with validation, enrollment management, a comprehensive dashboard, dark/light mode, search functionality, and form validation for data integrity.

## 2. User Manual

### 2.1 Installation Guide

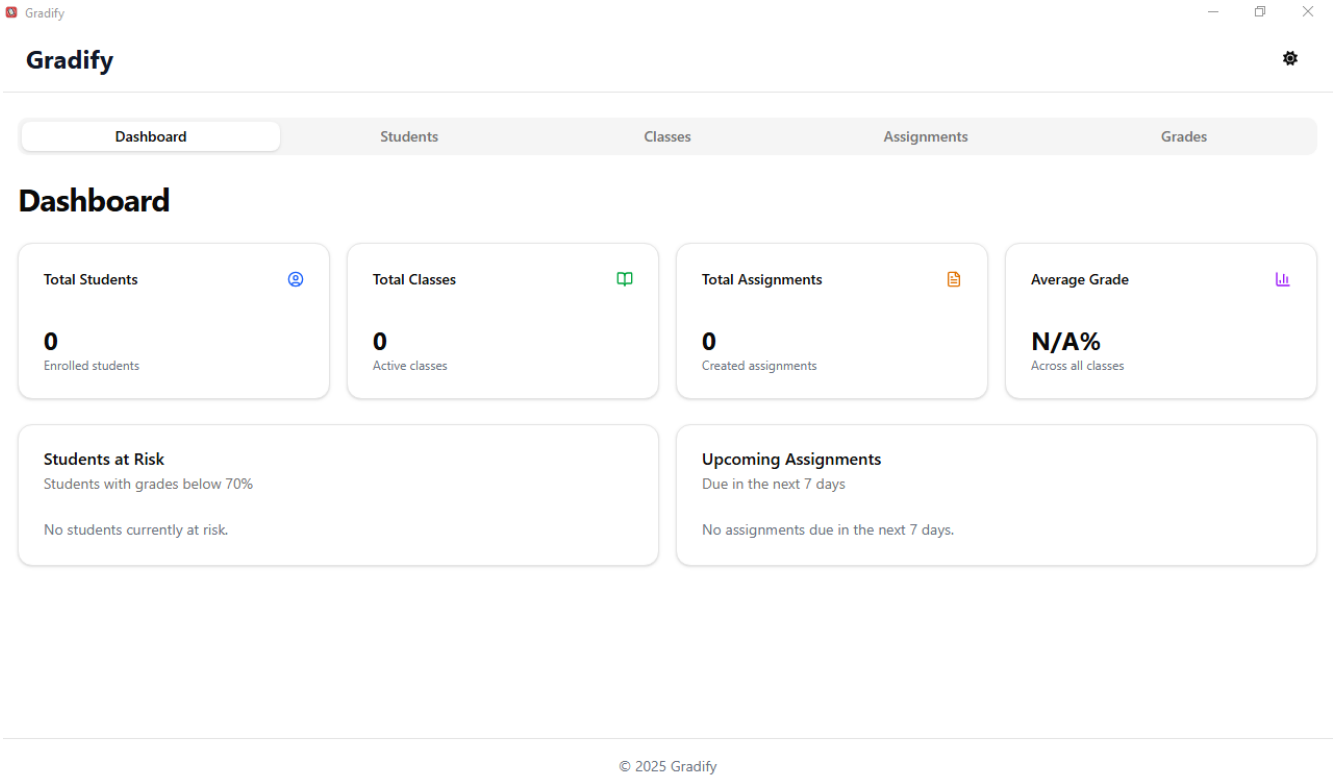
1. Download the Gradify installer (gradify_1.0.1_x64-setup.exe)
2. Run the installer - application will install to C:\Gradify
3. A desktop shortcut will be created automatically for all users
4. Launch Gradify using the desktop shortcut

## 2.2 Application Overview

Gradify provides a tab-based interface with five main sections:

Dashboard: Statistics overview and at-risk students
Students: Manage student records
Classes: Create and edit class information
Assignments: Define homework and test assignments
Grades: Record and view student grades

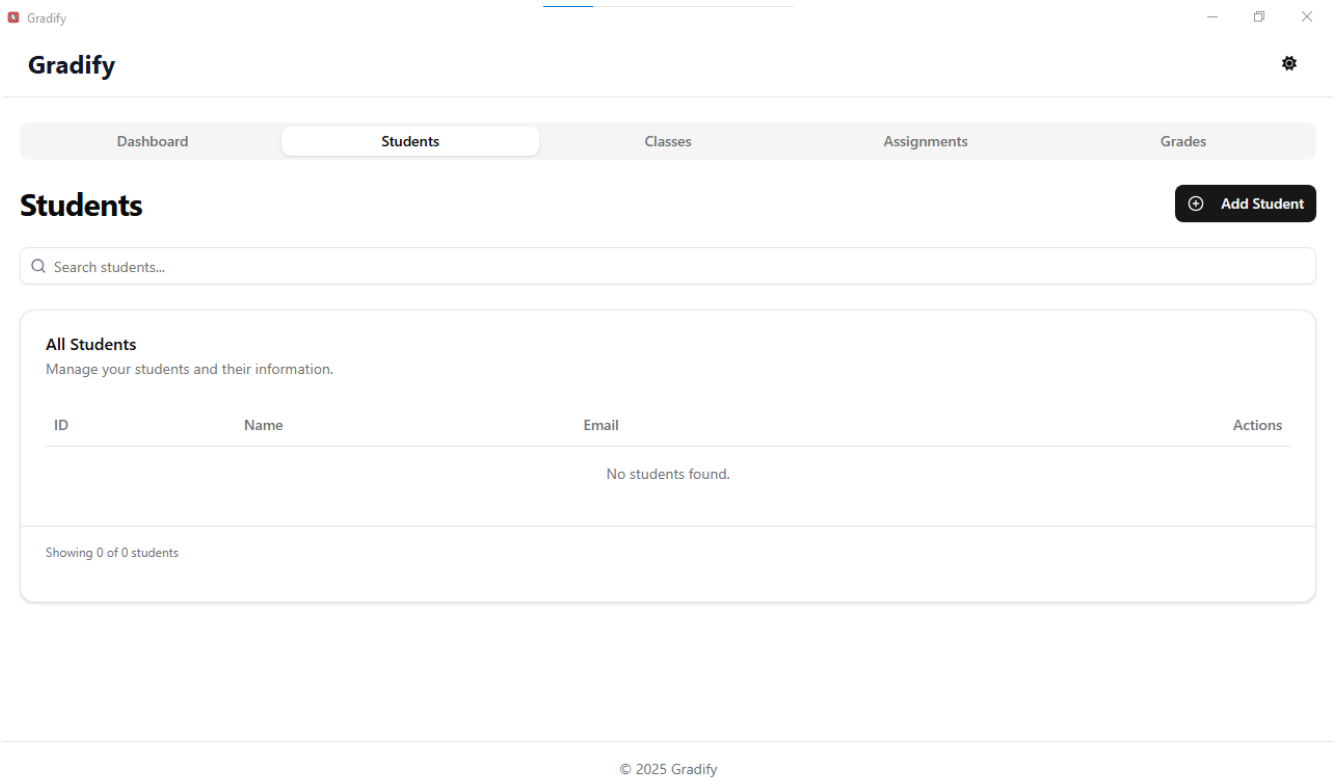
## 2.3 Using the Dashboard




The Dashboard shows:

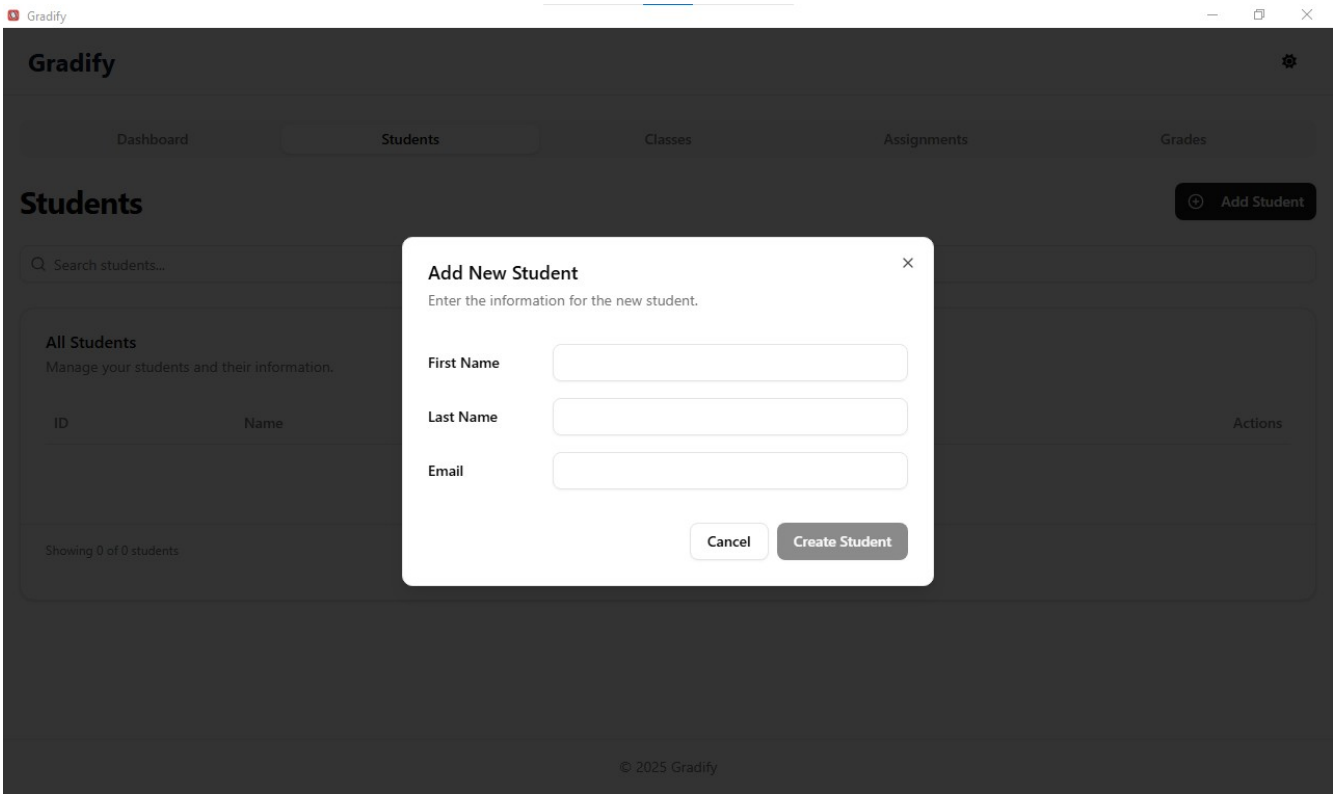
- Summary statistics
- At-risk students
- Upcoming assignments

## 2.4 Managing Students



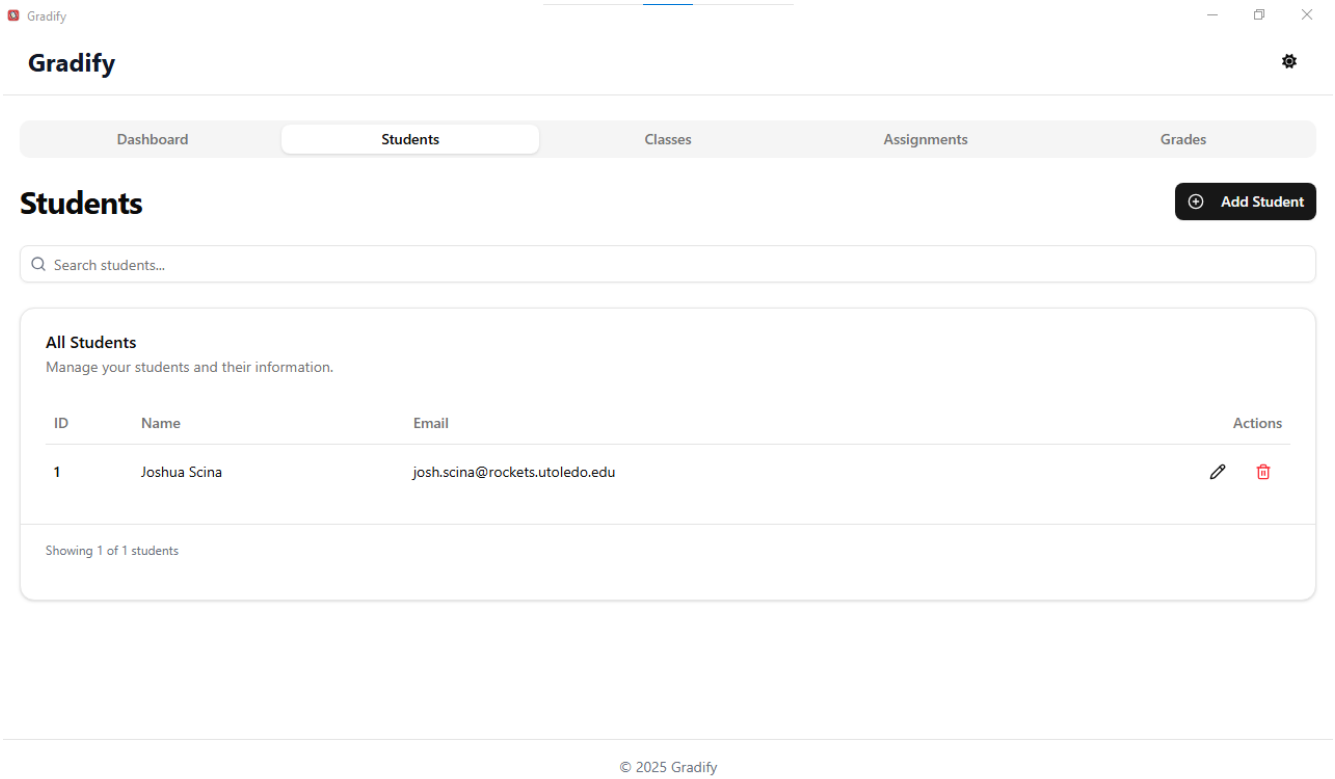

### Adding a Student

1. Click "Add Student"
2. Enter first name, last name, and optional email
3. Click "Create Student"

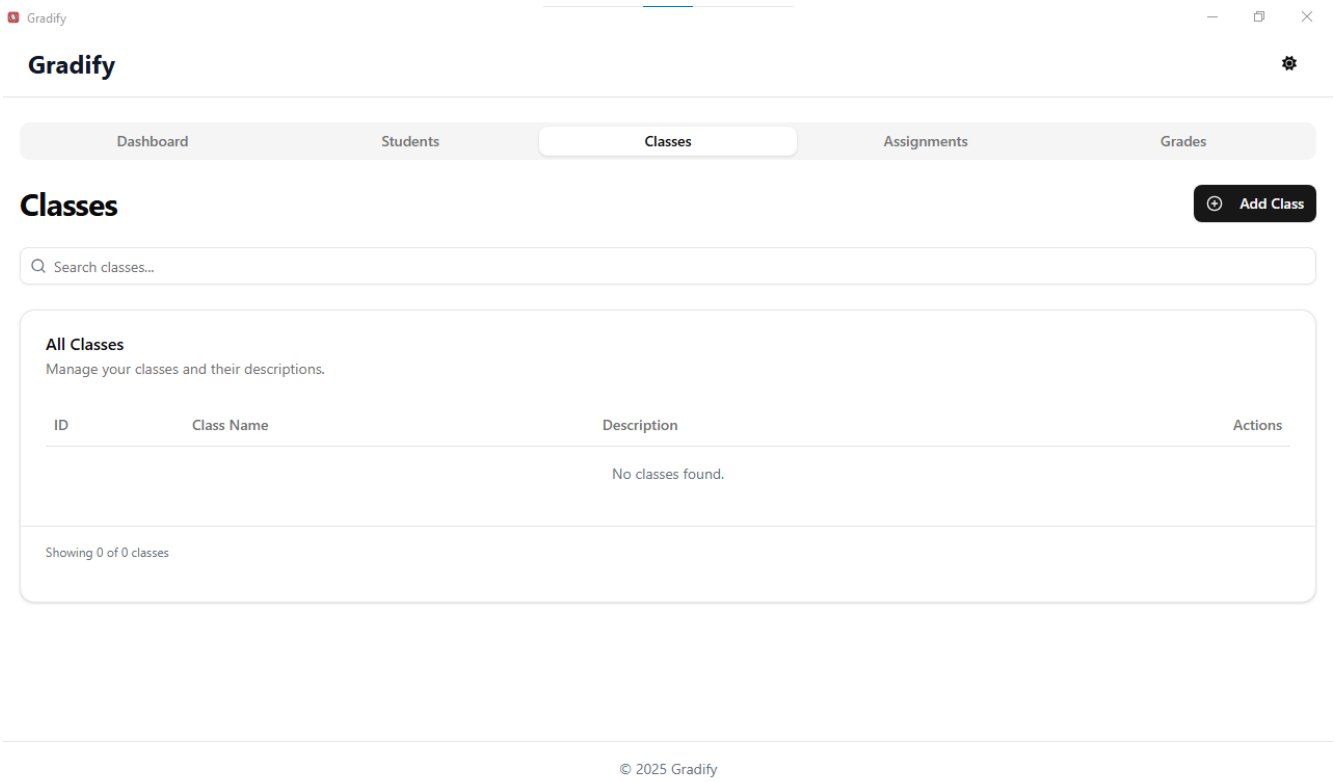



Editing/Deleting Students

- Click pencil icon to edit
- Click trash icon to delete

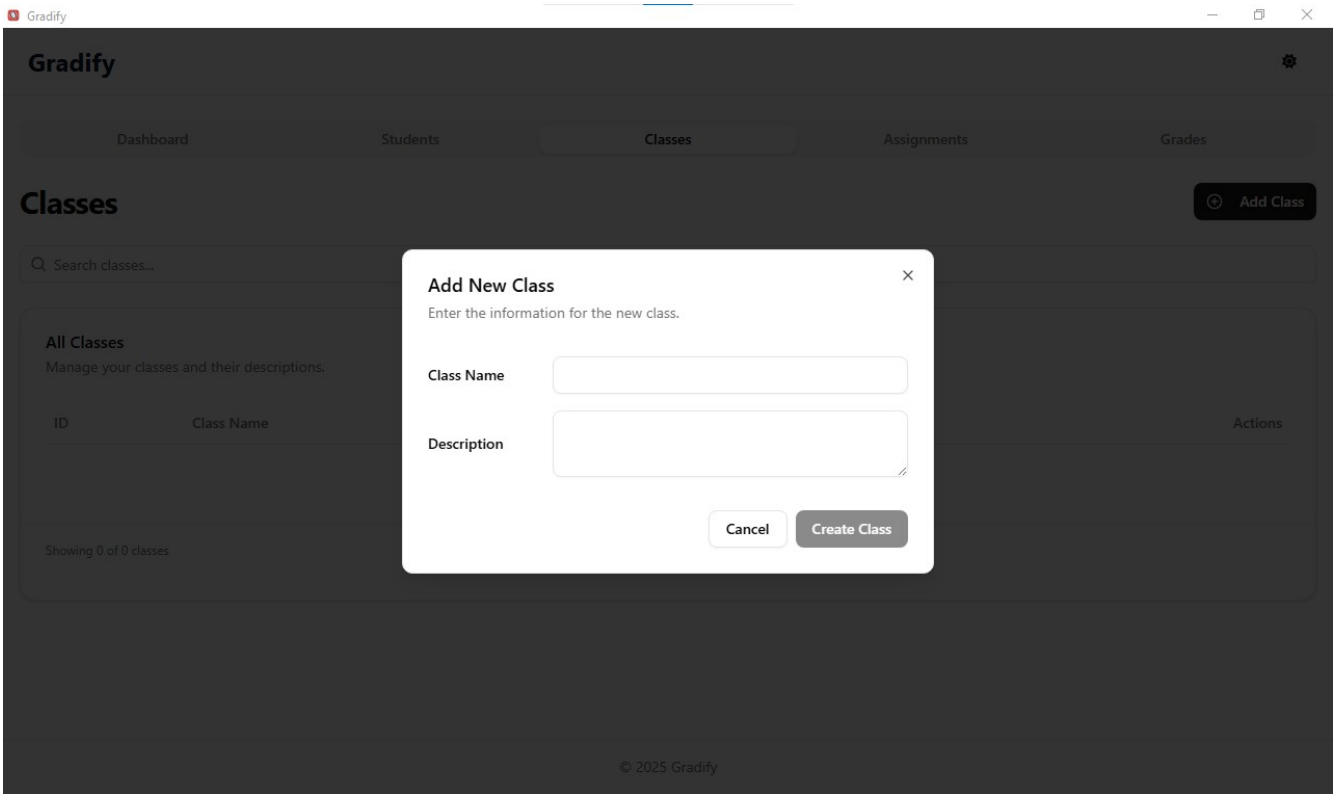


## 2.5 Managing Classes



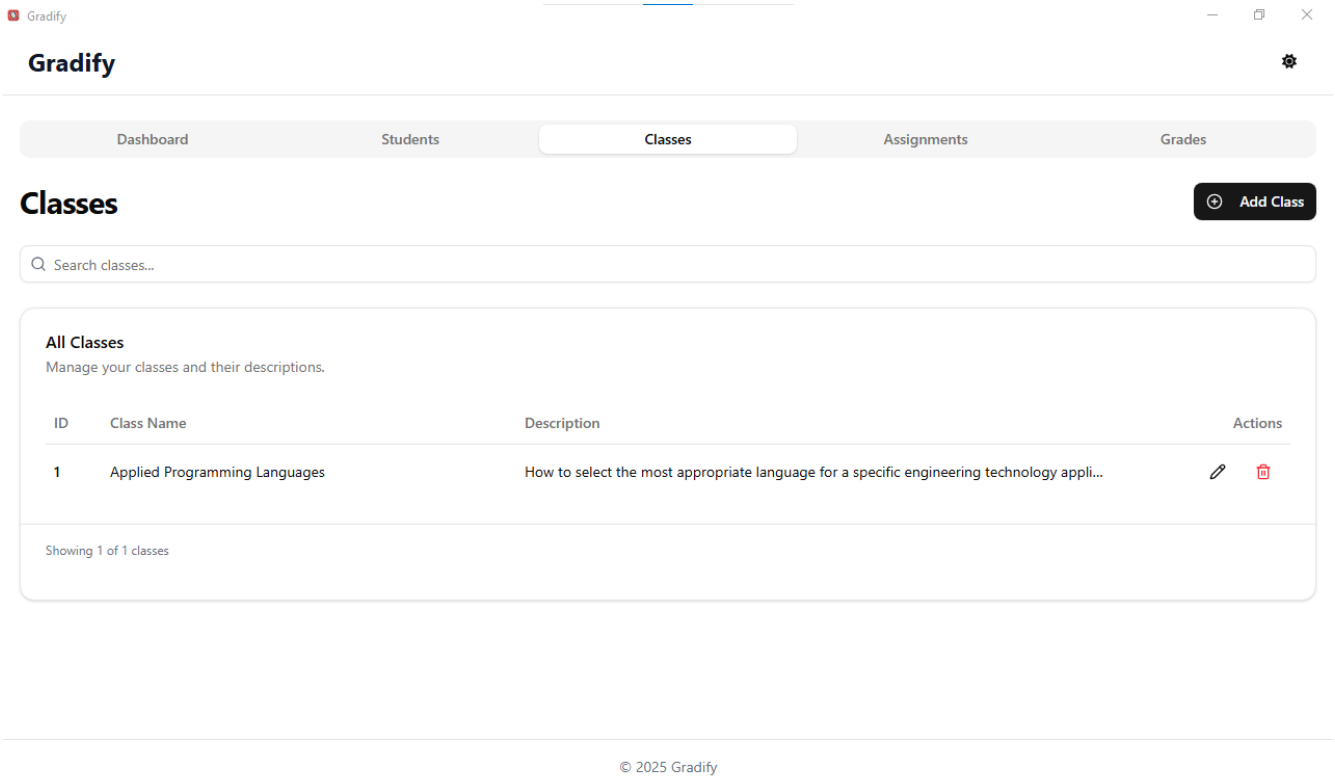

### Adding a Class

1. Click "Add Class"
2. Enter class name and optional description
3. Click "Create Class"

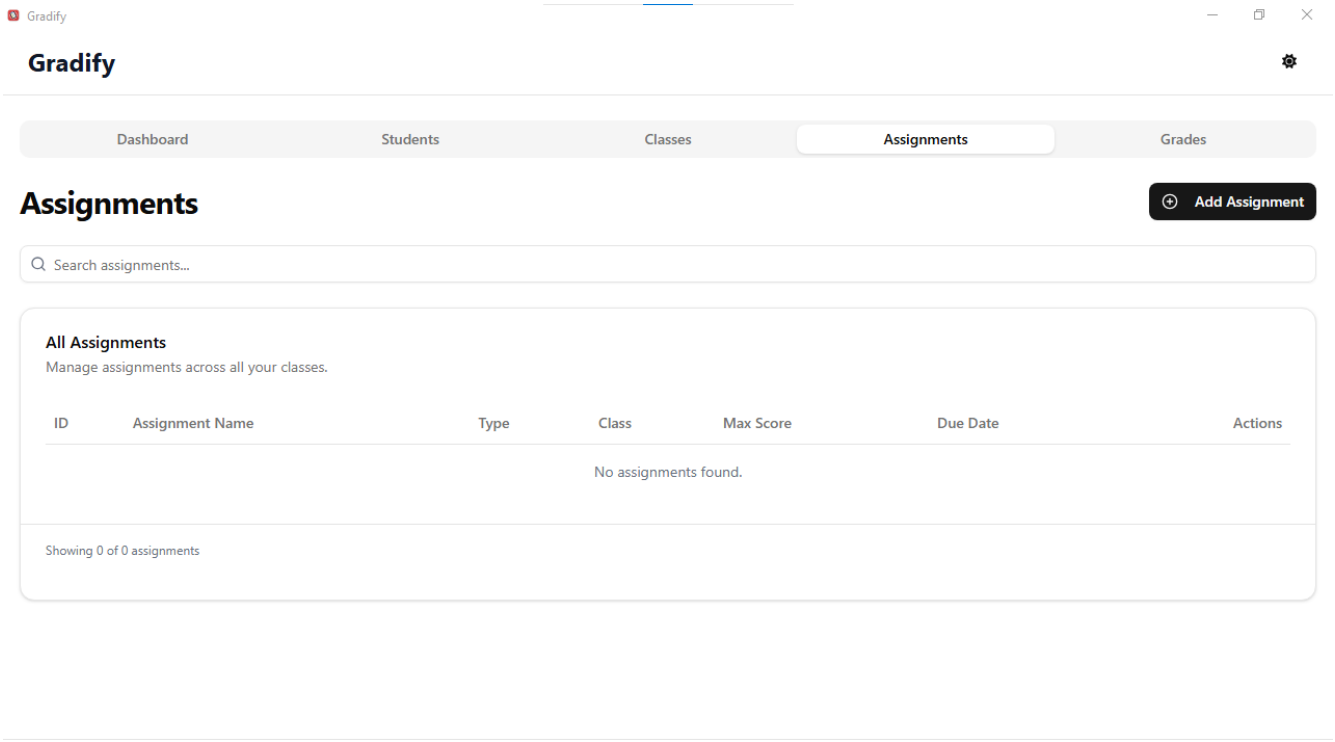



Editing/Deleting Classes

- Click pencil icon to edit
- Click trash icon to delete



## 2.6 Managing Assignments

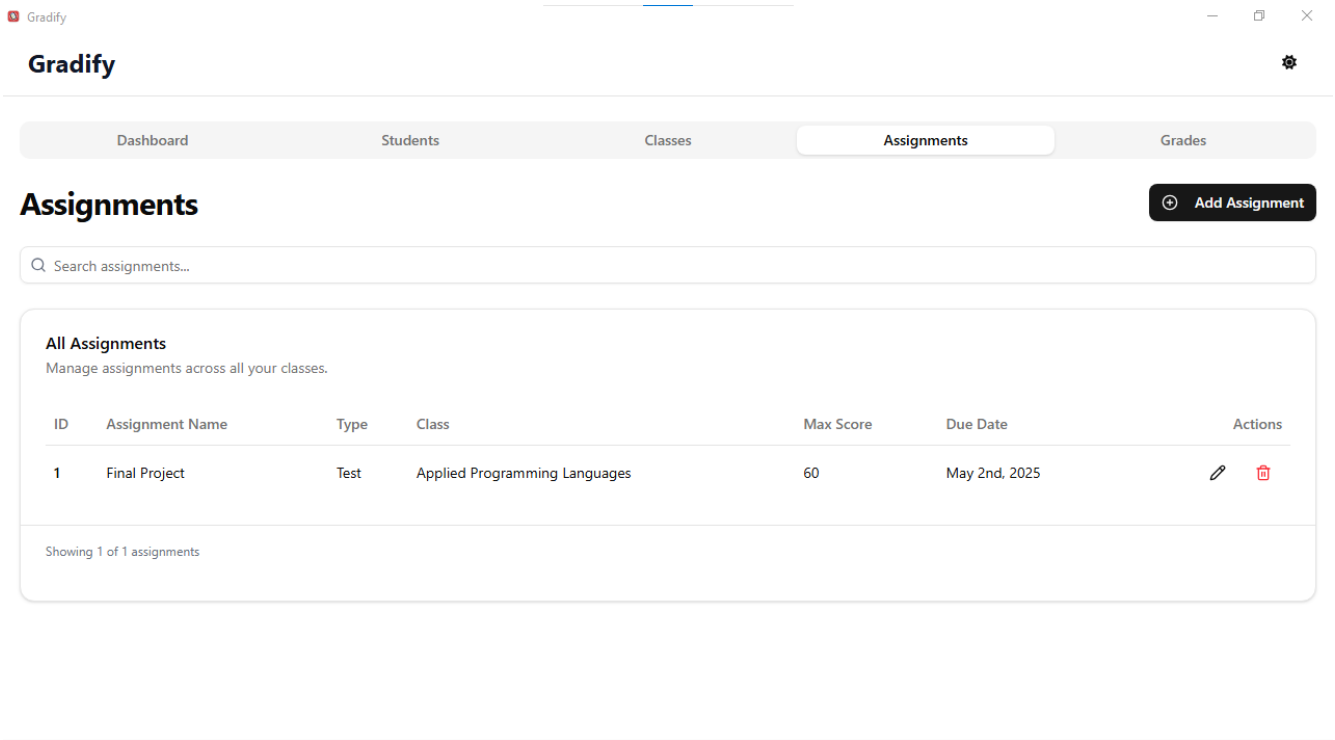



### Adding an Assignment

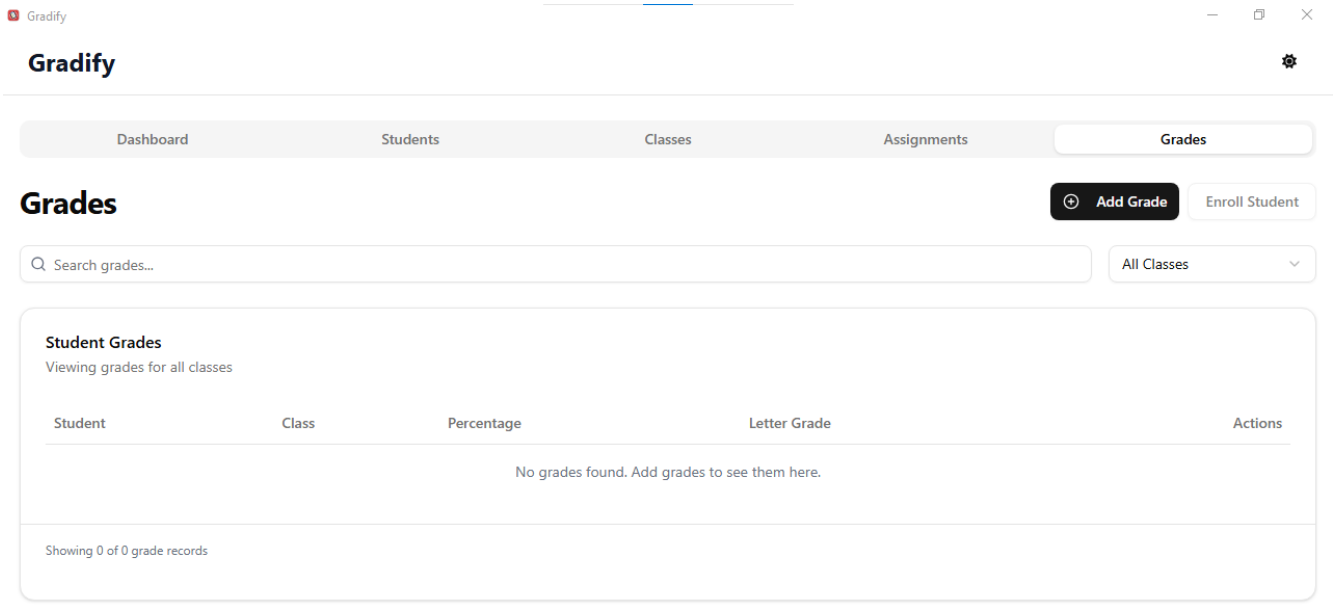
1. Click "Add Assignment"
2. Enter name, select class and type (Homework/Test)
3. Set maximum score and optional due date
4. Click "Create Assignment"

Editing/Deleting Assignments

- Click pencil icon to edit
- Click trash icon to delete



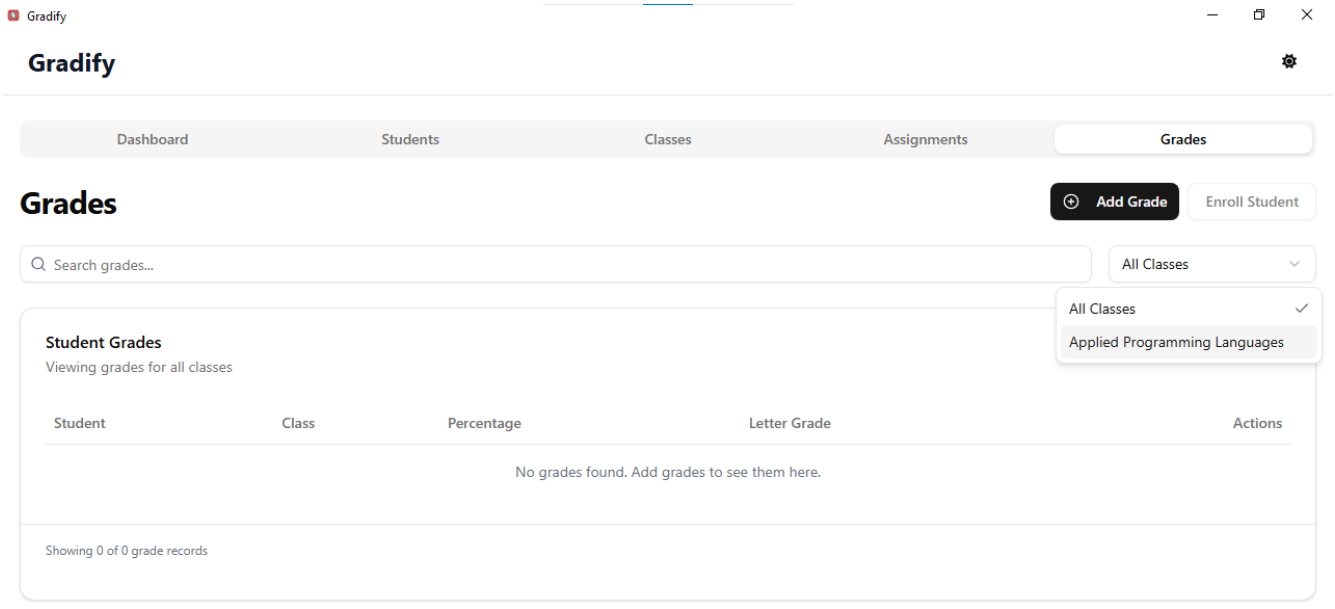

## 2.7 Managing Grades



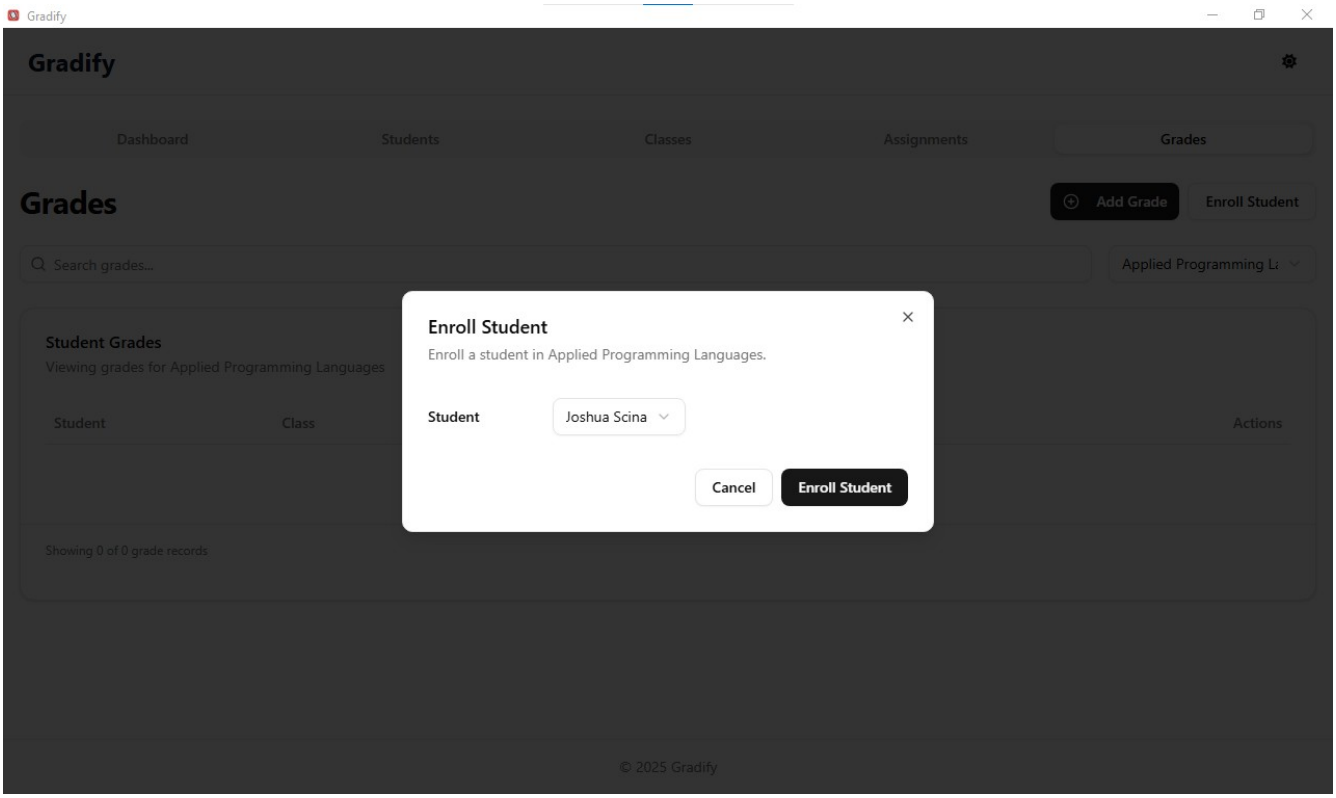



## Student Enrollment

- Select a Class

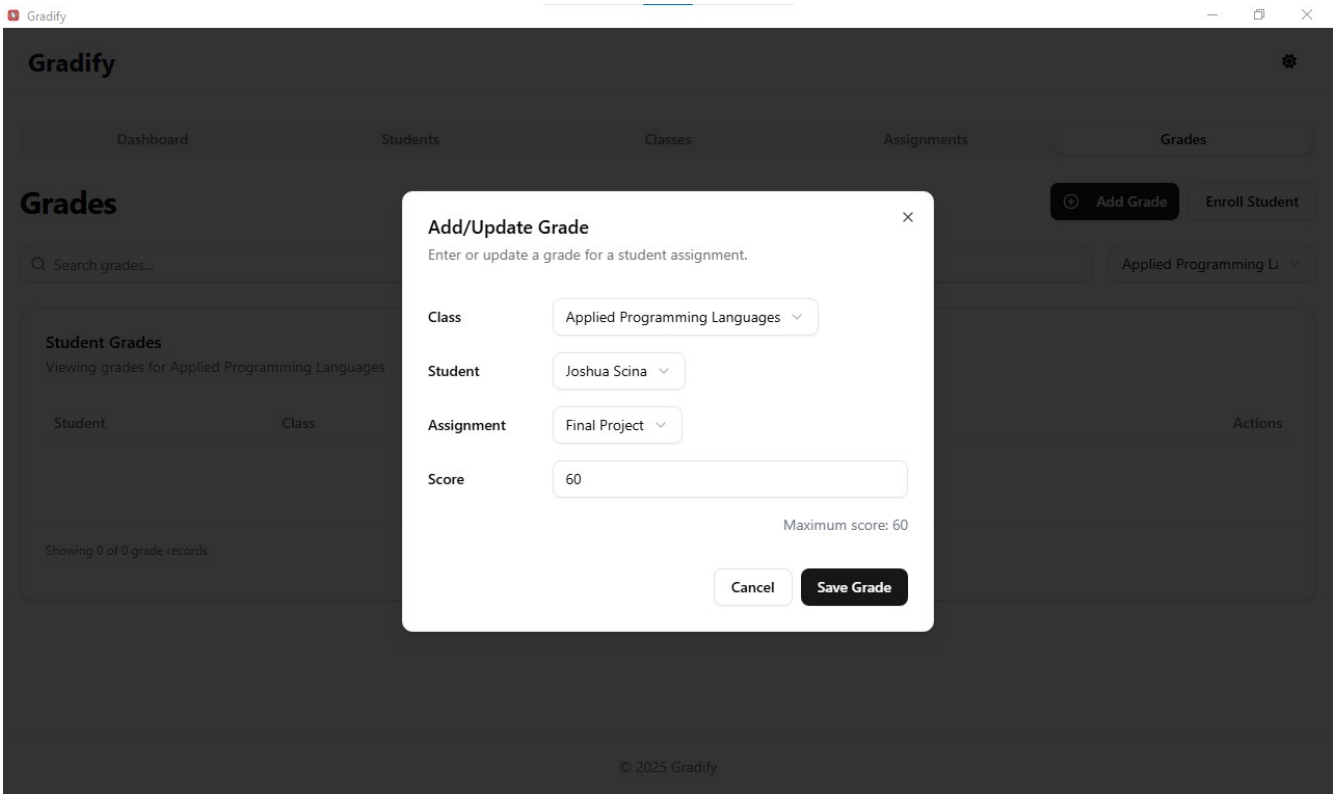


- Use "Enroll Student" to add students to classes

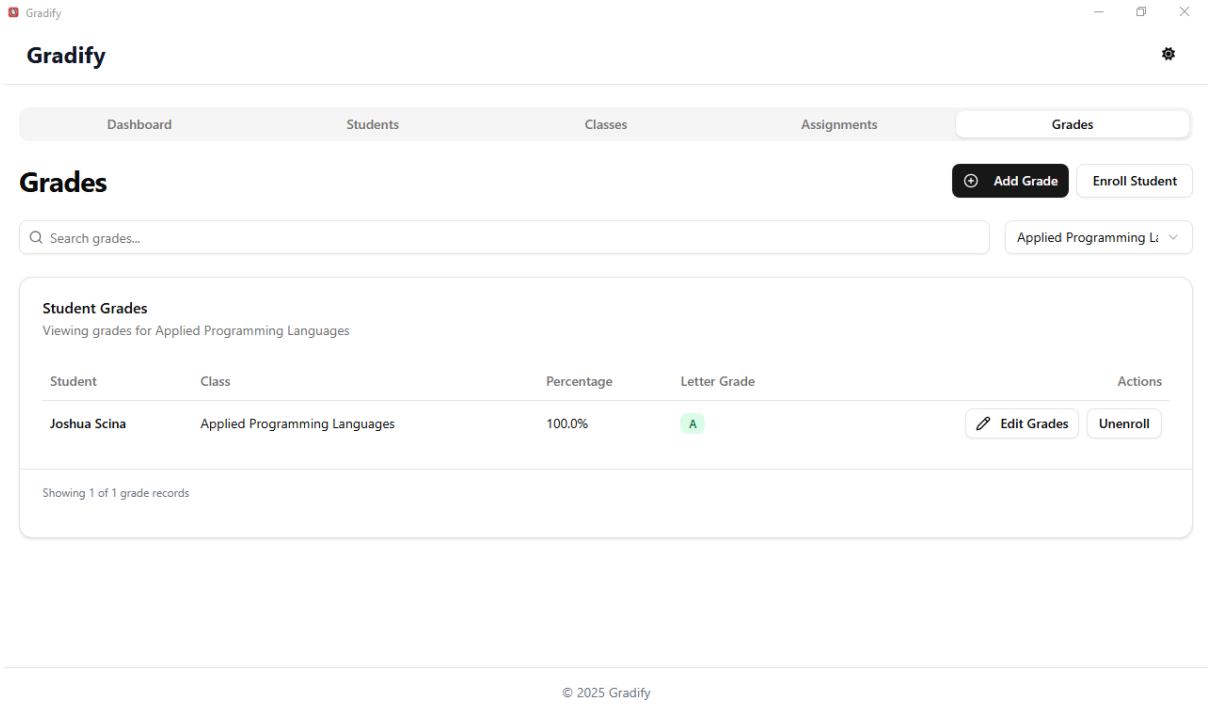



Adding/Editing Grades

1. Click "Add Grade" or "Edit Grades"
2. Select class, student, assignment (for adding)

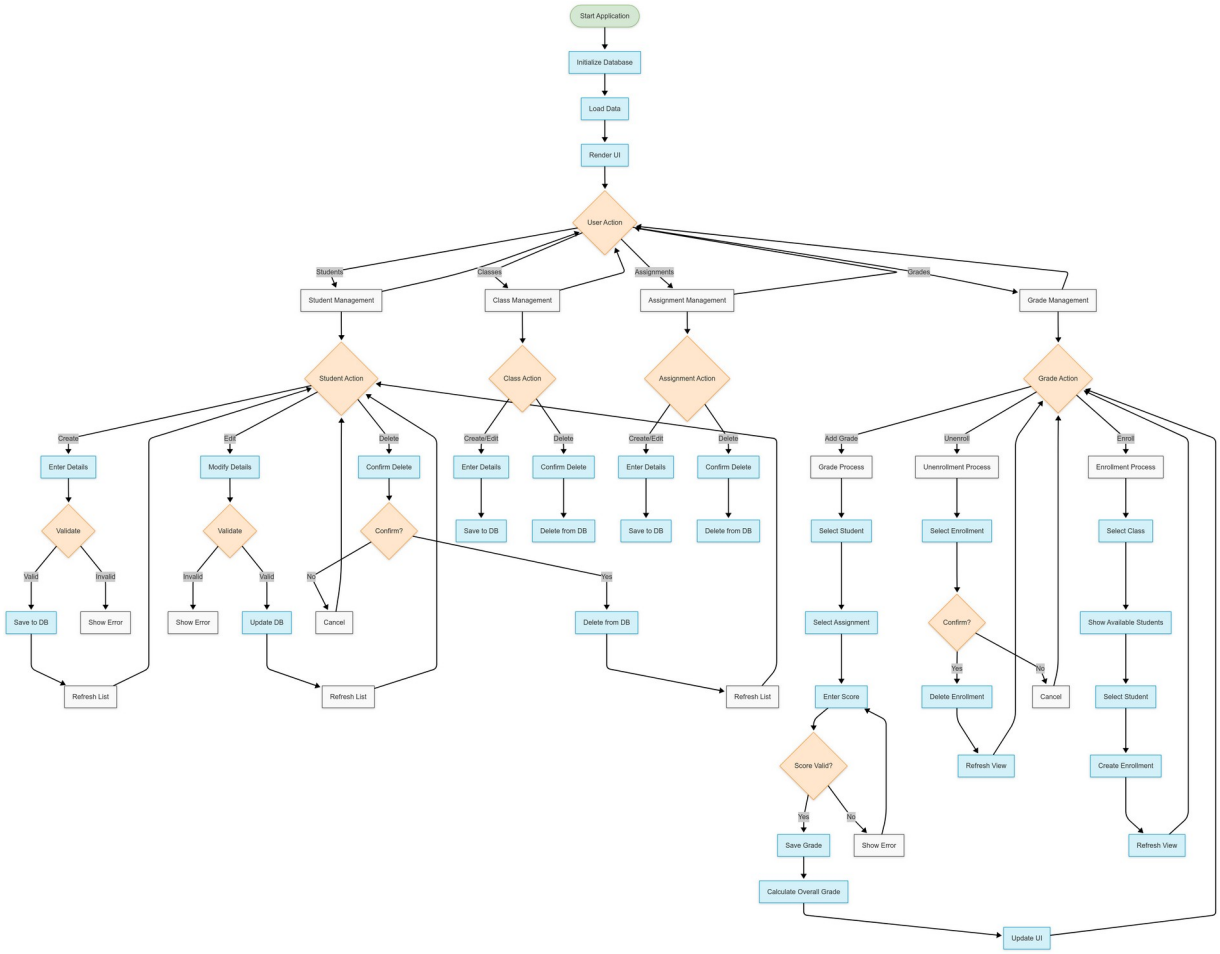



3. Enter score(s)
4. Click "Save Grade" or "Update Grades"




5. Student Grades can then be edited or unenrolled from the class selected.

## 2.8 Application Data Flow




## 2.9 Error Handling

Gradify validates all inputs to ensure data integrity:

- Form validation prevents invalid data entry
- Grade constraints ensure scores don't exceed maximums
- Database constraints maintain referential integrity

# 3. Technical Background

## 3.1 History and Overview of React/TypeScript (Frontend)

React, developed by Facebook and released in 2013, is a JavaScript library for building user interfaces. Its key innovations include the virtual DOM and component-based architecture, enabling efficient UI updates and code reusability.

TypeScript, created by Microsoft in 2012, extends JavaScript with static typing. It helps catch errors during development and provides better tooling support, making it ideal for large applications like Gradify.

In our application, React handles the interactive interface while TypeScript ensures type safety for student, class, assignment, and grade data structures.

### 3.2 History and Overview of Rust (Backend)

Rust, initially designed at Mozilla Research and officially released in 2015, is a systems programming language focused on memory safety without sacrificing performance. Its ownership system prevents common memory errors at compile time rather than runtime.

Key features include pattern matching, zero-cost abstractions, and guaranteed thread safety, making it excellent for reliable backend systems.

In Gradify, Rust powers all backend functionality, including database operations and business logic implementation.

### 3.3 Tauri Framework Overview

Tauri is a modern framework for building desktop applications with web technologies (frontend) and Rust (backend). Released in 2019, it offers smaller application sizes compared to alternatives like Electron by using the native OS WebView instead of bundling Chromium.

Tauri provides secure communication between the UI and system resources, with granular permissions and a reduced attack surface.

In our application, Tauri connects the React frontend with the Rust backend through message-passing commands.

### 3.4 SQLite Database

SQLite is a lightweight, embedded relational database created in 2000. Unlike client-server databases, it operates directly within the application and stores all data in a single file, making deployment simple.

Despite its small size, SQLite is ACID-compliant and requires no configuration, making it ideal for desktop applications.

Gradify uses SQLite with the SQLx Rust toolkit to store all application data with type-safe queries.

## 4. Language Comparison

### 4.1 Overview of Comparison Languages

This section compares our chosen technologies (React/TypeScript and Rust) with three alternative programming languages for developing Gradify:

Java: Object-oriented language commonly used for desktop applications
Python: High-level interpreted language known for simplicity
C#: Microsoft's general-purpose language integrated with .NET

## 4.2 Readability Comparison

React/TypeScript

Strengths: Type annotations provide self-documentation; JSX connects code to UI visually
Weaknesses: JSX can confuse newcomers; complex types can be verbose
Example: Component structure clearly shows UI organization and data flow

Rust

Strengths: Ownership makes data flow explicit; consistent syntax with pattern matching
Weaknesses: Steep learning curve; lifetime annotations can be complex
Example: Backend code shows strong typing with explicit error handling

Java

Strengths: Clear class structure; standard naming conventions; strong typing
Weaknesses: Verbose with boilerplate code; nested exception handling
Example: Equivalent code requires more ceremonial structure

Python

Strengths: Minimal syntax; focus on logic over declarations; clean appearance
Weaknesses: Dynamic typing can obscure variable purpose; indentation errors
Example: Simpler syntax but less self-documenting without type information

C#

Strengths: LINQ for data operations; concise property syntax; async/await pattern
Weaknesses: Attributes can obscure code flow; dependency on .NET ecosystem
Example: Modern features provide good balance of clarity and conciseness

Readability Ranking: Python > TypeScript/React > Rust > C# > Java

## 4.3 Writeability Comparison

React/TypeScript

Strengths: Rich component ecosystem; hot reloading; excellent IDE support
Weaknesses: Type definitions require initial investment; state management complexity
Development velocity: High

Rust

Strengths: Early error detection; pattern matching; excellent package management
Weaknesses: Strict compiler; ownership model learning curve; less mature ecosystem
Development velocity: Medium

Java

Strengths: Mature IDEs with refactoring tools; large ecosystem; established patterns
Weaknesses: Verbose syntax; boilerplate code; complex build processes

Development velocity: Medium

Python

Strengths: Concise syntax; dynamic typing; interactive development; extensive library
Weaknesses: Runtime errors; performance limitations; package management issues
Development velocity: Very High

C#

Strengths: Excellent Visual Studio integration; LINQ; Windows platform integration
Weaknesses: Platform bias; complex Microsoft ecosystem; deployment limitations
Development velocity: High

Writeability Ranking: Python > TypeScript/React > Rust > C# > Java

4.4 Reliability Comparison

React/TypeScript

Strengths: Static typing; immutable state patterns; component isolation
Weaknesses: Possible JavaScript runtime errors; state management complexity
Error prevention: High

Rust

Strengths: Memory safety without garbage collection; mandatory error handling; no null
Weaknesses: Learning curve; occasional need for unsafe code; maturing ecosystem
Error prevention: Very High

Java

Strengths: Strong type system; exception handling; mature tooling; stable JVM
Weaknesses: Null references; verbose error handling; maintenance challenges
Error prevention: High

Python

Strengths: Readable code reduces errors; flexible error handling; optional type hints
Weaknesses: Runtime type errors; limited concurrency with GIL; dynamic typing issues
Error prevention: Medium

C#

Strengths: Strong type system; structured exception handling; nullable reference types
Weaknesses: Legacy null handling; interop risks; async complexity
Error prevention: High

Reliability Ranking: Rust > C# > Java > TypeScript/React > Python

4.5 Cost Comparison (Performance)

React/TypeScript & Rust (Tauri)

Computational efficiency: Excellent - Rust provides near-native performance
Memory usage: Low - No garbage collection overhead in core logic
Startup time: Fast - Small binary size loads quickly
Application size: Small (~5-20MB) - Uses system WebView
Resource requirements: Minimal - Efficient compilation and execution

Java

Computational efficiency: Good - JIT compilation provides reasonable performance
Memory usage: High - JVM overhead and garbage collection
Startup time: Slow - JVM initialization overhead
Application size: Large (~100MB with runtime)
Resource requirements: Moderate - JVM needs substantial memory

Python

Computational efficiency: Poor - Interpreted execution with dynamic dispatch
Memory usage: Moderate - Reference counting with cycle detection
Startup time: Moderate - Interpreter initialization
Application size: Moderate - Requires Python runtime
Resource requirements: Varies - CPU-bound operations perform poorly

C#

Computational efficiency: Good - JIT compilation with good optimization
Memory usage: Moderate - Garbage collection more efficient than Java
Startup time: Moderate - Runtime initialization improved in recent versions
Application size: Moderate to Large (~50-100MB with runtime)
Resource requirements: Moderate - Similar to Java but more efficient

Performance Cost Ranking (from best to worst): Rust > C# > Java > Python

4.6 Comparison Conclusion

The combination of React/TypeScript for the frontend and Rust for the backend via Tauri provides an excellent balance for Gradify:

Readability: TypeScript's type system with React's component model creates maintainable code
Writeability: Component reuse and good tooling balance development speed with safety
Reliability: Strong typing and Rust's memory safety prevent many common errors
Performance: Rust backend provides excellent performance with minimal resource usage

While Python offers faster initial development and Java or C# have larger developer pools, our stack delivers the best combination of safety, performance, and maintainability for a desktop grading application.

### 4.7 NSIS (Nullsoft Scriptable Install System)

NSIS is a specialized scripting language for creating Windows installers, developed by Nullsoft in 2001.

Readability

Strengths: Clear section-based structure; straightforward command syntax
Weaknesses: Limited modern language features; becomes unwieldy for complex installations

Writeability

Strengths: Extensive documentation; powerful macros; large community
Weaknesses: Limited abstractions; verbose for complex operations
Development velocity: Medium

Reliability

Strengths: Well-tested across Windows versions; mature codebase
Weaknesses: Limited debugging tools; cryptic error messages
Error prevention: Medium

Cost (Performance)

Computational efficiency: Excellent - minimal resource usage during installation
Memory usage: Very low - small installation footprint
Output size: Small - creates compact installers
Resource requirements: Minimal - runs on virtually any Windows system

For Gradify, NSIS provides an ideal solution for creating small, efficient installers with proper Windows integration, desktop shortcuts, and silent installation support.

# 5. Language Features

## 5.1 Syntax Explanations

React/TypeScript Syntax

Component Declarations with TypeScript:

```
interface StudentsViewProps {
  students: Student[];
  refreshData: () => Promise<void>;
  loading: boolean;
}

export default function StudentsView({ students, refreshData, loading }: StudentsViewProps) {
```

```
  // Implementation
}
```

State Management:

```
const [searchQuery, setSearchQuery] = useState("");
const [currentStudent, setCurrentStudent] = useState<Student | null>(null);
```

JSX for UI:

```
return (
  <div className="space-y-6">
    <h2 className="text-3xl font-bold">Students</h2>
    <Button onClick={openCreateDialog}>Add Student</Button>
    {loading ? <Skeleton /> : <Table>{/* content */}</Table>}
  </div>
);
```

Rust Syntax

Function Declarations:

```
#[tauri::command(async, rename_all = "snake_case")]
pub async fn create_student(
    state: State<'_, Mutex<AppState>>,
    first_name: String,
    last_name: String,
    email: Option<String>,
) -> Result<Student, String> {
    // Implementation
}
```

Struct Definitions:

```
#[derive(Debug, Serialize, Deserialize, FromRow)]
pub struct Student {
    #[sqlx(rename = "ID")]
    pub id: i64,
    #[sqlx(rename = "FIRST_NAME")]
```

```
    pub first_name: String,
    #[sqlx(rename = "LAST_NAME")]
    pub last_name: String,
    #[sqlx(rename = "EMAIL")]
    pub email: Option<String>,
}
```

## 5.2 Control Structures

Conditional Logic

React/TypeScript:

```
{loading ? <Skeleton /> : <Table>{/* content */}</Table>}

{error && <Alert variant="destructive">{error}</Alert>}

if (!firstName || !lastName) return;
```

Rust:

```
if student_id == 0 {
    return Err("Invalid student ID".to_string());
}

match result {
    Ok(student) => Ok(student),
    Err(e) => Err(format!("Database error: {}", e)),
}
```

Iteration

React/TypeScript:

```
{students.map((student) => (
  <TableRow key={student.id}>
    <TableCell>{student.first_name} {student.last_name}</TableCell>
  </TableRow>
```

))}

```
const filteredStudents = students.filter(student =>
  student.first_name.toLowerCase().includes(searchQuery)
);
```

Rust:

```
for grade in grades.iter() {
    total_score += grade.score;
}

let passing_grades = grades
    .iter()
    .filter(|g| g.score >= 60.0)
    .collect::<Vec<_>>();
```

5.3 Data Structures

React/TypeScript:

```
interface Student {
  id: number;
  first_name: string;
  last_name: string;
  email?: string;
}

const [students, setStudents] = useState<Student[]>([]);

type GradeStatus = 'passing' | 'failing' | 'incomplete';
```

Rust:

```
pub struct Grade {
    pub student_id: i64,
    pub assignment_id: i64,
```

```
    pub score: f64,
}

enum GradeResult {
    Pass(f64),
    Fail(f64),
    Incomplete,
}

let students: Vec<Student> = sqlx::query_as("SELECT * FROM STUDENTS")
    .fetch_all(&pool)
    .await?;
```

## 5.4 Input/Output Handling

React/TypeScript:

```
<Input
  value={firstName}
  onChange={(e) => setFirstName(e.target.value)}
/>

export async function createStudent(
  first_name: string,
  last_name: string,
  email?: string,
): Promise<Student> {
  return await invoke<Student>("create_student", {
    first_name,
    last_name,
    email,
  });
```

```
}
```

Rust:

```
let student = sqlx::query_as::<_, Student>(
  "SELECT ID, FIRST_NAME, LAST_NAME, EMAIL FROM STUDENTS WHERE ID = ?",
)
.bind(id)
.fetch_one(&state.db.pool)
.await
.map_err(|e| e.to_string())?;
```

## 5.5 Parameter Passing Methods

React/TypeScript:

```
// Props
function StudentsView({ students, refreshData }: StudentsViewProps) {
  // Implementation
}

// Callbacks
<StudentForm onSubmit={handleStudentCreated} />
```

Rust:

```
// Ownership transfer
fn process_student(student: Student) {
  // Function takes ownership
}

// Borrowing with references
fn display_student(student: &Student) {
  // Function borrows immutably
}
```

## 5.6 Scope Rules

React/TypeScript:

```
function StudentsList() {
  // Function scope
  const [students, setStudents] = useState<Student[]>([]);

  {
    // Block scope
    const temporaryVar = "only visible here";
  }

  // Closure capturing variables
  const deleteStudent = useCallback((id: number) => {
    setStudents(students.filter(student => student.id !== id));
  }, [students]);
}
```

Rust:

```
fn process_grades() {
    let total = 0.0;

    {
        // Block scope
        let grades = get_grades();
        for grade in grades {
            total += grade.score;
        }
    } // grades dropped here

    // Module visibility
```

```
    pub fn public_function() {}
    fn private_function() {}
}
```

## 5.7 Memory Management

React/TypeScript:

```
// Component lifecycle
useEffect(() => {
  let mounted = true;

  async function loadData() {
    const data = await fetchData();
    if (mounted) {
      setData(data);
    }
  }

  loadData();

  return () => {
    mounted = false; // Cleanup on unmount
  };
}, []);
```

Rust:

```
// Ownership
let student = Student { id: 1, first_name: "Jane".to_string() };
process_student(student); // student moved here
// student no longer accessible

// Borrowing
```

```
fn display_student(student: &Student) {
    // Can read but not modify
}

fn update_student(student: &mut Student) {
    // Can modify
    student.first_name = "Updated".to_string();
}
```

## 5.8 Error Handling

React/TypeScript:

```
// Try/catch
try {
  await createStudent(firstName, lastName, email);
  setIsCreateDialogOpen(false);
} catch (error) {
  setError("Failed to create student");
}

// Form validation
const validateEmail = (value: string) => {
  if (!value) return true;

  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  if (!emailRegex.test(value)) {
    setEmailError("Invalid email address");
    return false;
  }
  return true;
};
```

Rust:

```
// Result type
pub async fn get_student(id: i64) -> Result<Student, String> {
  // Implementation returning Ok or Err
}

// ? operator for propagation
let result = query.execute(&pool).await.map_err(|e| e.to_string())?;

// Pattern matching
match db_operation() {
  Ok(data) => process_data(data),
  Err(sqlx::Error::RowNotFound) => handle_missing_data(),
  Err(e) => handle_general_error(e)
}
```

### 5.9 Language Features Summary

The Gradify application leverages the strengths of both languages:

- React/TypeScript: Component-based UI, strong typing, and declarative rendering
- Rust: Memory safety, ownership model, and high performance for data operations

Together, this architecture ensures reliable grading with proper error handling, data validation, and responsive performance even with large datasets.

## Appendix: Source Code

Frontend (React/TypeScript)

Core Application

- src/App.tsx: Main application component

```
import { useState, useEffect } from "react";
import { Tabs, TabsContent, TabsList, TabsTrigger } from "@/components/ui/tabs";
import { Button } from "@/components/ui/button";
import { Alert, AlertTitle, AlertDescription } from "@/components/ui/alert";
import { AlertCircle } from "lucide-react";

import { getAllStudents } from "@/api/students";
```

```
import { getAllClasses } from "@/api/classes";
import { getAllAssignments } from "@/api/assignments";
import { getOverallGrades } from "@/api/overall-grades";

import StudentsView from "@/components/students-view";
import ClassesView from "@/components/classes-view";
import AssignmentsView from "@/components/assignments-view";
import GradesView from "@/components/grades-view";
import DashboardView from "@/components/dashboard-view";

import type { Student, Class, Assignment, OverallGrade } from "@/api/types";
import "./app.css";
import { faSun, faMoon } from "@fortawesome/free-solid-svg-icons";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";

function App() {
  const [students, setStudents] = useState<Student[]>([]);
  const [classes, setClasses] = useState<Class[]>([]);
  const [assignments, setAssignments] = useState<Assignment[]>([]);
  const [overallGrades, setOverallGrades] = useState<OverallGrade[]>([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);
  const [activeTab, setActiveTab] = useState("dashboard");
  const [isDarkMode, setIsDarkMode] = useState(
    document.body.classList.contains("dark"),
  );

  useEffect(() => {
    const handleThemeChange = () => {
      setIsDarkMode(document.body.classList.contains("dark"));
    };

    document.body.addEventListener("classChange", handleThemeChange);

    return () => {
      document.body.removeEventListener("classChange", handleThemeChange);
    };
  }, []);

  useEffect(() => {
    loadData();
  }, []);
```

```
const loadData = async () => {
  try {
    setLoading(true);
    setError(null);

    try {
      const studentsData = await getAllStudents();
      setStudents(studentsData);
    } catch (err) {
      console.error("Error loading students:", err);
      setError(
        "Failed to load students data. Please check database connection.",
      );
      return;
    }

    try {
      const classesData = await getAllClasses();
      setClasses(classesData);
    } catch (err) {
      console.error("Error loading classes:", err);
      setError("Failed to load classes data. Please check database schema.");
      return;
    }

    try {
      const assignmentsData = await getAllAssignments();
      setAssignments(assignmentsData);
    } catch (err) {
      console.error("Error loading assignments:", err);
      setError(
        "Failed to load assignments data. Please check database schema.",
      );
      return;
    }

    try {
      const overallGradesData = await getOverallGrades();
      setOverallGrades(overallGradesData);
    } catch (err) {
      console.error("Error loading overall grades:", err);
```

```
      setError("Failed to load grades data. Please check database schema.");
      return;
    }
  } catch (error) {
    console.error("Error loading data:", error);
    setError("An unexpected error occurred while loading data.");
  } finally {
    setLoading(false);
  }
};

const refreshData = async () => {
  await loadData();
};

return (
  <div className="flex flex-col min-h-screen">
    <header className="sticky top-0 z-10 border-b bg-white dark:bg-gray-950 px-6 py-4">
      <div className="flex items-center justify-between">
        <h1 className="text-2xl font-bold text-gray-900 dark:text-white">
          Gradify
        </h1>
        <div className="flex items-center gap-4">
          <Button
            variant="ghost"
            size="sm"
            onClick={() => {
              document.body.classList.toggle("dark");
              setIsDarkMode(!isDarkMode);
            }}
          >
            {isDarkMode ? (
              <FontAwesomeIcon icon={faMoon} />
            ) : (
              <FontAwesomeIcon icon={faSun} />
            )}
          </Button>
        </div>
      </div>
    </header>

    <main className="container mx-auto py-6 px-4 flex-1">
```

```
{error && (
  <Alert variant="destructive" className="mb-6">
    <AlertCircle className="h-4 w-4" />
    <AlertTitle>Error</AlertTitle>
    <AlertDescription>{error}</AlertDescription>
    <Button
      variant="outline"
      size="lg"
      onClick={loadData}
      className="mt-2"
    >
      Retry
    </Button>
  </Alert>
)}

<Tabs
  value={activeTab}
  onValueChange={setActiveTab}
  className="space-y-4"
>
  <TabsList className="grid w-full grid-cols-5">
    <TabsTrigger value="dashboard">Dashboard</TabsTrigger>
    <TabsTrigger value="students">Students</TabsTrigger>
    <TabsTrigger value="classes">Classes</TabsTrigger>
    <TabsTrigger value="assignments">Assignments</TabsTrigger>
    <TabsTrigger value="grades">Grades</TabsTrigger>
  </TabsList>

  <TabsContent value="dashboard" className="space-y-4">
    <DashboardView
      students={students}
      classes={classes}
      assignments={assignments}
      overallGrades={overallGrades}
      loading={loading}
    />
  </TabsContent>

  <TabsContent value="students" className="space-y-4">
    <StudentsView
      students={students}
```

```
    refreshData={refreshData}
    loading={loading}
   />
</TabsContent>

<TabsContent value="classes" className="space-y-4">
  <ClassesView
    classes={classes}
    refreshData={refreshData}
    loading={loading}
  />
</TabsContent>

<TabsContent value="assignments" className="space-y-4">
  <AssignmentsView
    assignments={assignments}
    classes={classes}
    refreshData={refreshData}
    loading={loading}
  />
</TabsContent>

<TabsContent value="grades" className="space-y-4">
  {classes.length > 0 && students.length > 0 ? (
    <GradesView
      grades={overallGrades}
      students={students}
      classes={classes}
      assignments={assignments}
      refreshData={refreshData}
      loading={loading}
    />
  ) : (
    <Alert className="mb-6">
      <AlertCircle className="h-4 w-4" />
      <AlertTitle>No data available</AlertTitle>
      <AlertDescription>
        Please create at least one class and one student before
        accessing grades.
      </AlertDescription>
    </Alert>
  )}
```

```
        </TabsContent>
      </Tabs>
    </main>

    <footer className="border-t bg-white dark:bg-gray-950 px-6 py-4 text-center text-sm text-gray-500 mt-auto">
      <p>© 2025 Gradify</p>
    </footer>
  </div>
 );
}

export default App;
```

src/api/types.ts: Core data type definitions

```
export interface Student {
 id: number;
 first_name: string;
 last_name: string;
 email?: string;
}

export interface Grade {
 student_id: number;
 assignment_id: number;
 score: number;
}

export interface Class {
 id: number;
 class_name: string;
 description?: string;
}

export interface Assignment {
 id: number;
 class_id: number;
 assignment_name: string;
 assignment_type: string;
 maximum_score: number;
 // Represent due_date as a string in ISO8601 format, or undefined if not set.
 due_date?: string;
}
```

```
export interface StudentClass {
  student_id: number;
  class_id: number;
}

export interface OverallGrade {
  student_id: number;
  class_id: number;
  percentage: number;
  letter_grade: string;
}
```

src/api/students.ts: Student API interface

```
import { invoke } from "@tauri-apps/api/core";
import type { Student } from "./types";

export async function createStudent(
  first_name: string,
  last_name: string,
  email?: string,
): Promise<Student> {
  return await invoke<Student>("create_student", {
    first_name,
    last_name,
    email,
  });
}

export async function getStudent(id: number): Promise<Student> {
  return await invoke<Student>("get_student", { id });
}

export async function getAllStudents(): Promise<Student[]> {
  return await invoke<Student[]>("get_all_students");
}

export async function updateStudent(
  id: number,
  first_name: string,
  last_name: string,
  email?: string,
): Promise<Student> {
```

```
  return await invoke<Student>("update_student", {
    id,
    first_name,
    last_name,
    email,
  });
}

export async function deleteStudent(id: number): Promise<void> {
  return await invoke("delete_student", { id });
}
```

src/api/classes.ts: Class API interface

```
import { invoke } from "@tauri-apps/api/core";
import type { Class } from "./types";

export async function createClass(
  class_name: string,
  description?: string,
): Promise<Class> {
  return await invoke<Class>("create_class", { class_name, description });
}

export async function getClass(id: number): Promise<Class> {
  return await invoke<Class>("get_class", { id });
}

export async function getAllClasses(): Promise<Class[]> {
  return await invoke<Class[]>("get_all_classes");
}

export async function updateClass(
  id: number,
  class_name: string,
  description?: string,
): Promise<Class> {
  return await invoke<Class>("update_class", { id, class_name, description });
}

export async function deleteClass(id: number): Promise<void> {
  return await invoke("delete_class", { id });
}
```

src/api/assignments.ts: Assignment API interface

```
import { invoke } from "@tauri-apps/api/core";
import type { Assignment } from "./types";

export async function createAssignment(
  class_id: number,
  assignment_name: string,
  assignment_type: string,
  maximum_score: number,
  due_date?: string,
): Promise<Assignment> {
  return await invoke<Assignment>("create_assignment", {
    class_id,
    assignment_name,
    assignment_type,
    maximum_score,
    due_date,
  });
}

export async function getAssignment(id: number): Promise<Assignment> {
  return await invoke<Assignment>("get_assignment", { id });
}

export async function getAllAssignments(): Promise<Assignment[]> {
  return await invoke<Assignment[]>("get_all_assignments");
}

export async function updateAssignment(
  id: number,
  class_id: number,
  assignment_name: string,
  assignment_type: string,
  maximum_score: number,
  due_date?: string,
): Promise<Assignment> {
  return await invoke<Assignment>("update_assignment", {
    id,
    class_id,
    assignment_name,
    assignment_type,
    maximum_score,
```

```
    due_date,
  });
}

export async function deleteAssignment(id: number): Promise<void> {
  return await invoke("delete_assignment", { id });
          src/api/grades.ts: Grade API interface

import { invoke } from "@tauri-apps/api/core";
import type { Grade } from "./types";


export async function createGrade(
  student_id: number,
  assignment_id: number,
  score: number,
): Promise<Grade> {
  return await invoke<Grade>("create_grade", {
    student_id,
    assignment_id,
    score,
  });
}


export async function getGrade(
  student_id: number,
  assignment_id: number,
): Promise<Grade> {
  return await invoke<Grade>("get_grade", { student_id, assignment_id });
}


export async function getAllGrades(): Promise<Grade[]> {
  return await invoke<Grade[]>("get_all_grades");
```

```
}

export async function updateGrade(
  student_id: number,
  assignment_id: number,
  score: number,
): Promise<Grade> {
  return await invoke<Grade>("update_grade", {
    student_id,
    assignment_id,
    score,
  });
}

export async function deleteGrade(
  student_id: number,
  assignment_id: number,
): Promise<void> {
  return await invoke("delete_grade", { student_id, assignment_id });
}
```

UI Components

src/components/students-view.tsx: Student management UI

```
import { useState } from "react";
import {
  Card,
  CardContent,
  CardDescription,
  CardFooter,
  CardHeader,
  CardTitle,
} from "@/components/ui/card";
import { Button } from "@/components/ui/button";
import {
```

```
  Dialog,
  DialogContent,
  DialogDescription,
  DialogFooter,
  DialogHeader,
  DialogTitle,
} from "@/components/ui/dialog";
import { Input } from "@/components/ui/input";
import { Label } from "@/components/ui/label";
import {
  Table,
  TableBody,
  TableCell,
  TableHead,
  TableHeader,
  TableRow,
} from "@/components/ui/table";
import { Skeleton } from "@/components/ui/skeleton";
import { PlusCircle, Pencil, Trash2, Search } from "lucide-react";
import { createStudent, updateStudent, deleteStudent } from "@/api/students";
import type { Student } from "@/api/types";

interface StudentsViewProps {
  students: Student[];
  refreshData: () => Promise<void>;
  loading: boolean;
}

export default function StudentsView({
  students,
  refreshData,
  loading,
}: StudentsViewProps) {
  const [searchQuery, setSearchQuery] = useState("");
  const [isCreateDialogOpen, setIsCreateDialogOpen] = useState(false);
  const [isEditDialogOpen, setIsEditDialogOpen] = useState(false);
  const [isDeleteDialogOpen, setIsDeleteDialogOpen] = useState(false);
  const [currentStudent, setCurrentStudent] = useState<Student | null>(null);

  const [firstName, setFirstName] = useState("");
  const [lastName, setLastName] = useState("");
  const [email, setEmail] = useState("");
```

```
const [emailError, setEmailError] = useState("");
const [firstNameError, setFirstNameError] = useState("");
const [lastNameError, setLastNameError] = useState("");

const filteredStudents = students.filter((student) => {
  const fullName = `${student.first_name} ${student.last_name}`.toLowerCase();
  const emailLower = student.email?.toLowerCase() || "";
  const query = searchQuery.toLowerCase();

  return fullName.includes(query) || emailLower.includes(query);
});

const validateFirstName = (value: string) => {
  if (!value.trim()) {
    setFirstNameError("First name is required");
    return false;
  }
  if (value.length > 50) {
    setFirstNameError("First name cannot exceed 50 characters");
    return false;
  }
  setFirstNameError("");
  return true;
};

const validateLastName = (value: string) => {
  if (!value.trim()) {
    setLastNameError("Last name is required");
    return false;
  }
  if (value.length > 50) {
    setLastNameError("Last name cannot exceed 50 characters");
    return false;
  }
  setLastNameError("");
  return true;
};

const validateEmail = (emailValue: string, studentId?: number) => {
  if (!emailValue) {
    setEmailError("");
    return true;
```

```
  }

  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  if (!emailRegex.test(emailValue)) {
    setEmailError("Please enter a valid email address");
    return false;
  }

  const isDuplicate = students.some(
    (student) =>
      student.email?.toLowerCase() === emailValue.toLowerCase() &&
      student.id !== studentId,
  );

  if (isDuplicate) {
    setEmailError("This email is already in use");
    return false;
  }

  setEmailError("");
  return true;
};

const openCreateDialog = () => {
  setFirstName("");
  setLastName("");
  setEmail("");
  setEmailError("");
  setFirstNameError("");
  setLastNameError("");
  setIsCreateDialogOpen(true);
};

const openEditDialog = (student: Student) => {
  setCurrentStudent(student);
  setFirstName(student.first_name);
  setLastName(student.last_name);
  setEmail(student.email || "");
  setEmailError("");
  setFirstNameError("");
  setLastNameError("");
  setIsEditDialogOpen(true);
```

```
};

const openDeleteDialog = (student: Student) => {
  setCurrentStudent(student);
  setIsDeleteDialogOpen(true);
};

const handleCreateStudent = async () => {
  const isFirstNameValid = validateFirstName(firstName);
  const isLastNameValid = validateLastName(lastName);
  const isEmailValid = validateEmail(email);

  if (!isFirstNameValid || !isLastNameValid || !isEmailValid) return;

  try {
    await createStudent(firstName, lastName, email || undefined);
    await refreshData();
    setIsCreateDialogOpen(false);
  } catch (error) {
    console.error("Error creating student:", error);
  }
};

const handleUpdateStudent = async () => {
  if (!currentStudent) return;

  const isFirstNameValid = validateFirstName(firstName);
  const isLastNameValid = validateLastName(lastName);
  const isEmailValid = validateEmail(email, currentStudent.id);

  if (!isFirstNameValid || !isLastNameValid || !isEmailValid) return;

  try {
    await updateStudent(
      currentStudent.id,
      firstName,
      lastName,
      email || undefined,
    );
    await refreshData();
    setIsEditDialogOpen(false);
  } catch (error) {
```

```
    console.error("Error updating student:", error);
  }
};

const handleDeleteStudent = async () => {
  if (!currentStudent) return;

  try {
    await deleteStudent(currentStudent.id);
    await refreshData();
    setIsDeleteDialogOpen(false);
  } catch (error) {
    console.error("Error deleting student:", error);
  }
};

const handleFirstNameChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  const value = e.target.value;
  setFirstName(value);
  validateFirstName(value);
};

const handleLastNameChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  const value = e.target.value;
  setLastName(value);
  validateLastName(value);
};

const handleEmailChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  const value = e.target.value;
  setEmail(value);
  validateEmail(value, currentStudent?.id);
};

return (
  <div className="space-y-6">
    <div className="flex items-center justify-between">
      <h2 className="text-3xl font-bold tracking-tight">Students</h2>
      <Button onClick={openCreateDialog}>
        <PlusCircle className="mr-2 h-4 w-4" />
        Add Student
      </Button>
```

```
</div>

<div className="flex items-center space-x-2">
  <div className="relative flex-1">
    <Search className="absolute left-2.5 top-2.5 h-4 w-4 text-gray-500" />
    <Input
      type="search"
      placeholder="Search students..."
      className="pl-8"
      value={searchQuery}
      onChange={(e) => setSearchQuery(e.target.value)}
    />
  </div>
</div>

<Card>
  <CardHeader>
    <CardTitle>All Students</CardTitle>
    <CardDescription>
      Manage your students and their information.
    </CardDescription>
  </CardHeader>
  <CardContent>
    {loading ? (
      <div className="space-y-2">
        <Skeleton className="h-8 w-full" />
        <Skeleton className="h-8 w-full" />
        <Skeleton className="h-8 w-full" />
        <Skeleton className="h-8 w-full" />
      </div>
    ) : (
      <Table>
        <TableHeader>
          <TableRow>
            <TableHead>ID</TableHead>
            <TableHead>Name</TableHead>
            <TableHead>Email</TableHead>
            <TableHead className="text-right">Actions</TableHead>
          </TableRow>
        </TableHeader>
        <TableBody>
          {filteredStudents.length > 0 ? (
```

```
  filteredStudents.map((student) => (
    <TableRow key={student.id}>
      <TableCell className="font-medium">
        {student.id}
      </TableCell>
      <TableCell>
        {student.first_name} {student.last_name}
      </TableCell>
      <TableCell>{student.email || "-"}</TableCell>
      <TableCell className="text-right">
        <div className="flex justify-end gap-2">
          <Button
            variant="ghost"
            size="icon"
            onClick={() => openEditDialog(student)}
          >
            <Pencil className="h-4 w-4" />
            <span className="sr-only">Edit</span>
          </Button>
          <Button
            variant="ghost"
            size="icon"
            className="text-red-500"
            onClick={() => openDeleteDialog(student)}
          >
            <Trash2 className="h-4 w-4" />
            <span className="sr-only">Delete</span>
          </Button>
        </div>
      </TableCell>
    </TableRow>
  ))
) : (
  <TableRow>
    <TableCell
      colSpan={4}
      className="text-center py-4 text-gray-500"
    >
      No students found.
    </TableCell>
  </TableRow>
)}
```

```
        </TableBody>
      </Table>
    )}
  </CardContent>
  <CardFooter className="border-t px-6 py-4">
    <div className="text-xs text-gray-500">
      Showing {filteredStudents.length} of {students.length} students
    </div>
  </CardFooter>
</Card>

<Dialog open={isCreateDialogOpen} onOpenChange={setIsCreateDialogOpen}>
  <DialogContent>
    <DialogHeader>
      <DialogTitle>Add New Student</DialogTitle>
      <DialogDescription>
        Enter the information for the new student.
      </DialogDescription>
    </DialogHeader>
    <div className="grid gap-4 py-4">
      <div className="grid grid-cols-4 items-center gap-4">
        <Label htmlFor="firstName" className="text-right">
          First Name
        </Label>
        <div className="col-span-3 space-y-1">
          <Input
            id="firstName"
            value={firstName}
            onChange={handleFirstNameChange}
            className={firstNameError ? "border-red-500" : ""}
          />
          {firstNameError && (
            <p className="text-xs text-red-500">{firstNameError}</p>
          )}
        </div>
      </div>
      <div className="grid grid-cols-4 items-center gap-4">
        <Label htmlFor="lastName" className="text-right">
          Last Name
        </Label>
        <div className="col-span-3 space-y-1">
          <Input
```

```
        id="lastName"
        value={lastName}
        onChange={handleLastNameChange}
        className={lastNameError ? "border-red-500" : ""}
      />
      {lastNameError && (
        <p className="text-xs text-red-500">{lastNameError}</p>
      )}
    </div>
  </div>
  <div className="grid grid-cols-4 items-center gap-4">
    <Label htmlFor="email" className="text-right">
      Email
    </Label>
    <div className="col-span-3 space-y-1">
      <Input
        id="email"
        type="email"
        value={email}
        onChange={handleEmailChange}
        className={emailError ? "border-red-500" : ""}
      />
      {emailError && (
        <p className="text-xs text-red-500">{emailError}</p>
      )}
    </div>
  </div>
</div>
<DialogFooter>
  <Button
    variant="outline"
    onClick={() => setIsCreateDialogOpen(false)}
  >
    Cancel
  </Button>
  <Button
    onClick={handleCreateStudent}
    disabled={
      !firstName ||
      !lastName ||
      !!firstNameError ||
      !!lastNameError ||
```

```
          !!emailError
        }
      >
        Create Student
      </Button>
    </DialogFooter>
  </DialogContent>
</Dialog>

<Dialog open={isEditDialogOpen} onOpenChange={setIsEditDialogOpen}>
  <DialogContent>
    <DialogHeader>
      <DialogTitle>Edit Student</DialogTitle>
      <DialogDescription>
        Update the student's information.
      </DialogDescription>
    </DialogHeader>
    <div className="grid gap-4 py-4">
      <div className="grid grid-cols-4 items-center gap-4">
        <Label htmlFor="edit-firstName" className="text-right">
          First Name
        </Label>
        <div className="col-span-3 space-y-1">
          <Input
            id="edit-firstName"
            value={firstName}
            onChange={handleFirstNameChange}
            className={firstNameError ? "border-red-500" : ""}
          />
          {firstNameError && (
            <p className="text-xs text-red-500">{firstNameError}</p>
          )}
        </div>
      </div>
      <div className="grid grid-cols-4 items-center gap-4">
        <Label htmlFor="edit-lastName" className="text-right">
          Last Name
        </Label>
        <div className="col-span-3 space-y-1">
          <Input
            id="edit-lastName"
            value={lastName}
```

```
        onChange={handleLastNameChange}
        className={lastNameError ? "border-red-500" : ""}
      />
      {lastNameError && (
        <p className="text-xs text-red-500">{lastNameError}</p>
      )}
    </div>
  </div>
  <div className="grid grid-cols-4 items-center gap-4">
    <Label htmlFor="edit-email" className="text-right">
      Email
    </Label>
    <div className="col-span-3 space-y-1">
      <Input
        id="edit-email"
        type="email"
        value={email}
        onChange={handleEmailChange}
        className={emailError ? "border-red-500" : ""}
      />
      {emailError && (
        <p className="text-xs text-red-500">{emailError}</p>
      )}
    </div>
  </div>
</div>
<DialogFooter>
  <Button
    variant="outline"
    onClick={() => setIsEditDialogOpen(false)}
  >
    Cancel
  </Button>
  <Button
    onClick={handleUpdateStudent}
    disabled={
      !firstName ||
      !lastName ||
      !!firstNameError ||
      !!lastNameError ||
      !!emailError
    }
```

```
        >
          Update Student
        </Button>
      </DialogFooter>
    </DialogContent>
  </Dialog>

  <Dialog open={isDeleteDialogOpen} onOpenChange={setIsDeleteDialogOpen}>
    <DialogContent>
      <DialogHeader>
        <DialogTitle>Delete Student</DialogTitle>
        <DialogDescription>
          Are you sure you want to delete this student? This action cannot
          be undone.
        </DialogDescription>
      </DialogHeader>
      <div className="py-4">
        {currentStudent && (
          <p>
            You are about to delete{" "}
            <strong>
              {currentStudent.first_name} {currentStudent.last_name}
            </strong>
            .
          </p>
        )}
      </div>
      <DialogFooter>
        <Button
          variant="outline"
          onClick={() => setIsDeleteDialogOpen(false)}
        >
          Cancel
        </Button>
        <Button variant="destructive" onClick={handleDeleteStudent}>
          Delete Student
        </Button>
      </DialogFooter>
    </DialogContent>
  </Dialog>
 </div>
);
```

}

src/components/classes-view.tsx: Class management UI

```
import { useState } from "react";
import {
  Card,
  CardContent,
  CardDescription,
  CardFooter,
  CardHeader,
  CardTitle,
} from "@/components/ui/card";
import { Button } from "@/components/ui/button";
import {
  Dialog,
  DialogContent,
  DialogDescription,
  DialogFooter,
  DialogHeader,
  DialogTitle,
} from "@/components/ui/dialog";
import { Input } from "@/components/ui/input";
import { Label } from "@/components/ui/label";
import {
  Table,
  TableBody,
  TableCell,
  TableHead,
  TableHeader,
  TableRow,
} from "@/components/ui/table";
import { Textarea } from "@/components/ui/textarea";
import { Skeleton } from "@/components/ui/skeleton";
import { PlusCircle, Pencil, Trash2, Search } from "lucide-react";
import { createClass, updateClass, deleteClass } from "@/api/classes";
import type { Class } from "@/api/types";

interface ClassesViewProps {
  classes: Class[];
  refreshData: () => Promise<void>;
  loading: boolean;
}
```

```
export default function ClassesView({
  classes,
  refreshData,
  loading,
}: ClassesViewProps) {
  const [searchQuery, setSearchQuery] = useState("");
  const [isCreateDialogOpen, setIsCreateDialogOpen] = useState(false);
  const [isEditDialogOpen, setIsEditDialogOpen] = useState(false);
  const [isDeleteDialogOpen, setIsDeleteDialogOpen] = useState(false);
  const [currentClass, setCurrentClass] = useState<Class | null>(null);

  const [className, setClassName] = useState("");
  const [description, setDescription] = useState("");

  const filteredClasses = classes.filter((classItem) => {
    const classNameLower = classItem.class_name.toLowerCase();
    const descriptionLower = classItem.description?.toLowerCase() || "";
    const query = searchQuery.toLowerCase();

    return classNameLower.includes(query) || descriptionLower.includes(query);
  });

  const openCreateDialog = () => {
    setClassName("");
    setDescription("");
    setIsCreateDialogOpen(true);
  };

  const openEditDialog = (classItem: Class) => {
    setCurrentClass(classItem);
    setClassName(classItem.class_name);
    setDescription(classItem.description || "");
    setIsEditDialogOpen(true);
  };

  const openDeleteDialog = (classItem: Class) => {
    setCurrentClass(classItem);
    setIsDeleteDialogOpen(true);
  };

  const handleCreateClass = async () => {
    try {
```

```
    await createClass(className, description || undefined);
    await refreshData();
    setIsCreateDialogOpen(false);
  } catch (error) {
    console.error("Error creating class:", error);
  }
};

const handleUpdateClass = async () => {
  if (!currentClass) return;

  try {
    await updateClass(currentClass.id, className, description || undefined);
    await refreshData();
    setIsEditDialogOpen(false);
  } catch (error) {
    console.error("Error updating class:", error);
  }
};

const handleDeleteClass = async () => {
  if (!currentClass) return;

  try {
    await deleteClass(currentClass.id);
    await refreshData();
    setIsDeleteDialogOpen(false);
  } catch (error) {
    console.error("Error deleting class:", error);
  }
};

return (
  <div className="space-y-6">
    <div className="flex items-center justify-between">
      <h2 className="text-3xl font-bold tracking-tight">Classes</h2>
      <Button onClick={openCreateDialog}>
        <PlusCircle className="mr-2 h-4 w-4" />
        Add Class
      </Button>
    </div>
```

```
<div className="flex items-center space-x-2">
  <div className="relative flex-1">
    <Search className="absolute left-2.5 top-2.5 h-4 w-4 text-gray-500" />
    <Input
      type="search"
      placeholder="Search classes..."
      className="pl-8"
      value={searchQuery}
      onChange={(e) => setSearchQuery(e.target.value)}
    />
  </div>
</div>

<Card>
  <CardHeader>
    <CardTitle>All Classes</CardTitle>
    <CardDescription>
      Manage your classes and their descriptions.
    </CardDescription>
  </CardHeader>
  <CardContent>
    {loading ? (
      <div className="space-y-2">
        <Skeleton className="h-8 w-full" />
        <Skeleton className="h-8 w-full" />
        <Skeleton className="h-8 w-full" />
        <Skeleton className="h-8 w-full" />
      </div>
    ) : (
      <Table>
        <TableHeader>
          <TableRow>
            <TableHead>ID</TableHead>
            <TableHead>Class Name</TableHead>
            <TableHead>Description</TableHead>
            <TableHead className="text-right">Actions</TableHead>
          </TableRow>
        </TableHeader>
        <TableBody>
          {filteredClasses.length > 0 ? (
            filteredClasses.map((classItem) => (
              <TableRow key={classItem.id}>
```

```
          <TableCell className="font-medium">
            {classItem.id}
          </TableCell>
          <TableCell>{classItem.class_name}</TableCell>
          <TableCell className="max-w-xs truncate">
            {classItem.description || "-"}
          </TableCell>
          <TableCell className="text-right">
            <div className="flex justify-end gap-2">
              <Button
                variant="ghost"
                size="icon"
                onClick={() => openEditDialog(classItem)}
              >
                <Pencil className="h-4 w-4" />
                <span className="sr-only">Edit</span>
              </Button>
              <Button
                variant="ghost"
                size="icon"
                className="text-red-500"
                onClick={() => openDeleteDialog(classItem)}
              >
                <Trash2 className="h-4 w-4" />
                <span className="sr-only">Delete</span>
              </Button>
            </div>
          </TableCell>
        </TableRow>
      ))
    ) : (
      <TableRow>
        <TableCell
          colSpan={4}
          className="text-center py-4 text-gray-500"
        >
          No classes found.
        </TableCell>
      </TableRow>
    )}
  </TableBody>
</Table>
```

```
    )}
  </CardContent>
  <CardFooter className="border-t px-6 py-4">
    <div className="text-xs text-gray-500">
      Showing {filteredClasses.length} of {classes.length} classes
    </div>
  </CardFooter>
</Card>

<Dialog open={isCreateDialogOpen} onOpenChange={setIsCreateDialogOpen}>
  <DialogContent>
    <DialogHeader>
      <DialogTitle>Add New Class</DialogTitle>
      <DialogDescription>
        Enter the information for the new class.
      </DialogDescription>
    </DialogHeader>
    <div className="grid gap-4 py-4">
      <div className="grid grid-cols-4 items-center gap-4">
        <Label htmlFor="className" className="text-right">
          Class Name
        </Label>
        <Input
          id="className"
          value={className}
          onChange={(e) => setClassName(e.target.value)}
          className="col-span-3"
          required
        />
      </div>
      <div className="grid grid-cols-4 items-center gap-4">
        <Label htmlFor="description" className="text-right">
          Description
        </Label>
        <Textarea
          id="description"
          value={description}
          onChange={(e) => setDescription(e.target.value)}
          className="col-span-3"
          rows={3}
        />
      </div>
```

```
    </div>
    <DialogFooter>
      <Button
        variant="outline"
        onClick={() => setIsCreateDialogOpen(false)}
      >
        Cancel
      </Button>
      <Button onClick={handleCreateClass} disabled={!className}>
        Create Class
      </Button>
    </DialogFooter>
  </DialogContent>
</Dialog>

<Dialog open={isEditDialogOpen} onOpenChange={setIsEditDialogOpen}>
  <DialogContent>
    <DialogHeader>
      <DialogTitle>Edit Class</DialogTitle>
      <DialogDescription>Update the class information.</DialogDescription>
    </DialogHeader>
    <div className="grid gap-4 py-4">
      <div className="grid grid-cols-4 items-center gap-4">
        <Label htmlFor="edit-className" className="text-right">
          Class Name
        </Label>
        <Input
          id="edit-className"
          value={className}
          onChange={(e) => setClassName(e.target.value)}
          className="col-span-3"
          required
        />
      </div>
      <div className="grid grid-cols-4 items-center gap-4">
        <Label htmlFor="edit-description" className="text-right">
          Description
        </Label>
        <Textarea
          id="edit-description"
          value={description}
          onChange={(e) => setDescription(e.target.value)}
```

```
          className="col-span-3"
          rows={3}
        />
      </div>
    </div>
    <DialogFooter>
      <Button
        variant="outline"
        onClick={() => setIsEditDialogOpen(false)}
      >
        Cancel
      </Button>
      <Button onClick={handleUpdateClass} disabled={!className}>
        Update Class
      </Button>
    </DialogFooter>
  </DialogContent>
</Dialog>

<Dialog open={isDeleteDialogOpen} onOpenChange={setIsDeleteDialogOpen}>
  <DialogContent>
    <DialogHeader>
      <DialogTitle>Delete Class</DialogTitle>
      <DialogDescription>
        Are you sure you want to delete this class? This action cannot be
        undone.
      </DialogDescription>
    </DialogHeader>
    <div className="py-4">
      {currentClass && (
        <p>
          You are about to delete{" "}
          <strong>{currentClass.class_name}</strong>.
        </p>
      )}
    </div>
    <DialogFooter>
      <Button
        variant="outline"
        onClick={() => setIsDeleteDialogOpen(false)}
      >
        Cancel
```

```
          </Button>
          <Button variant="destructive" onClick={handleDeleteClass}>
            Delete Class
          </Button>
        </DialogFooter>
      </DialogContent>
    </Dialog>
  </div>
 );
}
```

src/components/assignments-view.tsx: Assignment management UI

```
import { useState } from "react";
import {
 Card,
 CardContent,
 CardDescription,
 CardFooter,
 CardHeader,
 CardTitle,
} from "@/components/ui/card";
import { Button } from "@/components/ui/button";
import {
 Dialog,
 DialogContent,
 DialogDescription,
 DialogFooter,
 DialogHeader,
 DialogTitle,
} from "@/components/ui/dialog";
import { Input } from "@/components/ui/input";
import { Label } from "@/components/ui/label";
import {
 Table,
 TableBody,
 TableCell,
 TableHead,
 TableHeader,
 TableRow,
} from "@/components/ui/table";
import {
 Select,
 SelectContent,
```

```
  SelectItem,
  SelectTrigger,
  SelectValue,
} from "@/components/ui/select";
import { Skeleton } from "@/components/ui/skeleton";
import { PlusCircle, Pencil, Trash2, Search } from "lucide-react";
import { format } from "date-fns";
import {
  createAssignment,
  updateAssignment,
  deleteAssignment,
} from "@/api/assignments";
import type { Assignment, Class } from "@/api/types";
import { DatePicker } from "./date-picker";

interface AssignmentsViewProps {
  assignments: Assignment[];
  classes: Class[];
  refreshData: () => Promise<void>;
  loading: boolean;
}

export default function AssignmentsView({
  assignments,
  classes,
  refreshData,
  loading,
}: AssignmentsViewProps) {
  const [searchQuery, setSearchQuery] = useState("");
  const [isCreateDialogOpen, setIsCreateDialogOpen] = useState(false);
  const [isEditDialogOpen, setIsEditDialogOpen] = useState(false);
  const [isDeleteDialogOpen, setIsDeleteDialogOpen] = useState(false);
  const [currentAssignment, setCurrentAssignment] = useState<Assignment | null>(
    null,
  );

  const [assignmentName, setAssignmentName] = useState("");
  const [assignmentType, setAssignmentType] = useState<string>("Homework");
  const [classId, setClassId] = useState<number | null>(null);
  const [maximumScore, setMaximumScore] = useState<number>(100);
  const [dueDate, setDueDate] = useState<Date | undefined>(undefined);
```

```
const assignmentTypes = ["Homework", "Test"];

const filteredAssignments = assignments.filter((assignment) => {
  const assignmentNameLower = assignment.assignment_name.toLowerCase();
  const assignmentTypeLower = assignment.assignment_type.toLowerCase();
  const query = searchQuery.toLowerCase();

  return (
    assignmentNameLower.includes(query) || assignmentTypeLower.includes(query)
  );
});

const getClassName = (classId: number) => {
  const foundClass = classes.find((c) => c.id === classId);
  return foundClass ? foundClass.class_name : "Unknown";
};

const formatDate = (dateString?: string) => {
  if (!dateString) return "No due date";
  try {
    return format(new Date(dateString), "PPP");
  } catch (error) {
    return "Invalid date";
  }
};

const openCreateDialog = () => {
  setAssignmentName("");
  setAssignmentType("Homework");
  setClassId(null);
  setMaximumScore(100);
  setDueDate(undefined);
  setIsCreateDialogOpen(true);
};

const openEditDialog = (assignment: Assignment) => {
  setCurrentAssignment(assignment);
  setAssignmentName(assignment.assignment_name);
  setAssignmentType(assignment.assignment_type);
  setClassId(assignment.class_id);
  setMaximumScore(assignment.maximum_score);
  setDueDate(assignment.due_date ? new Date(assignment.due_date) : undefined);
```

```
  setIsEditDialogOpen(true);
};

const openDeleteDialog = (assignment: Assignment) => {
  setCurrentAssignment(assignment);
  setIsDeleteDialogOpen(true);
};

const handleCreateAssignment = async () => {
  if (!classId) return;

  try {
    let dueDateString = undefined;
    if (dueDate) {
      dueDateString = dueDate.toISOString().replace("Z", "");
    }

    await createAssignment(
      classId,
      assignmentName,
      assignmentType,
      maximumScore,
      dueDateString,
    );
    await refreshData();
    setIsCreateDialogOpen(false);
  } catch (error) {
    console.error("Error creating assignment:", error);
  }
};

const handleUpdateAssignment = async () => {
  if (!currentAssignment || !classId) return;

  try {
    let dueDateString = undefined;
    if (dueDate) {
      dueDateString = dueDate.toISOString().replace("Z", "");
    }

    await updateAssignment(
      currentAssignment.id,
```

```
      classId,
      assignmentName,
      assignmentType,
      maximumScore,
      dueDateString,
    );
    await refreshData();
    setIsEditDialogOpen(false);
  } catch (error) {
    console.error("Error updating assignment:", error);
  }
};

const handleDeleteAssignment = async () => {
  if (!currentAssignment) return;

  try {
    await deleteAssignment(currentAssignment.id);
    await refreshData();
    setIsDeleteDialogOpen(false);
  } catch (error) {
    console.error("Error deleting assignment:", error);
  }
};

return (
  <div className="space-y-6">
    <div className="flex items-center justify-between">
      <h2 className="text-3xl font-bold tracking-tight">Assignments</h2>
      <Button onClick={openCreateDialog}>
        <PlusCircle className="mr-2 h-4 w-4" />
        Add Assignment
      </Button>
    </div>

    {/* Search */}
    <div className="flex items-center space-x-2">
      <div className="relative flex-1">
        <Search className="absolute left-2.5 top-2.5 h-4 w-4 text-gray-500" />
        <Input
          type="search"
          placeholder="Search assignments..."
```

```
      className="pl-8"
      value={searchQuery}
      onChange={(e) => setSearchQuery(e.target.value)}
    />
  </div>
</div>

<Card>
  <CardHeader>
    <CardTitle>All Assignments</CardTitle>
    <CardDescription>
      Manage assignments across all your classes.
    </CardDescription>
  </CardHeader>
  <CardContent>
    {loading ? (
      <div className="space-y-2">
        <Skeleton className="h-8 w-full" />
        <Skeleton className="h-8 w-full" />
        <Skeleton className="h-8 w-full" />
        <Skeleton className="h-8 w-full" />
      </div>
    ) : (
      <Table>
        <TableHeader>
          <TableRow>
            <TableHead>ID</TableHead>
            <TableHead>Assignment Name</TableHead>
            <TableHead>Type</TableHead>
            <TableHead>Class</TableHead>
            <TableHead>Max Score</TableHead>
            <TableHead>Due Date</TableHead>
            <TableHead className="text-right">Actions</TableHead>
          </TableRow>
        </TableHeader>
        <TableBody>
          {filteredAssignments.length > 0 ? (
            filteredAssignments.map((assignment) => (
              <TableRow key={assignment.id}>
                <TableCell className="font-medium">
                  {assignment.id}
                </TableCell>
```

```
            <TableCell>{assignment.assignment_name}</TableCell>
            <TableCell>{assignment.assignment_type}</TableCell>
            <TableCell>{getClassName(assignment.class_id)}</TableCell>
            <TableCell>{assignment.maximum_score}</TableCell>
            <TableCell>{formatDate(assignment.due_date)}</TableCell>
            <TableCell className="text-right">
              <div className="flex justify-end gap-2">
                <Button
                  variant="ghost"
                  size="icon"
                  onClick={() => openEditDialog(assignment)}
                >
                  <Pencil className="h-4 w-4" />
                  <span className="sr-only">Edit</span>
                </Button>
                <Button
                  variant="ghost"
                  size="icon"
                  className="text-red-500"
                  onClick={() => openDeleteDialog(assignment)}
                >
                  <Trash2 className="h-4 w-4" />
                  <span className="sr-only">Delete</span>
                </Button>
              </div>
            </TableCell>
          </TableRow>
        ))
      ) : (
        <TableRow>
          <TableCell
            colSpan={7}
            className="text-center py-4 text-gray-500"
          >
            No assignments found.
          </TableCell>
        </TableRow>
      )}
    </TableBody>
  </Table>
)}
</CardContent>
```

```
  <CardFooter className="border-t px-6 py-4">
    <div className="text-xs text-gray-500">
      Showing {filteredAssignments.length} of {assignments.length}{" "}
      assignments
    </div>
  </CardFooter>
</Card>

<Dialog open={isCreateDialogOpen} onOpenChange={setIsCreateDialogOpen}>
  <DialogContent className="sm:max-w-[525px]">
    <DialogHeader>
      <DialogTitle>Add New Assignment</DialogTitle>
      <DialogDescription>
        Enter the details for the new assignment.
      </DialogDescription>
    </DialogHeader>
    <div className="grid gap-4 py-4">
      <div className="grid grid-cols-4 items-center gap-4">
        <Label htmlFor="assignmentName" className="text-right">
          Name
        </Label>
        <Input
          id="assignmentName"
          value={assignmentName}
          onChange={(e) => setAssignmentName(e.target.value)}
          className="col-span-3"
          required
        />
      </div>
      <div className="grid grid-cols-4 items-center gap-4">
        <Label htmlFor="assignmentClass" className="text-right">
          Class
        </Label>
        <Select
          value={classId?.toString() || "selectClass"}
          onValueChange={(value) => {
            if (value !== "selectClass") {
              setClassId(parseInt(value));
            }
          }}
        >
          <SelectTrigger className="col-span-3">
```

```
      <SelectValue placeholder="Select a class" />
    </SelectTrigger>
    <SelectContent>
      {classes.map((classItem) => (
        <SelectItem
          key={classItem.id}
          value={classItem.id.toString()}
        >
          {classItem.class_name}
        </SelectItem>
      ))}
    </SelectContent>
  </Select>
</div>
<div className="grid grid-cols-4 items-center gap-4">
  <Label htmlFor="assignmentType" className="text-right">
    Type
  </Label>
  <Select
    value={assignmentType || "Homework"}
    onValueChange={(value) => setAssignmentType(value)}
  >
    <SelectTrigger className="col-span-3">
      <SelectValue placeholder="Select a type" />
    </SelectTrigger>
    <SelectContent>
      {assignmentTypes.map((type) => (
        <SelectItem key={type} value={type}>
          {type}
        </SelectItem>
      ))}
    </SelectContent>
  </Select>
</div>
<div className="grid grid-cols-4 items-center gap-4">
  <Label htmlFor="maximumScore" className="text-right">
    Max Score
  </Label>
  <Input
    id="maximumScore"
    type="number"
    value={maximumScore === 0 ? "" : maximumScore}
```

```
            onChange={(e) => setMaximumScore(Number(e.target.value))}
            className="col-span-3"
            min={0}
            required
          />
        </div>
        <div className="grid grid-cols-4 items-center gap-4">
          <Label htmlFor="dueDate" className="text-right">
            Due Date
          </Label>
          <div className="col-span-3">
            <DatePicker value={dueDate} onChange={setDueDate} />
          </div>
        </div>
      </div>
      <DialogFooter>
        <Button
          variant="outline"
          onClick={() => setIsCreateDialogOpen(false)}
        >
          Cancel
        </Button>
        <Button
          onClick={handleCreateAssignment}
          disabled={!assignmentName || !assignmentType || !classId}
        >
          Create Assignment
        </Button>
      </DialogFooter>
    </DialogContent>
  </Dialog>

  <Dialog open={isEditDialogOpen} onOpenChange={setIsEditDialogOpen}>
    <DialogContent className="sm:max-w-[525px]">
      <DialogHeader>
        <DialogTitle>Edit Assignment</DialogTitle>
        <DialogDescription>
          Update the assignment details.
        </DialogDescription>
      </DialogHeader>
      <div className="grid gap-4 py-4">
        <div className="grid grid-cols-4 items-center gap-4">
```

```
  <Label htmlFor="edit-assignmentName" className="text-right">
    Name
  </Label>
  <Input
    id="edit-assignmentName"
    value={assignmentName}
    onChange={(e) => setAssignmentName(e.target.value)}
    className="col-span-3"
    required
  />
</div>
<div className="grid grid-cols-4 items-center gap-4">
  <Label htmlFor="edit-assignmentClass" className="text-right">
    Class
  </Label>
  <Select
    value={classId?.toString() || "selectClass"}
    onValueChange={(value) => {
      if (value !== "selectClass") {
        setClassId(parseInt(value));
      }
    }}
  >
    <SelectTrigger className="col-span-3">
      <SelectValue placeholder="Select a class" />
    </SelectTrigger>
    <SelectContent>
      {classes.map((classItem) => (
        <SelectItem
          key={classItem.id}
          value={classItem.id.toString()}
        >
          {classItem.class_name}
        </SelectItem>
      ))}
    </SelectContent>
  </Select>
</div>
<div className="grid grid-cols-4 items-center gap-4">
  <Label htmlFor="edit-assignmentType" className="text-right">
    Type
  </Label>
```

```
    <Select
      value={assignmentType || "Homework"}
      onValueChange={(value) => setAssignmentType(value)}
    >
      <SelectTrigger className="col-span-3">
        <SelectValue placeholder="Select a type" />
      </SelectTrigger>
      <SelectContent>
        {assignmentTypes.map((type) => (
          <SelectItem key={type} value={type}>
            {type}
          </SelectItem>
        ))}
      </SelectContent>
    </Select>
  </div>
  <div className="grid grid-cols-4 items-center gap-4">
    <Label htmlFor="edit-maximumScore" className="text-right">
      Max Score
    </Label>
    <Input
      id="edit-maximumScore"
      type="number"
      value={maximumScore === 0 ? "" : maximumScore}
      onChange={(e) => setMaximumScore(Number(e.target.value))}
      className="col-span-3"
      min={0}
      placeholder="Enter Maximum Score"
      required
    />
  </div>
  <div className="grid grid-cols-4 items-center gap-4">
    <Label htmlFor="edit-dueDate" className="text-right">
      Due Date
    </Label>
    <div className="col-span-3">
      <DatePicker value={dueDate} onChange={setDueDate} />
    </div>
  </div>
</div>
<DialogFooter>
  <Button
```

```
        variant="outline"
        onClick={() => setIsEditDialogOpen(false)}
      >
        Cancel
      </Button>
      <Button
        onClick={handleUpdateAssignment}
        disabled={!assignmentName || !assignmentType || !classId}
      >
        Update Assignment
      </Button>
    </DialogFooter>
  </DialogContent>
</Dialog>

<Dialog open={isDeleteDialogOpen} onOpenChange={setIsDeleteDialogOpen}>
  <DialogContent>
    <DialogHeader>
      <DialogTitle>Delete Assignment</DialogTitle>
      <DialogDescription>
        Are you sure you want to delete this assignment? This action
        cannot be undone.
      </DialogDescription>
    </DialogHeader>
    <div className="py-4">
      {currentAssignment && (
        <p>
          You are about to delete{" "}
          <strong>{currentAssignment.assignment_name}</strong>.
        </p>
      )}
    </div>
    <DialogFooter>
      <Button
        variant="outline"
        onClick={() => setIsDeleteDialogOpen(false)}
      >
        Cancel
      </Button>
      <Button variant="destructive" onClick={handleDeleteAssignment}>
        Delete Assignment
      </Button>
```

```
        </DialogFooter>
      </DialogContent>
    </Dialog>
  </div>
 );
}
```

src/components/grades-view.tsx: Grade management UI

```
import { useState, useEffect } from "react";
import {
 Card,
 CardContent,
 CardDescription,
 CardFooter,
 CardHeader,
 CardTitle,
} from "@/components/ui/card";
import { Button } from "@/components/ui/button";
import {
 Dialog,
 DialogContent,
 DialogDescription,
 DialogFooter,
 DialogHeader,
 DialogTitle,
} from "@/components/ui/dialog";
import { Input } from "@/components/ui/input";
import { Label } from "@/components/ui/label";
import {
 Table,
 TableBody,
 TableCell,
 TableHead,
 TableHeader,
 TableRow,
} from "@/components/ui/table";
import {
 Select,
 SelectContent,
 SelectItem,
 SelectTrigger,
 SelectValue,
} from "@/components/ui/select";
```

```
import { Skeleton } from "@/components/ui/skeleton";
import { PlusCircle, Search, Pencil, AlertCircle } from "lucide-react";
import { Badge } from "@/components/ui/badge";
import { Alert, AlertDescription, AlertTitle } from "@/components/ui/alert";
import { createGrade, updateGrade, getGrade, getAllGrades } from "@/api/grades";
import {
  enrollStudent,
  unenrollStudent,
  getEnrollments,
} from "@/api/student-classes";
import type {
  OverallGrade,
  Student,
  Class,
  Assignment,
  StudentClass,
  Grade,
} from "@/api/types";

interface GradesViewProps {
  grades: OverallGrade[];
  students: Student[];
  classes: Class[];
  assignments: Assignment[];
  refreshData: () => Promise<void>;
  loading: boolean;
}

export default function GradesView({
  grades,
  students,
  classes,
  assignments,
  refreshData,
  loading,
}: GradesViewProps) {
  const [searchQuery, setSearchQuery] = useState("");
  const [selectedClass, setSelectedClass] = useState<number | null>(null);

  const [isGradeDialogOpen, setIsGradeDialogOpen] = useState(false);
  const [isEnrollDialogOpen, setIsEnrollDialogOpen] = useState(false);
  const [isUnenrollDialogOpen, setIsUnenrollDialogOpen] = useState(false);
```

```
const [isEditGradeDialogOpen, setIsEditGradeDialogOpen] = useState(false);

const [selectedStudent, setSelectedStudent] = useState<number | null>(null);
const [selectedAssignment, setSelectedAssignment] = useState<number | null>(
  null,
);
const [score, setScore] = useState<number>(0);
const [enrollStudentId, setEnrollStudentId] = useState<number | null>(null);
const [enrollClassId, setEnrollClassId] = useState<number | null>(null);
const [editingGrades, setEditingGrades] = useState<Grade[]>([]);
const [allGrades, setAllGrades] = useState<Grade[]>([]);

const [studentToUnenroll, setStudentToUnenroll] = useState<{
  id: number;
  name: string;
} | null>(null);
const [classToUnenroll, setClassToUnenroll] = useState<{
  id: number;
  name: string;
} | null>(null);

const [error, setError] = useState<string | null>(null);
const [isSubmitting, setIsSubmitting] = useState(false);
const [enrollments, setEnrollments] = useState<StudentClass[]>([]);

const loadEnrollments = async () => {
  try {
    const allEnrollments = await getEnrollments();
    setEnrollments(allEnrollments);
  } catch (error) {
    console.error("Error loading enrollments:", error);
  }
};

const loadAllGrades = async () => {
  try {
    const grades = await getAllGrades();
    setAllGrades(grades);
  } catch (error) {
    console.error("Error loading grades:", error);
  }
};
```

```
useEffect(() => {
  loadEnrollments();
  loadAllGrades();
}, []);

useEffect(() => {
  if (selectedClass) {
    setEnrollClassId(selectedClass);
  }
}, [selectedClass]);

const filteredGrades = grades.filter((grade) => {
  const student = students.find((s) => s.id === grade.student_id);
  const classItem = classes.find((c) => c.id === grade.class_id);

  if (!student || !classItem) return false;

  const studentName =
    `${student.first_name} ${student.last_name}`.toLowerCase();
  const className = classItem.class_name.toLowerCase();
  const query = searchQuery.toLowerCase();

  const matchesSearch =
    studentName.includes(query) || className.includes(query);
  const matchesClass = selectedClass
    ? grade.class_id === selectedClass
    : true;

  return matchesSearch && matchesClass;
});

const getStudentName = (studentId: number): string => {
  const student = students.find((s) => s.id === studentId);
  return student ? `${student.first_name} ${student.last_name}` : "Unknown";
};

const getClassName = (classId: number): string => {
  const classItem = classes.find((c) => c.id === classId);
  return classItem ? classItem.class_name : "Unknown";
};
```

```
const getAssignmentName = (assignmentId: number): string => {
  const assignment = assignments.find((a) => a.id === assignmentId);
  return assignment ? assignment.assignment_name : "Unknown";
};

const getAssignmentMaxScore = (assignmentId: number): number => {
  const assignment = assignments.find((a) => a.id === assignmentId);
  return assignment ? assignment.maximum_score : 0;
};

const getLetterGradeColor = (letterGrade: string): string => {
  switch (letterGrade) {
    case "A":
    case "A+":
      return "bg-green-100 text-green-800";
    case "A-":
    case "B+":
    case "B":
      return "bg-blue-100 text-blue-800";
    case "B-":
    case "C+":
    case "C":
      return "bg-yellow-100 text-yellow-800";
    case "C-":
    case "D+":
    case "D":
      return "bg-orange-100 text-orange-800";
    case "D-":
    case "F":
      return "bg-red-100 text-red-800";
    default:
      return "bg-gray-100 text-gray-800";
  }
};

const getAssignmentsForClass = (classId: number) => {
  return assignments.filter((a) => a.class_id === classId);
};

const getStudentGrades = (studentId: number, classId: number) => {
  const classAssignments = getAssignmentsForClass(classId);
  const studentGrades: Grade[] = [];
```

```
  for (const assignment of classAssignments) {
    const existingGrade = allGrades.find(
      (g) => g.student_id === studentId && g.assignment_id === assignment.id,
    );

    if (existingGrade) {
      studentGrades.push(existingGrade);
    }
  }

  return studentGrades;
};

const getStudentsInClass = (classId: number) => {
  const enrolledStudentIds = enrollments
    .filter((enrollment) => enrollment.class_id === classId)
    .map((enrollment) => enrollment.student_id);

  return students.filter((student) =>
    enrolledStudentIds.includes(student.id),
  );
};

const getStudentsNotInClass = (classId: number) => {
  const enrolledStudentIds = enrollments
    .filter((enrollment) => enrollment.class_id === classId)
    .map((enrollment) => enrollment.student_id);

  return students.filter(
    (student) => !enrolledStudentIds.includes(student.id),
  );
};

const openGradeDialog = () => {
  setSelectedStudent(null);
  setSelectedAssignment(null);
  setScore(0);
  setError(null);
  setIsGradeDialogOpen(true);
};
```

```
const openEditGradesDialog = (studentId: number, classId: number) => {
  const studentGrades = getStudentGrades(studentId, classId);
  setEditingGrades(studentGrades);
  setSelectedStudent(studentId);
  setSelectedClass(classId);
  setError(null);
  setIsEditGradeDialogOpen(true);
};

const openEnrollDialog = () => {
  if (!selectedClass) {
    return;
  }

  setEnrollStudentId(null);
  setEnrollClassId(selectedClass);
  setError(null);
  setIsEnrollDialogOpen(true);
};

const openUnenrollDialog = (studentId: number, classId: number) => {
  const student = students.find((s) => s.id === studentId);
  const classItem = classes.find((c) => c.id === classId);

  if (!student || !classItem) {
    console.error("Student or class not found");
    return;
  }

  setStudentToUnenroll({
    id: studentId,
    name: `${student.first_name} ${student.last_name}`,
  });

  setClassToUnenroll({
    id: classId,
    name: classItem.class_name,
  });

  setError(null);
  setIsUnenrollDialogOpen(true);
};
```

```
const handleGradeSubmit = async () => {
  if (!selectedStudent || !selectedAssignment) return;

  setIsSubmitting(true);
  setError(null);

  try {
    try {
      await getGrade(selectedStudent, selectedAssignment);
      await updateGrade(selectedStudent, selectedAssignment, score);
    } catch (error) {
      await createGrade(selectedStudent, selectedAssignment, score);
    }

    await refreshData();
    await loadAllGrades();
    setIsGradeDialogOpen(false);
  } catch (error) {
    console.error("Error saving grade:", error);
    setError("Failed to save grade. Please try again.");
  } finally {
    setIsSubmitting(false);
  }
};

const handleGradeChange = (index: number, newScore: number) => {
  const updatedGrades = [...editingGrades];
  updatedGrades[index] = { ...updatedGrades[index], score: newScore };
  setEditingGrades(updatedGrades);
};

const handleUpdateGrades = async () => {
  setIsSubmitting(true);
  setError(null);

  try {
    for (const grade of editingGrades) {
      await updateGrade(grade.student_id, grade.assignment_id, grade.score);
    }
    await refreshData();
    await loadAllGrades();
```

```
    setIsEditGradeDialogOpen(false);
  } catch (error) {
    console.error("Error updating grades:", error);
    setError("Failed to update grades. Please try again.");
  } finally {
    setIsSubmitting(false);
  }
};

const handleEnrollStudent = async () => {
  if (!enrollStudentId || !enrollClassId) return;

  setIsSubmitting(true);
  setError(null);

  try {
    await enrollStudent(enrollStudentId, enrollClassId);
    await refreshData();
    await loadEnrollments();
    setIsEnrollDialogOpen(false);
  } catch (error) {
    console.error("Error enrolling student:", error);
    setError("Failed to enroll student. Please try again.");
  } finally {
    setIsSubmitting(false);
  }
};

const handleUnenrollStudent = async () => {
  if (!studentToUnenroll || !classToUnenroll) return;

  setIsSubmitting(true);
  setError(null);

  try {
    await unenrollStudent(studentToUnenroll.id, classToUnenroll.id);
    await refreshData();
    await loadEnrollments();
    setIsUnenrollDialogOpen(false);
  } catch (error) {
    console.error("Error unenrolling student:", error);
    setError("Failed to unenroll student. Please try again.");
```

```
  } finally {
    setIsSubmitting(false);
  }
};

return (
  <div className="space-y-6">
    <div className="flex items-center justify-between">
      <h2 className="text-3xl font-bold tracking-tight">Grades</h2>
      <div className="flex gap-2">
        <Button onClick={openGradeDialog}>
          <PlusCircle className="mr-2 h-4 w-4" />
          Add Grade
        </Button>
        <Button
          variant="outline"
          onClick={openEnrollDialog}
          disabled={!selectedClass}
        >
          Enroll Student
        </Button>
      </div>
    </div>

    <div className="flex flex-col sm:flex-row gap-4">
      <div className="relative flex-1">
        <Search className="absolute left-2.5 top-2.5 h-4 w-4 text-gray-500" />
        <Input
          type="search"
          placeholder="Search grades..."
          className="pl-8"
          value={searchQuery}
          onChange={(e) => setSearchQuery(e.target.value)}
        />
      </div>
      <Select
        value={selectedClass?.toString() || "all"}
        onValueChange={(value) =>
          setSelectedClass(value === "all" ? null : parseInt(value))
        }
      >
        <SelectTrigger className="w-full sm:w-[200px]">
```

```
      <SelectValue placeholder="All Classes" />
    </SelectTrigger>
    <SelectContent>
      <SelectItem value="all">All Classes</SelectItem>
      {classes.map((classItem) => (
        <SelectItem key={classItem.id} value={classItem.id.toString()}>
          {classItem.class_name}
        </SelectItem>
      ))}
    </SelectContent>
  </Select>
</div>

<Card>
  <CardHeader>
    <CardTitle>Student Grades</CardTitle>
    <CardDescription>
      {selectedClass
        ? `Viewing grades for ${getClassName(selectedClass)}`
        : "Viewing grades for all classes"}
    </CardDescription>
  </CardHeader>
  <CardContent>
    {loading ? (
      <div className="space-y-2">
        <Skeleton className="h-8 w-full" />
        <Skeleton className="h-8 w-full" />
        <Skeleton className="h-8 w-full" />
        <Skeleton className="h-8 w-full" />
      </div>
    ) : (
      <Table>
        <TableHeader>
          <TableRow>
            <TableHead>Student</TableHead>
            <TableHead>Class</TableHead>
            <TableHead>Percentage</TableHead>
            <TableHead>Letter Grade</TableHead>
            <TableHead className="text-right">Actions</TableHead>
          </TableRow>
        </TableHeader>
        <TableBody>
```

```
{filteredGrades.length > 0 ? (
 filteredGrades.map((grade) => (
  <TableRow key={`${grade.student_id}-${grade.class_id}`}>
   <TableCell className="font-medium">
    {getStudentName(grade.student_id)}
   </TableCell>
   <TableCell>{getClassName(grade.class_id)}</TableCell>
   <TableCell>{grade.percentage.toFixed(1)}%</TableCell>
   <TableCell>
    <Badge
     className={getLetterGradeColor(grade.letter_grade)}
    >
     {grade.letter_grade}
    </Badge>
   </TableCell>
   <TableCell className="text-right">
    <div className="flex justify-end gap-2">
     <Button
      variant="outline"
      size="sm"
      onClick={() =>
       openEditGradesDialog(
        grade.student_id,
        grade.class_id,
       )
      }
     >
      <Pencil className="mr-1 h-3 w-3" />
      Edit Grades
     </Button>
     <Button
      variant="outline"
      size="sm"
      onClick={() =>
       openUnenrollDialog(
        grade.student_id,
        grade.class_id,
       )
      }
     >
      Unenroll
     </Button>
```

```
                </div>
              </TableCell>
            </TableRow>
          ))
        ) : (
          <TableRow>
            <TableCell
              colSpan={5}
              className="text-center py-4 text-gray-500"
            >
              {searchQuery || selectedClass
                ? "No matching grades found."
                : "No grades found. Add grades to see them here."}
            </TableCell>
          </TableRow>
        )}
      </TableBody>
    </Table>
  )}
 </CardContent>
 <CardFooter className="border-t px-6 py-4">
  <div className="text-xs text-gray-500">
   Showing {filteredGrades.length} of {grades.length} grade records
  </div>
 </CardFooter>
</Card>

<Dialog open={isGradeDialogOpen} onOpenChange={setIsGradeDialogOpen}>
 <DialogContent>
  <DialogHeader>
   <DialogTitle>Add/Update Grade</DialogTitle>
   <DialogDescription>
    Enter or update a grade for a student assignment.
   </DialogDescription>
  </DialogHeader>

  {error && (
   <Alert variant="destructive" className="mt-4">
    <AlertCircle className="h-4 w-4" />
    <AlertTitle>Error</AlertTitle>
    <AlertDescription>{error}</AlertDescription>
   </Alert>
```

```
)}

<div className="grid gap-4 py-4">
  <div className="grid grid-cols-4 items-center gap-4">
    <Label htmlFor="gradeClass" className="text-right">
      Class
    </Label>
    <Select
      value={selectedClass?.toString() || "selectClass"}
      onValueChange={(value) => {
        if (value === "selectClass") return;
        const classId = parseInt(value);
        setSelectedClass(classId);
        setSelectedStudent(null);
        setSelectedAssignment(null);
      }}
    >
      <SelectTrigger className="col-span-3">
        <SelectValue placeholder="Select a class" />
      </SelectTrigger>
      <SelectContent>
        {classes.map((classItem) => (
          <SelectItem
            key={classItem.id}
            value={classItem.id.toString()}
          >
            {classItem.class_name}
          </SelectItem>
        ))}
      </SelectContent>
    </Select>
  </div>

  <div className="grid grid-cols-4 items-center gap-4">
    <Label htmlFor="gradeStudent" className="text-right">
      Student
    </Label>
    <Select
      value={selectedStudent?.toString() || "selectStudent"}
      onValueChange={(value) => {
        if (value === "selectStudent" || value === "no-students")
          return;
```

```
    setSelectedStudent(parseInt(value));
   }}
   disabled={!selectedClass}
  >
   <SelectTrigger className="col-span-3">
    <SelectValue placeholder="Select a student" />
   </SelectTrigger>
   <SelectContent>
    {selectedClass &&
    getStudentsInClass(selectedClass).length > 0 ? (
     getStudentsInClass(selectedClass).map((student) => (
      <SelectItem
       key={student.id}
       value={student.id.toString()}
      >
       {student.first_name} {student.last_name}
      </SelectItem>
     ))
    ) : (
     <SelectItem value="no-students" disabled>
      No students enrolled in this class
     </SelectItem>
    )}
   </SelectContent>
  </Select>
</div>

<div className="grid grid-cols-4 items-center gap-4">
  <Label htmlFor="gradeAssignment" className="text-right">
   Assignment
  </Label>
  <Select
   value={selectedAssignment?.toString() || "selectAssignment"}
   onValueChange={(value) => {
    if (
     value === "selectAssignment" ||
     value === "no-assignments"
    )
     return;
    setSelectedAssignment(parseInt(value));
   }}
   disabled={!selectedClass}
```

```
  >
    <SelectTrigger className="col-span-3">
      <SelectValue placeholder="Select an assignment" />
    </SelectTrigger>
    <SelectContent>
      {selectedClass &&
      getAssignmentsForClass(selectedClass).length > 0 ? (
        getAssignmentsForClass(selectedClass).map((assignment) => (
          <SelectItem
            key={assignment.id}
            value={assignment.id.toString()}
          >
            {assignment.assignment_name}
          </SelectItem>
        ))
      ) : (
        <SelectItem value="no-assignments" disabled>
          No assignments for this class
        </SelectItem>
      )}
    </SelectContent>
  </Select>
</div>

<div className="grid grid-cols-4 items-center gap-4">
  <Label htmlFor="gradeScore" className="text-right">
    Score
  </Label>
  <Input
    id="gradeScore"
    type="number"
    value={score === 0 ? "" : score}
    onChange={(e) => setScore(Number(e.target.value))}
    className="col-span-3"
    min={0}
    step={0.1}
    required
  />
  {selectedAssignment && (
    <div className="col-span-4 text-right text-sm text-gray-500">
      Maximum score: {getAssignmentMaxScore(selectedAssignment)}
    </div>
```

```
          )}
        </div>
      </div>
      <DialogFooter>
        <Button
          variant="outline"
          onClick={() => setIsGradeDialogOpen(false)}
          disabled={isSubmitting}
        >
          Cancel
        </Button>
        <Button
          onClick={handleGradeSubmit}
          disabled={!selectedStudent || !selectedAssignment || isSubmitting}
        >
          {isSubmitting ? "Saving..." : "Save Grade"}
        </Button>
      </DialogFooter>
    </DialogContent>
  </Dialog>

  <Dialog
    open={isEditGradeDialogOpen}
    onOpenChange={setIsEditGradeDialogOpen}
  >
    <DialogContent className="sm:max-w-[600px]">
      <DialogHeader>
        <DialogTitle>Edit Student Grades</DialogTitle>
        <DialogDescription>
          {selectedStudent && selectedClass
            ? `Editing grades for ${getStudentName(selectedStudent)} in ${
getClassName(selectedClass)}`
            : "Edit grades for this student"}
        </DialogDescription>
      </DialogHeader>

      {error && (
        <Alert variant="destructive" className="mt-4">
          <AlertCircle className="h-4 w-4" />
          <AlertTitle>Error</AlertTitle>
          <AlertDescription>{error}</AlertDescription>
        </Alert>
```

```
)}

<div className="space-y-4 py-4 max-h-[300px] overflow-y-auto">
  {editingGrades.length > 0 ? (
    editingGrades.map((grade, index) => {
      const maxScore = getAssignmentMaxScore(grade.assignment_id);
      const percentage =
        maxScore > 0 ? (grade.score / maxScore) * 100 : 0;

      return (
        <div
          key={grade.assignment_id}
          className="grid grid-cols-12 items-center gap-4"
        >
          <div className="col-span-5">
            <Label htmlFor={`grade-${index}`} className="font-medium">
              {getAssignmentName(grade.assignment_id)}
            </Label>
          </div>
          <div className="col-span-3">
            <Input
              id={`grade-${index}`}
              type="number"
              value={grade.score === 0 ? "" : grade.score}
              onChange={(e) =>
                handleGradeChange(index, Number(e.target.value))
              }
              min={0}
              step={0.1}
              required
            />
          </div>
          <div className="col-span-4 text-gray-500 text-sm">
            / {maxScore} ({percentage.toFixed(1)}%)
          </div>
        </div>
      );
    })
  ) : (
    <p className="text-center text-gray-500">
      No grades found for this student in this class.
    </p>
```

```
      )}
    </div>

    <DialogFooter>
      <Button
        variant="outline"
        onClick={() => setIsEditGradeDialogOpen(false)}
        disabled={isSubmitting}
      >
        Cancel
      </Button>
      <Button
        onClick={handleUpdateGrades}
        disabled={editingGrades.length === 0 || isSubmitting}
      >
        {isSubmitting ? "Updating..." : "Update Grades"}
      </Button>
    </DialogFooter>
  </DialogContent>
</Dialog>

<Dialog open={isEnrollDialogOpen} onOpenChange={setIsEnrollDialogOpen}>
  <DialogContent>
    <DialogHeader>
      <DialogTitle>Enroll Student</DialogTitle>
      <DialogDescription>
        Enroll a student in{" "}
        {selectedClass ? getClassName(selectedClass) : "a class"}.
      </DialogDescription>
    </DialogHeader>

    {error && (
      <Alert variant="destructive" className="mt-4">
        <AlertCircle className="h-4 w-4" />
        <AlertTitle>Error</AlertTitle>
        <AlertDescription>{error}</AlertDescription>
      </Alert>
    )}

    <div className="grid gap-4 py-4">
      <div className="grid grid-cols-4 items-center gap-4">
        <Label htmlFor="enrollStudent" className="text-right">
```

```
      Student
    </Label>
    <Select
      value={enrollStudentId?.toString() || "selectEnrollStudent"}
      onValueChange={(value) => {
        if (
          value === "selectEnrollStudent" ||
          value === "all-enrolled"
        )
          return;
        setEnrollStudentId(parseInt(value));
      }}
    >
      <SelectTrigger className="col-span-3">
        <SelectValue placeholder="Select a student" />
      </SelectTrigger>
      <SelectContent>
        {selectedClass &&
        getStudentsNotInClass(selectedClass).length > 0 ? (
          getStudentsNotInClass(selectedClass).map((student) => (
            <SelectItem
              key={student.id}
              value={student.id.toString()}
            >
              {student.first_name} {student.last_name}
            </SelectItem>
          ))
        ) : (
          <SelectItem value="all-enrolled" disabled>
            All students are already enrolled
          </SelectItem>
        )}
      </SelectContent>
    </Select>
  </div>
</div>
<DialogFooter>
  <Button
    variant="outline"
    onClick={() => setIsEnrollDialogOpen(false)}
    disabled={isSubmitting}
  >
```

```
        Cancel
      </Button>
      <Button
        onClick={handleEnrollStudent}
        disabled={!enrollStudentId || !enrollClassId || isSubmitting}
      >
        {isSubmitting ? "Enrolling..." : "Enroll Student"}
      </Button>
    </DialogFooter>
  </DialogContent>
</Dialog>

<Dialog
  open={isUnenrollDialogOpen}
  onOpenChange={setIsUnenrollDialogOpen}
>
  <DialogContent>
    <DialogHeader>
      <DialogTitle>Unenroll Student</DialogTitle>
      <DialogDescription>
        Are you sure you want to unenroll this student from the class?
        This action cannot be undone.
      </DialogDescription>
    </DialogHeader>

    {error && (
      <Alert variant="destructive" className="mt-4">
        <AlertCircle className="h-4 w-4" />
        <AlertTitle>Error</AlertTitle>
        <AlertDescription>{error}</AlertDescription>
      </Alert>
    )}

    <div className="py-4">
      {studentToUnenroll && classToUnenroll && (
        <p>
          You are about to unenroll{" "}
          <strong>{studentToUnenroll.name}</strong> from{" "}
          <strong>{classToUnenroll.name}</strong>.
        </p>
      )}
    </div>
```

```
        <DialogFooter>
          <Button
            variant="outline"
            onClick={() => setIsUnenrollDialogOpen(false)}
            disabled={isSubmitting}
          >
            Cancel
          </Button>
          <Button
            variant="destructive"
            onClick={handleUnenrollStudent}
            disabled={isSubmitting}
          >
            {isSubmitting ? "Unenrolling..." : "Unenroll Student"}
          </Button>
        </DialogFooter>
      </DialogContent>
    </Dialog>
  </div>
 );
}
```

src/components/dashboard-view.tsx: Main dashboard UI

```
import React from "react";
import {
  Card,
  CardContent,
  CardDescription,
  CardHeader,
  CardTitle,
} from "@/components/ui/card";
import { Skeleton } from "@/components/ui/skeleton";
import { UserCircle, BookOpen, FileText, BarChart3 } from "lucide-react";
import type { Student, Class, Assignment, OverallGrade } from "@/api/types";

interface DashboardViewProps {
  students: Student[];
```

```
  classes: Class[];
  assignments: Assignment[];
  overallGrades: OverallGrade[];
  loading: boolean;
}

export default function DashboardView({
  students,
  classes,
  assignments,
  overallGrades,
  loading,
}: DashboardViewProps) {
  const totalStudents = students.length;
  const totalClasses = classes.length;
  const totalAssignments = assignments.length;

  const avgGrade =
    overallGrades.length > 0
      ? (
          overallGrades.reduce((sum, grade) => sum + grade.percentage, 0) /
          overallGrades.length
        ).toFixed(1)
      : "N/A";

  const atRiskStudents = overallGrades
    .filter((grade) => grade.percentage < 70)
    .map((grade) => {
      const student = students.find((s) => s.id === grade.student_id);
      const className = classes.find(
```

```
      (c) => c.id === grade.class_id,
    )?.class_name;
    return {
      studentName: student
        ? `${student.first_name} ${student.last_name}`
        : "Unknown",
      className: className || "Unknown",
      percentage: grade.percentage,
      letterGrade: grade.letter_grade,
    };
  });

const assignmentsByClass = assignments.reduce(
  (acc, assignment) => {
    const classId = assignment.class_id;
    if (!acc[classId]) {
      acc[classId] = [];
    }
    acc[classId].push(assignment);
    return acc;
  },
  {} as Record<number, Assignment[]>,
);

let mostAssignmentsClass = { classId: 0, count: 0, name: "" };
Object.entries(assignmentsByClass).forEach(([classId, assignments]) => {
  if (assignments.length > mostAssignmentsClass.count) {
    const classInfo = classes.find((c) => c.id === parseInt(classId));
    mostAssignmentsClass = {
      classId: parseInt(classId),
```

```
      count: assignments.length,
      name: classInfo?.class_name || "Unknown",
    };
  }
});

const today = new Date();
const nextWeek = new Date(today);
nextWeek.setDate(today.getDate() + 7);

const upcomingAssignments = assignments
  .filter((assignment) => {
    if (!assignment.due_date) return false;
    const dueDate = new Date(assignment.due_date);
    return dueDate >= today && dueDate <= nextWeek;
  })
  .map((assignment) => {
    const className = classes.find(
      (c) => c.id === assignment.class_id,
    )?.class_name;
    return {
      ...assignment,
      className: className || "Unknown",
      formattedDueDate: new Date(assignment.due_date!).toLocaleDateString(),
    };
  })
  .sort(
    (a, b) =>
      new Date(a.due_date!).getTime() - new Date(b.due_date!).getTime(),
  );
```

```
return (
  <div className="space-y-6">
    <h2 className="text-3xl font-bold tracking-tight">Dashboard</h2>

    <div className="grid gap-4 md:grid-cols-2 lg:grid-cols-4">
      <StatsCard
        title="Total Students"
        value={loading ? "Loading..." : totalStudents.toString()}
        description="Enrolled students"
        icon={<UserCircle className="h-4 w-4 text-blue-600" />}
        loading={loading}
      />
      <StatsCard
        title="Total Classes"
        value={loading ? "Loading..." : totalClasses.toString()}
        description="Active classes"
        icon={<BookOpen className="h-4 w-4 text-green-600" />}
        loading={loading}
      />
      <StatsCard
        title="Total Assignments"
        value={loading ? "Loading..." : totalAssignments.toString()}
        description="Created assignments"
        icon={<FileText className="h-4 w-4 text-amber-600" />}
        loading={loading}
      />
      <StatsCard
        title="Average Grade"
        value={loading ? "Loading..." : `${avgGrade}%`}
```

```
    description="Across all classes"
    icon={<BarChart3 className="h-4 w-4 text-purple-600" />}
    loading={loading}
  />
</div>

<div className="grid gap-4 md:grid-cols-2">
  <Card>
    <CardHeader>
      <CardTitle>Students at Risk</CardTitle>
      <CardDescription>Students with grades below 70%</CardDescription>
    </CardHeader>
    <CardContent>
      {loading ? (
        <div className="space-y-2">
          <Skeleton className="h-4 w-full" />
          <Skeleton className="h-4 w-full" />
          <Skeleton className="h-4 w-full" />
        </div>
      ) : atRiskStudents.length > 0 ? (
        <div className="space-y-2">
          {atRiskStudents.map((student, index) => (
            <div
              key={index}
              className="flex justify-between items-center py-2 border-b"
            >
              <div>
                <p className="font-medium">{student.studentName}</p>
                <p className="text-sm text-gray-500">
                  {student.className}
```

```
            </p>
          </div>
          <div className="flex items-center gap-2">
            <span
              className={`px-2 py-1 rounded text-xs font-medium ${
                student.percentage < 60
                  ? "bg-red-100 text-red-800"
                  : "bg-amber-100 text-amber-800"
              }`}
            >
              {student.percentage}% ({student.letterGrade})
            </span>
          </div>
        </div>
      ))}
    </div>
  ) : (
    <p className="text-sm text-gray-500">
      No students currently at risk.
    </p>
  )}
 </CardContent>
</Card>

<Card>
 <CardHeader>
  <CardTitle>Upcoming Assignments</CardTitle>
  <CardDescription>Due in the next 7 days</CardDescription>
 </CardHeader>
 <CardContent>
```

```
{loading ? (
  <div className="space-y-2">
    <Skeleton className="h-4 w-full" />
    <Skeleton className="h-4 w-full" />
    <Skeleton className="h-4 w-full" />
  </div>
) : upcomingAssignments.length > 0 ? (
  <div className="space-y-2">
    {upcomingAssignments.map((assignment) => (
      <div
        key={assignment.id}
        className="flex justify-between items-center py-2 border-b"
      >
        <div>
          <p className="font-medium">
            {assignment.assignment_name}
          </p>
          <p className="text-sm text-gray-500">
            {assignment.className}
          </p>
        </div>
        <div className="flex items-center gap-2">
          <span className="px-2 py-1 bg-blue-100 text-blue-800 rounded text-xs font-medium">
            Due: {assignment.formattedDueDate}
          </span>
        </div>
      </div>
    ))}
  </div>
) : (
```

```
            <p className="text-sm text-gray-500">
              No assignments due in the next 7 days.
            </p>
          )}
        </CardContent>
      </Card>
    </div>
  </div>
 );
}

function StatsCard({
 title,
 value,
 description,
 icon,
 loading,
}: {
 title: string;
 value: string;
 description: string;
 icon: React.ReactNode;
 loading: boolean;
}) {
 return (
  <Card>
   <CardHeader className="flex flex-row items-center justify-between pb-2">
    <CardTitle className="text-sm font-medium">{title}</CardTitle>
    {icon}
   </CardHeader>
```

```
      <CardContent>
        {loading ? (
          <Skeleton className="h-7 w-1/2" />
        ) : (
          <>
            <p className="text-2xl font-bold">{value}</p>
            <p className="text-xs text-gray-500">{description}</p>
          </>
        )}
      </CardContent>
    </Card>
  );
}
```

Backend (Rust)

Core Application

src-tauri/src/main.rs: Main application entry point

```
// Prevents additional console window on Windows in release, DO NOT REMOVE!!
#![cfg_attr(not(debug_assertions), windows_subsystem = "windows")]

fn main() {
    gradify_lib::run()
}
```

src-tauri/src/lib.rs: Application setup and initialization

```
use crate::database::db::Database;
use tauri::{async_runtime, Manager};
use tauri_plugin_fs::FsExt;
use tokio::{fs, sync::Mutex};


mod database {
    pub mod db;
    pub mod models;
}
```

```
mod commands {
    pub mod assignments;
    pub mod classes;
    pub mod grades;
    pub mod overall_grades;
    pub mod student_classes;
    pub mod students;
}

struct AppState {
    db: Database,
}

impl AppState {
    pub fn new(db: Database) -> Self {
        Self { db }
    }
}

#[cfg_attr(mobile, tauri::mobile_entry_point)]
pub fn run() {
    tauri::Builder::default()
        .plugin(tauri_plugin_fs::init())
        .plugin(tauri_plugin_opener::init())
        .setup(|app| {
            let scope = app.fs_scope();

            let exe_path = std::env::current_exe().expect("Can't get exe path");
            let install_dir = exe_path
```

```
            .parent()
            .expect("Can't find parent dir")
            .to_path_buf();
        let db_path = install_dir.join("db.sqlite");

        scope
            .allow_directory(&install_dir, true)
            .expect("Can't allow install dir");

        async_runtime::block_on(async {
            if !db_path.exists() {
                fs::write(&db_path, b"")
                    .await
                    .expect("Can't create db file");
            }

            let db_url = format!("sqlite://{}", db_path.display());
            let db = Database::new(&db_url).await;
            let state = AppState::new(db);
            app.manage(Mutex::new(state));
        });

        Ok(())
    })
    .invoke_handler(tauri::generate_handler![
        commands::students::create_student,
        commands::students::get_student,
        commands::students::get_all_students,
        commands::students::update_student,
        commands::students::delete_student,
```

```
            commands::grades::create_grade,
            commands::grades::get_grade,
            commands::grades::get_all_grades,
            commands::grades::update_grade,
            commands::grades::delete_grade,
            commands::classes::create_class,
            commands::classes::get_class,
            commands::classes::get_all_classes,
            commands::classes::update_class,
            commands::classes::delete_class,
            commands::assignments::create_assignment,
            commands::assignments::get_assignment,
            commands::assignments::get_all_assignments,
            commands::assignments::update_assignment,
            commands::assignments::delete_assignment,
            commands::student_classes::enroll_student,
            commands::student_classes::get_enrollments,
            commands::student_classes::unenroll_student,
            commands::overall_grades::get_overall_grades
        ])
        .run(tauri::generate_context!())
        .expect("error while running tauri application");
}
```

Database Models

src-tauri/src/database/models.rs: Data structures

```
use chrono::NaiveDateTime;
use serde::{Deserialize, Serialize};
use sqlx::FromRow;

#[derive(Debug, Serialize, Deserialize, FromRow)]
pub struct Student {
    #[sqlx(rename = "ID")]
```

```
    pub id: i64,
    #[sqlx(rename = "FIRST_NAME")]
    pub first_name: String,
    #[sqlx(rename = "LAST_NAME")]
    pub last_name: String,
    #[sqlx(rename = "EMAIL")]
    pub email: Option<String>,
}

#[derive(Debug, Serialize, Deserialize, FromRow)]
pub struct Class {
    #[sqlx(rename = "ID")]
    pub id: i64,
    #[sqlx(rename = "CLASS_NAME")]
    pub class_name: String,
    #[sqlx(rename = "DESCRIPTION")]
    pub description: Option<String>,
}

#[derive(Debug, Serialize, Deserialize, FromRow)]
pub struct StudentClass {
    #[sqlx(rename = "STUDENT_ID")]
    pub student_id: i64,
    #[sqlx(rename = "CLASS_ID")]
    pub class_id: i64,
}

#[derive(Debug, Serialize, Deserialize, FromRow)]
pub struct Assignment {
    #[sqlx(rename = "ID")]
    pub id: i64,
    #[sqlx(rename = "CLASS_ID")]
    pub class_id: i64,
    #[sqlx(rename = "ASSIGNMENT_NAME")]
    pub assignment_name: String,
    #[sqlx(rename = "ASSIGNMENT_TYPE")]
    pub assignment_type: String,
    #[sqlx(rename = "MAXIMUM_SCORE")]
    pub maximum_score: f64,
    #[sqlx(rename = "DUE_DATE")]
    pub due_date: Option<NaiveDateTime>,
}
```

```
#[derive(Debug, Serialize, Deserialize, FromRow)]
pub struct Grade {
    #[sqlx(rename = "STUDENT_ID")]
    pub student_id: i64,
    #[sqlx(rename = "ASSIGNMENT_ID")]
    pub assignment_id: i64,
    #[sqlx(rename = "SCORE")]
    pub score: f64,
}

#[derive(Debug, Serialize, Deserialize, FromRow)]
pub struct OverallGrade {
    #[sqlx(rename = "STUDENT_ID")]
    pub student_id: i64,
    #[sqlx(rename = "CLASS_ID")]
    pub class_id: i64,
    #[sqlx(rename = "PERCENTAGE")]
    pub percentage: f64,
    #[sqlx(rename = "LETTER_GRADE")]
    pub letter_grade: String,
}
```

src-tauri/src/database/db.rs: Database connection and migration

```
use sqlx::SqlitePool;


pub struct Database {
    pub pool: SqlitePool,
}


impl Database {
    pub async fn new(url: &str) -> Self {
        let pool = SqlitePool::connect(url).await.expect("Can't connect to db");
        sqlx::migrate!("./migrations")
            .run(&pool)
            .await
            .expect("Can't run migrations");
```

```
        Self { pool }
    }
}
```

Command Handlers

src-tauri/src/commands/students.rs: Student operations

```
use crate::{database::models::Student, AppState};
use tauri::State;
use tokio::sync::Mutex;

#[tauri::command(async, rename_all = "snake_case")]
pub async fn create_student(
    state: State<'_, Mutex<AppState>>,
    first_name: String,
    last_name: String,
    email: Option<String>,
) -> Result<Student, String> {
    let state = state.lock().await;

    let result = sqlx::query(
        "INSERT INTO STUDENTS (FIRST_NAME, LAST_NAME, EMAIL)
         VALUES (?, ?, ?)",
    )
    .bind(first_name.clone())
    .bind(last_name.clone())
    .bind(&email)
    .execute(&state.db.pool)
    .await
    .map_err(|e| e.to_string())?;

    let id = result.last_insert_rowid();

    let student = sqlx::query_as::<_, Student>(
        "SELECT ID, FIRST_NAME, LAST_NAME, EMAIL FROM STUDENTS WHERE ID = ?",
    )
    .bind(id)
    .fetch_one(&state.db.pool)
    .await
    .map_err(|e| e.to_string())?;

    Ok(student)
}
```

```
#[tauri::command(async, rename_all = "snake_case")]
pub async fn get_student(state: State<'_, Mutex<AppState>>, id: i64) -> Result<Student, String> {
    let state = state.lock().await;
    let student = sqlx::query_as::<_, Student>(
        "SELECT ID, FIRST_NAME, LAST_NAME, EMAIL FROM STUDENTS WHERE ID = ?",
    )
    .bind(id)
    .fetch_one(&state.db.pool)
    .await
    .map_err(|e| e.to_string())?;

    Ok(student)
}

#[tauri::command(async, rename_all = "snake_case")]
pub async fn get_all_students(state: State<'_, Mutex<AppState>>) -> Result<Vec<Student>, String> {
    let state = state.lock().await;
    let students =
        sqlx::query_as::<_, Student>("SELECT ID, FIRST_NAME, LAST_NAME, EMAIL FROM
STUDENTS")
            .fetch_all(&state.db.pool)
            .await
            .map_err(|e| e.to_string())?;

    Ok(students)
}

#[tauri::command(async, rename_all = "snake_case")]
pub async fn update_student(
    state: State<'_, Mutex<AppState>>,
    id: i64,
    first_name: String,
    last_name: String,
    email: Option<String>,
) -> Result<Student, String> {
    let state = state.lock().await;

    sqlx::query(
        "UPDATE STUDENTS
         SET FIRST_NAME = ?, LAST_NAME = ?, EMAIL = ?
         WHERE ID = ?",
```

```
    )
    .bind(&first_name)
    .bind(&last_name)
    .bind(&email)
    .bind(id)
    .execute(&state.db.pool)
    .await
    .map_err(|e| e.to_string())?;

    let student = sqlx::query_as::<_, Student>(
        "SELECT ID, FIRST_NAME, LAST_NAME, EMAIL FROM STUDENTS WHERE ID = ?",
    )
    .bind(id)
    .fetch_one(&state.db.pool)
    .await
    .map_err(|e| e.to_string())?;

    Ok(student)
}

#[tauri::command(async, rename_all = "snake_case")]
pub async fn delete_student(state: State<'_, Mutex<AppState>>, id: i64) -> Result<(), String> {
    let state = state.lock().await;
    sqlx::query("DELETE FROM STUDENTS WHERE ID = ?")
        .bind(id)
        .execute(&state.db.pool)
        .await
        .map_err(|e| e.to_string())?;

    Ok(())
}
```

src-tauri/src/commands/classes.rs: Class operations

```
use crate::{database::models::Class, AppState};
use tauri::State;
use tokio::sync::Mutex;

#[tauri::command(async, rename_all = "snake_case")]
pub async fn create_class(
    state: State<'_, Mutex<AppState>>,
    class_name: String,
    description: Option<String>,
) -> Result<Class, String> {
```

```
    let state = state.lock().await;

    let result = sqlx::query(
        "INSERT INTO CLASSES (CLASS_NAME, DESCRIPTION)
         VALUES (?, ?)",
    )
    .bind(&class_name)
    .bind(&description)
    .execute(&state.db.pool)
    .await
    .map_err(|e| e.to_string())?;

    let id = result.last_insert_rowid();

    let class =
        sqlx::query_as::<_, Class>("SELECT ID, CLASS_NAME, DESCRIPTION FROM CLASSES
WHERE ID = ?")
            .bind(id)
            .fetch_one(&state.db.pool)
            .await
            .map_err(|e| e.to_string())?;

    Ok(class)
}

#[tauri::command(async, rename_all = "snake_case")]
pub async fn get_class(state: State<'_, Mutex<AppState>>, id: i64) -> Result<Class, String> {
    let state = state.lock().await;
    let class =
        sqlx::query_as::<_, Class>("SELECT ID, CLASS_NAME, DESCRIPTION FROM CLASSES
WHERE ID = ?")
            .bind(id)
            .fetch_one(&state.db.pool)
            .await
            .map_err(|e| e.to_string())?;
    Ok(class)
}

#[tauri::command(async, rename_all = "snake_case")]
pub async fn get_all_classes(state: State<'_, Mutex<AppState>>) -> Result<Vec<Class>, String> {
    let state = state.lock().await;
```

```
    let classes = sqlx::query_as::<_, Class>("SELECT ID, CLASS_NAME, DESCRIPTION FROM
CLASSES")
        .fetch_all(&state.db.pool)
        .await
        .map_err(|e| e.to_string())?;
    Ok(classes)
}

#[tauri::command(async, rename_all = "snake_case")]
pub async fn update_class(
    state: State<'_, Mutex<AppState>>,
    id: i64,
    class_name: String,
    description: Option<String>,
) -> Result<Class, String> {
    let state = state.lock().await;

    sqlx::query(
        "UPDATE CLASSES
         SET CLASS_NAME = ?, DESCRIPTION = ?
         WHERE ID = ?",
    )
    .bind(&class_name)
    .bind(&description)
    .bind(id)
    .execute(&state.db.pool)
    .await
    .map_err(|e| e.to_string())?;

    let class =
        sqlx::query_as::<_, Class>("SELECT ID, CLASS_NAME, DESCRIPTION FROM CLASSES
WHERE ID = ?")
            .bind(id)
            .fetch_one(&state.db.pool)
            .await
            .map_err(|e| e.to_string())?;

    Ok(class)
}

#[tauri::command(async, rename_all = "snake_case")]
pub async fn delete_class(state: State<'_, Mutex<AppState>>, id: i64) -> Result<(), String> {
```

```
    let state = state.lock().await;
    sqlx::query("DELETE FROM CLASSES WHERE ID = ?")
        .bind(id)
        .execute(&state.db.pool)
        .await
        .map_err(|e| e.to_string())?;
    Ok(())
}
```

src-tauri/src/commands/assignments.rs: Assignment operations

```
use crate::{database::models::Assignment, AppState};
use chrono::NaiveDateTime;
use tauri::State;
use tokio::sync::Mutex;

#[tauri::command(async, rename_all = "snake_case")]
pub async fn create_assignment(
    state: State<'_, Mutex<AppState>>,
    class_id: i64,
    assignment_name: String,
    assignment_type: String,
    maximum_score: f64,
    due_date: Option<NaiveDateTime>,
) -> Result<Assignment, String> {
    let state = state.lock().await;

    let result = sqlx::query(
        "INSERT INTO ASSIGNMENTS (CLASS_ID, ASSIGNMENT_NAME, ASSIGNMENT_TYPE,
MAXIMUM_SCORE, DUE_DATE)
         VALUES (?, ?, ?, ?, ?)"
    )
    .bind(class_id)
    .bind(&assignment_name)
    .bind(&assignment_type)
    .bind(maximum_score)
    .bind(due_date)
    .execute(&state.db.pool)
    .await
    .map_err(|e| e.to_string())?;

    let id = result.last_insert_rowid();

    let assignment = sqlx::query_as::<_, Assignment>(
```

```
        "SELECT ID, CLASS_ID, ASSIGNMENT_NAME, ASSIGNMENT_TYPE,
MAXIMUM_SCORE, DUE_DATE
         FROM ASSIGNMENTS
         WHERE ID = ?",
    )
    .bind(id)
    .fetch_one(&state.db.pool)
    .await
    .map_err(|e| e.to_string())?;

    Ok(assignment)
}

#[tauri::command(async, rename_all = "snake_case")]
pub async fn get_assignment(
    state: State<'_, Mutex<AppState>>,
    id: i64,
) -> Result<Assignment, String> {
    let state = state.lock().await;
    let assignment = sqlx::query_as::<_, Assignment>(
        "SELECT ID, CLASS_ID, ASSIGNMENT_NAME, ASSIGNMENT_TYPE,
MAXIMUM_SCORE, DUE_DATE
         FROM ASSIGNMENTS
         WHERE ID = ?",
    )
    .bind(id)
    .fetch_one(&state.db.pool)
    .await
    .map_err(|e| e.to_string())?;
    Ok(assignment)
}

#[tauri::command(async, rename_all = "snake_case")]
pub async fn get_all_assignments(
    state: State<'_, Mutex<AppState>>,
) -> Result<Vec<Assignment>, String> {
    let state = state.lock().await;
    let assignments = sqlx::query_as::<_, Assignment>(
        "SELECT ID, CLASS_ID, ASSIGNMENT_NAME, ASSIGNMENT_TYPE,
MAXIMUM_SCORE, DUE_DATE
         FROM ASSIGNMENTS",
    )
```

```
    .fetch_all(&state.db.pool)
    .await
    .map_err(|e| e.to_string())?;
    Ok(assignments)
}

#[tauri::command(async, rename_all = "snake_case")]
pub async fn update_assignment(
    state: State<'_, Mutex<AppState>>,
    id: i64,
    class_id: i64,
    assignment_name: String,
    assignment_type: String,
    maximum_score: f64,
    due_date: Option<NaiveDateTime>,
) -> Result<Assignment, String> {
    let state = state.lock().await;

    sqlx::query(
        "UPDATE ASSIGNMENTS
         SET CLASS_ID = ?, ASSIGNMENT_NAME = ?, ASSIGNMENT_TYPE = ?,
MAXIMUM_SCORE = ?, DUE_DATE = ?
         WHERE ID = ?",
    )
    .bind(class_id)
    .bind(&assignment_name)
    .bind(&assignment_type)
    .bind(maximum_score)
    .bind(due_date)
    .bind(id)
    .execute(&state.db.pool)
    .await
    .map_err(|e| e.to_string())?;

    let assignment = sqlx::query_as::<_, Assignment>(
        "SELECT ID, CLASS_ID, ASSIGNMENT_NAME, ASSIGNMENT_TYPE,
MAXIMUM_SCORE, DUE_DATE
         FROM ASSIGNMENTS
         WHERE ID = ?",
    )
    .bind(id)
    .fetch_one(&state.db.pool)
```

```
    .await
    .map_err(|e| e.to_string())?;

    Ok(assignment)
}

#[tauri::command(async, rename_all = "snake_case")]
pub async fn delete_assignment(state: State<'_, Mutex<AppState>>, id: i64) -> Result<(), String> {
    let state = state.lock().await;
    sqlx::query("DELETE FROM ASSIGNMENTS WHERE ID = ?")
        .bind(id)
        .execute(&state.db.pool)
        .await
        .map_err(|e| e.to_string())?;
    Ok(())
}
```

src-tauri/src/commands/grades.rs: Grade operations

```
use crate::{database::models::Grade, AppState};
use tauri::State;
use tokio::sync::Mutex;

#[tauri::command(async, rename_all = "snake_case")]
pub async fn create_grade(
    state: State<'_, Mutex<AppState>>,
    student_id: i64,
    assignment_id: i64,
    score: f64,
) -> Result<Grade, String> {
    let state = state.lock().await;

    sqlx::query(
        "INSERT INTO GRADES (STUDENT_ID, ASSIGNMENT_ID, SCORE)
         VALUES (?, ?, ?)",
    )
    .bind(student_id)
    .bind(assignment_id)
    .bind(score)
    .execute(&state.db.pool)
    .await
    .map_err(|e| e.to_string())?;

    let grade = sqlx::query_as::<_, Grade>(
```

```
        "SELECT STUDENT_ID, ASSIGNMENT_ID, SCORE FROM GRADES
         WHERE STUDENT_ID = ? AND ASSIGNMENT_ID = ?",
    )
    .bind(student_id)
    .bind(assignment_id)
    .fetch_one(&state.db.pool)
    .await
    .map_err(|e| e.to_string())?;

    Ok(grade)
}

#[tauri::command(async, rename_all = "snake_case")]
pub async fn get_grade(
    state: State<'_, Mutex<AppState>>,
    student_id: i64,
    assignment_id: i64,
) -> Result<Grade, String> {
    let state = state.lock().await;
    let grade = sqlx::query_as::<_, Grade>(
        "SELECT STUDENT_ID, ASSIGNMENT_ID, SCORE FROM GRADES
         WHERE STUDENT_ID = ? AND ASSIGNMENT_ID = ?",
    )
    .bind(student_id)
    .bind(assignment_id)
    .fetch_one(&state.db.pool)
    .await
    .map_err(|e| e.to_string())?;
    Ok(grade)
}

#[tauri::command(async, rename_all = "snake_case")]
pub async fn get_all_grades(state: State<'_, Mutex<AppState>>) -> Result<Vec<Grade>, String> {
    let state = state.lock().await;
    let grades = sqlx::query_as::<_, Grade>("SELECT STUDENT_ID, ASSIGNMENT_ID, SCORE
FROM GRADES")
        .fetch_all(&state.db.pool)
        .await
        .map_err(|e| e.to_string())?;
    Ok(grades)
}
```

```
#[tauri::command(async, rename_all = "snake_case")]
pub async fn update_grade(
    state: State<'_, Mutex<AppState>>,
    student_id: i64,
    assignment_id: i64,
    score: f64,
) -> Result<Grade, String> {
    let state = state.lock().await;

    sqlx::query(
        "UPDATE GRADES
         SET SCORE = ?
         WHERE STUDENT_ID = ? AND ASSIGNMENT_ID = ?",
    )
    .bind(score)
    .bind(student_id)
    .bind(assignment_id)
    .execute(&state.db.pool)
    .await
    .map_err(|e| e.to_string())?;

    let grade = sqlx::query_as::<_, Grade>(
        "SELECT STUDENT_ID, ASSIGNMENT_ID, SCORE FROM GRADES
         WHERE STUDENT_ID = ? AND ASSIGNMENT_ID = ?",
    )
    .bind(student_id)
    .bind(assignment_id)
    .fetch_one(&state.db.pool)
    .await
    .map_err(|e| e.to_string())?;

    Ok(grade)
}

#[tauri::command(async, rename_all = "snake_case")]
pub async fn delete_grade(
    state: State<'_, Mutex<AppState>>,
    student_id: i64,
    assignment_id: i64,
) -> Result<(), String> {
    let state = state.lock().await;
    sqlx::query(
```

```
        "DELETE FROM GRADES
         WHERE STUDENT_ID = ? AND ASSIGNMENT_ID = ?",
    )
    .bind(student_id)
    .bind(assignment_id)
    .execute(&state.db.pool)
    .await
    .map_err(|e| e.to_string())?;
    Ok(())
}
```

src-tauri/src/commands/overall_grades.rs: Grade calculation

```
use crate::{database::models::OverallGrade, AppState};

use tauri::State;

use tokio::sync::Mutex;


#[tauri::command(async, rename_all = "snake_case")]

pub async fn get_overall_grades(

    state: State<'_, Mutex<AppState>>,

) -> Result<Vec<OverallGrade>, String> {

    let state = state.lock().await;

    let overall_grades = sqlx::query_as::<_, OverallGrade>(

        "SELECT STUDENT_ID, CLASS_ID, PERCENTAGE, LETTER_GRADE FROM
OVERALL_GRADES",

    )

    .fetch_all(&state.db.pool)

    .await

    .map_err(|e| e.to_string())?;

    Ok(overall_grades)

}
```

Database Schema

src-tauri/migrations/20250309022823_students.sql: Students table

```
CREATE TABLE IF NOT EXISTS STUDENTS (
    ID INTEGER PRIMARY KEY,
    FIRST_NAME TEXT NOT NULL,
```

```
    LAST_NAME TEXT NOT NULL,
    EMAIL TEXT UNIQUE
);
        src-tauri/migrations/20250309022955_classes.sql: Classes table
CREATE TABLE IF NOT EXISTS CLASSES (
    ID INTEGER PRIMARY KEY,
    CLASS_NAME TEXT NOT NULL,
    DESCRIPTION TEXT
);
        src-tauri/migrations/20250309023026_student_classes.sql: Student-class relationships
CREATE TABLE IF NOT EXISTS STUDENT_CLASSES (
    STUDENT_ID INTEGER NOT NULL,
    CLASS_ID INTEGER NOT NULL,
    PRIMARY KEY (STUDENT_ID, CLASS_ID),
    FOREIGN KEY (STUDENT_ID) REFERENCES STUDENTS (ID) ON DELETE CASCADE,
    FOREIGN KEY (CLASS_ID) REFERENCES CLASSES (ID) ON DELETE CASCADE
);
        src-tauri/migrations/20250309023049_assignments.sql: Assignments table
CREATE TABLE IF NOT EXISTS ASSIGNMENTS (
    ID INTEGER PRIMARY KEY,
    CLASS_ID INTEGER NOT NULL,
    ASSIGNMENT_NAME TEXT NOT NULL,
    ASSIGNMENT_TYPE TEXT NOT NULL CHECK (
        ASSIGNMENT_TYPE IN ('Homework', 'Test')
    ),
    MAXIMUM_SCORE REAL NOT NULL,
    DUE_DATE TIMESTAMP,
    FOREIGN KEY (CLASS_ID) REFERENCES CLASSES (ID) ON DELETE CASCADE
);
        src-tauri/migrations/20250309023132_grades.sql: Grades table
CREATE TABLE IF NOT EXISTS GRADES (
    STUDENT_ID INTEGER NOT NULL,
    ASSIGNMENT_ID INTEGER NOT NULL,
    SCORE REAL NOT NULL CHECK (SCORE >= 0),
    PRIMARY KEY (STUDENT_ID, ASSIGNMENT_ID),
    FOREIGN KEY (STUDENT_ID) REFERENCES STUDENTS (ID) ON DELETE CASCADE,
    FOREIGN KEY (ASSIGNMENT_ID) REFERENCES ASSIGNMENTS (ID) ON DELETE
CASCADE
);
        src-tauri/migrations/20250309224349_overall_grades.sql: Grade calculation view

CREATE VIEW OVERALL_GRADES AS

SELECT
```

```
    s.STUDENT_ID,
    a.CLASS_ID,
    AVG(g.SCORE / a.MAXIMUM_SCORE * 100) AS PERCENTAGE,
    CASE
        WHEN AVG(g.SCORE / a.MAXIMUM_SCORE * 100) >= 90 THEN 'A'
        WHEN AVG(g.SCORE / a.MAXIMUM_SCORE * 100) >= 80 THEN 'B'
        WHEN AVG(g.SCORE / a.MAXIMUM_SCORE * 100) >= 70 THEN 'C'
        WHEN AVG(g.SCORE / a.MAXIMUM_SCORE * 100) >= 60 THEN 'D'
        ELSE 'F'
    END AS LETTER_GRADE
FROM
    GRADES g
JOIN
    ASSIGNMENTS a ON g.ASSIGNMENT_ID = a.ID
JOIN
    STUDENT_CLASSES s ON g.STUDENT_ID = s.STUDENT_ID AND a.CLASS_ID =
s.CLASS_ID
GROUP BY
    s.STUDENT_ID, a.CLASS_ID;
```

```
Installer Script
        src-tauri/installer.nsi: NSIS installer configuration
Unicode true
ManifestDPIAware true
ManifestDPIAwareness PerMonitorV2

!if "lzma" == "none"
  SetCompress off
!else
  SetCompressor /SOLID "lzma"
!endif
```

```
!include MUI2.nsh
!include FileFunc.nsh
!include x64.nsh
!include WordFunc.nsh
!include "utils.nsh"
!include "FileAssociation.nsh"
!include "Win\COM.nsh"
!include "Win\Propkey.nsh"
!include "StrFunc.nsh"
${StrCase}
${StrLoc}

!define WEBVIEW2APPGUID "{F3017226-FE2A-4295-8BDF-00C3A9A7E4C5}"

!define MANUFACTURER "gradify"
!define PRODUCTNAME "gradify"
!define VERSION "1.0.1"
!define VERSIONWITHBUILD "1.0.1.0"
!define HOMEPAGE ""
!define INSTALLMODE "perMachine"
!define LICENSE ""
!define INSTALLERICON ""
!define SIDEBARIMAGE ""
!define HEADERIMAGE ""
!define MAINBINARYNAME "gradify"
!define MAINBINARYSRCPATH "/home/joshs/Repos/Gradify/src-tauri/target/x86_64-pc-windows-msvc/release/gradify.exe"
!define BUNDLEID "com.gradify.app"
!define COPYRIGHT ""
```

```
!define OUTFILE "nsis-output.exe"
!define ARCH "x64"
!define PLUGINSPATH "/home/joshs/.cache/tauri/NSIS/Plugins/x86-unicode"
!define ALLOWDOWNGRADES "true"
!define DISPLAYLANGUAGESELECTOR "false"
!define INSTALLWEBVIEW2MODE "downloadBootstrapper"
!define WEBVIEW2INSTALLERARGS "/silent"
!define WEBVIEW2BOOTSTRAPPERPATH ""
!define WEBVIEW2INSTALLERPATH ""
!define MINIMUMWEBVIEW2VERSION ""
!define UNINSTKEY "Software\Microsoft\Windows\CurrentVersion\Uninstall\${PRODUCTNAME}"
!define MANUPRODUCTKEY "Software\${MANUFACTURER}\${PRODUCTNAME}"
!define UNINSTALLERSIGNCOMMAND ""
!define ESTIMATEDSIZE "14065"
!define STARTMENUFOLDER ""

Var PassiveMode
Var UpdateMode
Var NoShortcutMode
Var WixMode
Var OldMainBinaryName

Name "${PRODUCTNAME}"
BrandingText "${COPYRIGHT}"
OutFile "${OUTFILE}"
; CUSTOM: Force install to C:\Gradify for everyone
!define PLACEHOLDER_INSTALL_DIR "C:\Gradify"
InstallDir "C:\Gradify"
InstallDirRegKey HKCU "Software\\Gradify" "Install_Dir"
VIProductVersion "${VERSIONWITHBUILD}"
```

```
VIAddVersionKey "ProductName" "${PRODUCTNAME}"
VIAddVersionKey "FileDescription" "${PRODUCTNAME}"
VIAddVersionKey "LegalCopyright" "${COPYRIGHT}"
VIAddVersionKey "FileVersion" "${VERSION}"
VIAddVersionKey "ProductVersion" "${VERSION}"

!if "${PLUGINSPATH}" != ""
   !addplugindir "${PLUGINSPATH}"
!endif

!if "${UNINSTALLERSIGNCOMMAND}" != ""
  !uninstfinalize '${UNINSTALLERSIGNCOMMAND}'
!endif

; Handle install mode, `perUser`, `perMachine` or `both`
!if "${INSTALLMODE}" == "perMachine"
  RequestExecutionLevel highest
!endif

!if "${INSTALLMODE}" == "currentUser"
  RequestExecutionLevel user
!endif

!if "${INSTALLMODE}" == "both"
  !define MULTIUSER_MUI
  !define MULTIUSER_INSTALLMODE_INSTDIR "${PRODUCTNAME}"
  !define MULTIUSER_INSTALLMODE_COMMANDLINE
  !if "${ARCH}" == "x64"
    !define MULTIUSER_USE_PROGRAMFILES64
  !else if "${ARCH}" == "arm64"
```

```
    !define MULTIUSER_USE_PROGRAMFILES64
  !endif
  !define MULTIUSER_INSTALLMODE_DEFAULT_REGISTRY_KEY "${UNINSTKEY}"
  !define MULTIUSER_INSTALLMODE_DEFAULT_REGISTRY_VALUENAME "CurrentUser"
  !define MULTIUSER_INSTALLMODEPAGE_SHOWUSERNAME
  !define MULTIUSER_INSTALLMODE_FUNCTION RestorePreviousInstallLocation
  !define MULTIUSER_EXECUTIONLEVEL Highest
  !include MultiUser.nsh
!endif

!if "${INSTALLERICON}" != ""
  !define MUI_ICON "${INSTALLERICON}"
!endif

!if "${SIDEBARIMAGE}" != ""
  !define MUI_WELCOMEFINISHPAGE_BITMAP "${SIDEBARIMAGE}"
!endif

!if "${HEADERIMAGE}" != ""
  !define MUI_HEADERIMAGE
  !define MUI_HEADERIMAGE_BITMAP  "${HEADERIMAGE}"
!endif

!define MUI_LANGDLL_REGISTRY_ROOT "HKCU"
!define MUI_LANGDLL_REGISTRY_KEY "${MANUPRODUCTKEY}"
!define MUI_LANGDLL_REGISTRY_VALUENAME "Installer Language"

!define MUI_PAGE_CUSTOMFUNCTION_PRE SkipIfPassive
!insertmacro MUI_PAGE_WELCOME
!undef MUI_PAGE_CUSTOMFUNCTION_PRE
```

```
!if "${LICENSE}" != ""
  !define MUI_PAGE_CUSTOMFUNCTION_PRE SkipIfPassive
  !insertmacro MUI_PAGE_LICENSE "${LICENSE}"
  !undef MUI_PAGE_CUSTOMFUNCTION_PRE
!endif

!if "${INSTALLMODE}" == "both"
  !define MUI_PAGE_CUSTOMFUNCTION_PRE SkipIfPassive
  !insertmacro MULTIUSER_PAGE_INSTALLMODE
  !undef MUI_PAGE_CUSTOMFUNCTION_PRE
!endif

Var ReinstallPageCheck
Page custom PageReinstall PageLeaveReinstall
Function PageReinstall
  StrCpy $0 0
  wix_loop:
    EnumRegKey $1 HKLM "SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall" $0
    StrCmp $1 "" wix_loop_done
    IntOp $0 $0 + 1
    ReadRegStr $R0 HKLM "SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\$1"
"DisplayName"
    ReadRegStr $R1 HKLM "SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\$1"
"Publisher"
    StrCmp "$R0$R1" "${PRODUCTNAME}${MANUFACTURER}" 0 wix_loop
    ReadRegStr $R0 HKLM "SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\$1"
"UninstallString"
    ${StrCase} $R1 $R0 "L"
    ${StrLoc} $R0 $R1 "msiexec" ">"
    StrCmp $R0 0 0 wix_loop_done
```

```
  StrCpy $WixMode 1
  StrCpy $R6 "SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\$1"
  Goto compare_version
wix_loop_done:

ReadRegStr $R0 SHCTX "${UNINSTKEY}" ""
ReadRegStr $R1 SHCTX "${UNINSTKEY}" "UninstallString"
${IfThen} "$R0$R1" == "" ${|} Abort ${|}

compare_version:
StrCpy $R4 "$(older)"
${If} $WixMode = 1
  ReadRegStr $R0 HKLM "$R6" "DisplayVersion"
${Else}
  ReadRegStr $R0 SHCTX "${UNINSTKEY}" "DisplayVersion"
${EndIf}
${IfThen} $R0 == "" ${|} StrCpy $R4 "$(unknown)" ${|}

nsis_tauri_utils::SemverCompare "${VERSION}" $R0
Pop $R0
${If} $R0 = 0
  StrCpy $R1 "$(alreadyInstalledLong)"
  StrCpy $R2 "$(addOrReinstall)"
  StrCpy $R3 "$(uninstallApp)"
  !insertmacro MUI_HEADER_TEXT "$(alreadyInstalled)" "$(chooseMaintenanceOption)"
${ElseIf} $R0 = 1
  StrCpy $R1 "$(olderOrUnknownVersionInstalled)"
  StrCpy $R2 "$(uninstallBeforeInstalling)"
  StrCpy $R3 "$(dontUninstall)"
  !insertmacro MUI_HEADER_TEXT "$(alreadyInstalled)" "$(choowHowToInstall)"
```

```
${ElseIf} $R0 = -1
  StrCpy $R1 "$(newerVersionInstalled)"
  StrCpy $R2 "$(uninstallBeforeInstalling)"
  !if "${ALLOWDOWNGRADES}" == "true"
    StrCpy $R3 "$(dontUninstall)"
  !else
    StrCpy $R3 "$(dontUninstallDowngrade)"
  !endif
  !insertmacro MUI_HEADER_TEXT "$(alreadyInstalled)" "$(choowHowToInstall)"
${Else}
  Abort
${EndIf}

${If} $PassiveMode = 1
  Call PageLeaveReinstall
${Else}
  nsDialogs::Create 1018
  Pop $R4
  ${IfThen} $(^RTL) = 1 ${|} nsDialogs::SetRTL $(^RTL) ${|}

  ${NSD_CreateLabel} 0 0 100% 24u $R1
  Pop $R1

  ${NSD_CreateRadioButton} 30u 50u -30u 8u $R2
  Pop $R2
  ${NSD_OnClick} $R2 PageReinstallUpdateSelection

  ${NSD_CreateRadioButton} 30u 70u -30u 8u $R3
  Pop $R3
  ${NSD_OnClick} $R3 PageReinstallUpdateSelection
```

```
    ${If} $ReinstallPageCheck <> 2
      SendMessage $R2 ${BM_SETCHECK} ${BST_CHECKED} 0
    ${Else}
      SendMessage $R3 ${BM_SETCHECK} ${BST_CHECKED} 0
    ${EndIf}

    ${NSD_SetFocus} $R2
    nsDialogs::Show
  ${EndIf}
FunctionEnd
Function PageReinstallUpdateSelection
  ${NSD_GetState} $R2 $R1
  ${If} $R1 == ${BST_CHECKED}
    StrCpy $ReinstallPageCheck 1
  ${Else}
    StrCpy $ReinstallPageCheck 2
  ${EndIf}
FunctionEnd
Function PageLeaveReinstall
  ${NSD_GetState} $R2 $R1

  ${If} $WixMode = 1
    Goto reinst_uninstall
  ${EndIf}

  ${If} $UpdateMode = 1
    Goto reinst_done
  ${EndIf}
```

```
${If} $R0 = 0
  ${If} $R1 = 1
    Goto reinst_done
  ${Else}
    Goto reinst_uninstall
  ${EndIf}
${ElseIf} $R0 = 1
  ${If} $R1 = 1
    Goto reinst_uninstall
  ${Else}
    Goto reinst_done
  ${EndIf}
${ElseIf} $R0 = -1
  ${If} $R1 = 1
    Goto reinst_uninstall
  ${Else}
    Goto reinst_done
  ${EndIf}
${EndIf}

reinst_uninstall:
  HideWindow
  ClearErrors
  ${If} $WixMode = 1
    ReadRegStr $R1 HKLM "$R6" "UninstallString"
    ExecWait '$R1' $0
  ${Else}
    ReadRegStr $4 SHCTX "${MANUPRODUCTKEY}" ""
    ReadRegStr $R1 SHCTX "${UNINSTKEY}" "UninstallString"
    ${IfThen} $UpdateMode = 1 ${|} StrCpy $R1 "$R1 /UPDATE" ${|}
```

```
    ${IfThen} $PassiveMode = 1 ${|} StrCpy $R1 "$R1 /P" ${|}
    StrCpy $R1 "$R1 _?=$4"
    ExecWait '$R1' $0
   ${EndIf}

   BringToFront
   ${IfThen} ${Errors} ${|} StrCpy $0 2 ${|}
   ${If} $0 <> 0
   ${OrIf} ${FileExists} "$INSTDIR\${MAINBINARYNAME}.exe"
    ${If} $WixMode = 1
    ${AndIf} $0 = 1
     Abort
    ${EndIf}
    ${If} $0 = 1
     Abort
    ${EndIf}
    MessageBox MB_ICONEXCLAMATION "$(unableToUninstall)"
    Abort
   ${EndIf}
  reinst_done:
FunctionEnd

!define MUI_PAGE_CUSTOMFUNCTION_PRE SkipIfPassive
!undef MUI_PAGE_CUSTOMFUNCTION_PRE

Var AppStartMenuFolder
!if "${STARTMENUFOLDER}" != ""
  !define MUI_PAGE_CUSTOMFUNCTION_PRE SkipIfPassive
  !define MUI_STARTMENUPAGE_DEFAULTFOLDER "${STARTMENUFOLDER}"
!undef MUI_PAGE_CUSTOMFUNCTION_PRE
```

```
!else
  !define MUI_PAGE_CUSTOMFUNCTION_PRE Skip
  !undef MUI_PAGE_CUSTOMFUNCTION_PRE
!endif
!insertmacro MUI_PAGE_STARTMENU Application $AppStartMenuFolder

!insertmacro MUI_PAGE_INSTFILES

!define MUI_FINISHPAGE_NOAUTOCLOSE
!define MUI_FINISHPAGE_SHOWREADME
!define MUI_FINISHPAGE_SHOWREADME_TEXT "$(createDesktop)"
!define MUI_FINISHPAGE_SHOWREADME_FUNCTION CreateOrUpdateDesktopShortcut
!define MUI_FINISHPAGE_RUN
!define MUI_FINISHPAGE_RUN_FUNCTION RunMainBinary
!define MUI_PAGE_CUSTOMFUNCTION_PRE SkipIfPassive
!undef MUI_PAGE_CUSTOMFUNCTION_PRE

!insertmacro MUI_PAGE_FINISH

Function RunMainBinary
  nsis_tauri_utils::RunAsUser "$INSTDIR\${MAINBINARYNAME}.exe" ""
FunctionEnd

Var DeleteAppDataCheckbox
Var DeleteAppDataCheckboxState
!define /ifndef WS_EX_LAYOUTRTL 0x00400000
!define MUI_PAGE_CUSTOMFUNCTION_SHOW un.ConfirmShow
Function un.ConfirmShow
  FindWindow $1 "#32770" "" $HWNDPARENT
  System::Call "user32::GetDpiForWindow(p r1) i .r2"
```

```
  ${If} $(^RTL) = 1
    StrCpy $3 "${__NSD_CheckBox_EXSTYLE} | ${WS_EX_LAYOUTRTL}"
    IntOp $4 50 * $2
  ${Else}
    StrCpy $3 "${__NSD_CheckBox_EXSTYLE}"
    IntOp $4 0 * $2
  ${EndIf}
  IntOp $5 100 * $2
  IntOp $6 400 * $2
  IntOp $7 25 * $2
  IntOp $4 $4 / 96
  IntOp $5 $5 / 96
  IntOp $6 $6 / 96
  IntOp $7 $7 / 96
  System::Call 'user32::CreateWindowEx(i r3, w "${__NSD_CheckBox_CLASS}", w "$
(deleteAppData)", i ${__NSD_CheckBox_STYLE}, i r4, i r5, i r6, i r7, p r1, i0, i0, i0) i .s'
  Pop $DeleteAppDataCheckbox
  SendMessage $HWNDPARENT ${WM_GETFONT} 0 0 $1
  SendMessage $DeleteAppDataCheckbox ${WM_SETFONT} $1 1
FunctionEnd
!define MUI_PAGE_CUSTOMFUNCTION_LEAVE un.ConfirmLeave
Function un.ConfirmLeave
  SendMessage $DeleteAppDataCheckbox ${BM_GETCHECK} 0 0 $DeleteAppDataCheckboxState
FunctionEnd
!define MUI_PAGE_CUSTOMFUNCTION_PRE un.SkipIfPassive
!insertmacro MUI_UNPAGE_CONFIRM
!insertmacro MUI_UNPAGE_INSTFILES

!insertmacro MUI_LANGUAGE "English"
!insertmacro MUI_RESERVEFILE_LANGDLL
```

```
  !include "/home/joshs/Repos/Gradify/src-tauri/target/x86_64-pc-windows-msvc/release/nsis/x64/
English.nsh"

Function .onInit
  ${GetOptions} $CMDLINE "/P" $PassiveMode
  ${IfNot} ${Errors}
    StrCpy $PassiveMode 1
  ${EndIf}

  ${GetOptions} $CMDLINE "/NS" $NoShortcutMode
  ${IfNot} ${Errors}
    StrCpy $NoShortcutMode 1
  ${EndIf}

  ${GetOptions} $CMDLINE "/UPDATE" $UpdateMode
  ${IfNot} ${Errors}
    StrCpy $UpdateMode 1
  ${EndIf}

  !if "${DISPLAYLANGUAGESELECTOR}" == "true"
    !insertmacro MUI_LANGDLL_DISPLAY
  !endif

  !insertmacro SetContext

  ${If} $INSTDIR == "${PLACEHOLDER_INSTALL_DIR}"
    Call RestorePreviousInstallLocation
  ${EndIf}

  !if "${INSTALLMODE}" == "both"
```

```
    !insertmacro MULTIUSER_INIT
  !endif
FunctionEnd

Section EarlyChecks
  !if "${ALLOWDOWNGRADES}" == "false"
  ${If} ${Silent}
    ${If} $R0 = -1
      System::Call 'kernel32::AttachConsole(i -1)i.r0'
      ${If} $0 <> 0
        System::Call 'kernel32::GetStdHandle(i -11)i.r0'
        System::call 'kernel32::SetConsoleTextAttribute(i r0, i 0x0004)'
        FileWrite $0 "$(silentDowngrades)"
      ${EndIf}
      Abort
    ${EndIf}
  ${EndIf}
  !endif
SectionEnd

Section WebView2
  ${If} ${RunningX64}
    ReadRegStr $4 HKLM "SOFTWARE\WOW6432Node\Microsoft\EdgeUpdate\Clients\${WEBVIEW2APPGUID}" "pv"
  ${Else}
    ReadRegStr $4 HKLM "SOFTWARE\Microsoft\EdgeUpdate\Clients\${WEBVIEW2APPGUID}" "pv"
  ${EndIf}
  ${If} $4 == ""
    ReadRegStr $4 HKCU "SOFTWARE\Microsoft\EdgeUpdate\Clients\${WEBVIEW2APPGUID}" "pv"
```

```
  ${EndIf}


  ${If} $4 == ""
    ${If} $UpdateMode <> 1
      !if "${INSTALLWEBVIEW2MODE}" == "downloadBootstrapper"
        Delete "$TEMP\MicrosoftEdgeWebview2Setup.exe"
        DetailPrint "$(webview2Downloading)"
        NSISdl::download "https://go.microsoft.com/fwlink/p/?LinkId=2124703"
"$TEMP\MicrosoftEdgeWebview2Setup.exe"
        Pop $0
        ${If} $0 == "success"
          DetailPrint "$(webview2DownloadSuccess)"
        ${Else}
          DetailPrint "$(webview2DownloadError)"
          Abort "$(webview2AbortError)"
        ${EndIf}
        StrCpy $6 "$TEMP\MicrosoftEdgeWebview2Setup.exe"
        Goto install_webview2
      !endif

      !if "${INSTALLWEBVIEW2MODE}" == "embedBootstrapper"
        Delete "$TEMP\MicrosoftEdgeWebview2Setup.exe"
        File "/oname=$TEMP\MicrosoftEdgeWebview2Setup.exe" "$
{WEBVIEW2BOOTSTRAPPERPATH}"
        DetailPrint "$(installingWebview2)"
        StrCpy $6 "$TEMP\MicrosoftEdgeWebview2Setup.exe"
        Goto install_webview2
      !endif

      !if "${INSTALLWEBVIEW2MODE}" == "offlineInstaller"
```

```
      Delete "$TEMP\MicrosoftEdgeWebView2RuntimeInstaller.exe"
      File "/oname=$TEMP\MicrosoftEdgeWebView2RuntimeInstaller.exe" "$
{WEBVIEW2INSTALLERPATH}"
      DetailPrint "$(installingWebview2)"
      StrCpy $6 "$TEMP\MicrosoftEdgeWebview2RuntimeInstaller.exe"
      Goto install_webview2
    !endif

    Goto webview2_done
    install_webview2:
      DetailPrint "$(installingWebview2)"
      ExecWait "$6 ${WEBVIEW2INSTALLERARGS} /install" $1
      ${If} $1 = 0
        DetailPrint "$(webview2InstallSuccess)"
      ${Else}
        DetailPrint "$(webview2InstallError)"
        Abort "$(webview2AbortError)"
      ${EndIf}
    webview2_done:
  ${EndIf}
 ${Else}
  !if "${MINIMUMWEBVIEW2VERSION}" != ""
    ${VersionCompare} "${MINIMUMWEBVIEW2VERSION}" "$4" $R0
    ${If} $R0 = 1
      update_webview:
        DetailPrint "$(installingWebview2)"
        ${If} ${RunningX64}
          ReadRegStr $R1 HKLM "SOFTWARE\WOW6432Node\Microsoft\EdgeUpdate" "path"
        ${Else}
          ReadRegStr $R1 HKLM "SOFTWARE\Microsoft\EdgeUpdate" "path"
```

```
        ${EndIf}
        ${If} $R1 == ""
          ReadRegStr $R1 HKCU "SOFTWARE\Microsoft\EdgeUpdate" "path"
        ${EndIf}
        ${If} $R1 != ""
          ExecWait "$R1 /install appguid=${WEBVIEW2APPGUID}&needsadmin=true" $1
          ${If} $1 = 0
            DetailPrint "$(webview2InstallSuccess)"
          ${Else}
            MessageBox MB_ICONEXCLAMATION|MB_ABORTRETRYIGNORE "$
(webview2InstallError)" IDIGNORE ignore IDRETRY update_webview
            Quit
            ignore:
          ${EndIf}
        ${EndIf}
      ${EndIf}
    !endif
  ${EndIf}
SectionEnd

Section Install
  SetOutPath $INSTDIR

  !ifmacrodef NSIS_HOOK_PREINSTALL
    !insertmacro NSIS_HOOK_PREINSTALL
  !endif

  !insertmacro CheckIfAppIsRunning

  ; Copy main executable
```

```
  File "${MAINBINARYSRCPATH}"

  ; (No DB file needed; your app creates it on first run)

  WriteUninstaller "$INSTDIR\uninstall.exe"
  WriteRegStr SHCTX "${MANUPRODUCTKEY}" "" $INSTDIR

  !if "${INSTALLMODE}" == "both"
    WriteRegStr SHCTX "${UNINSTKEY}" $MultiUser.InstallMode 1
  !endif

  ReadRegStr $OldMainBinaryName SHCTX "${UNINSTKEY}" "MainBinaryName"
  ${If} $OldMainBinaryName != ""
  ${AndIf} $OldMainBinaryName != "${MAINBINARYNAME}.exe"
    Delete "$INSTDIR\$OldMainBinaryName"
  ${EndIf}
  WriteRegStr SHCTX "${UNINSTKEY}" "MainBinaryName" "${MAINBINARYNAME}.exe"

  WriteRegStr SHCTX "${UNINSTKEY}" "DisplayName" "${PRODUCTNAME}"
  WriteRegStr SHCTX "${UNINSTKEY}" "DisplayIcon" "$\"$INSTDIR\$
{MAINBINARYNAME}.exe$\""
  WriteRegStr SHCTX "${UNINSTKEY}" "DisplayVersion" "${VERSION}"
  WriteRegStr SHCTX "${UNINSTKEY}" "Publisher" "${MANUFACTURER}"
  WriteRegStr SHCTX "${UNINSTKEY}" "InstallLocation" "$\"$INSTDIR$\""
  WriteRegStr SHCTX "${UNINSTKEY}" "UninstallString" "$\"$INSTDIR\uninstall.exe$\""
  WriteRegDWORD SHCTX "${UNINSTKEY}" "NoModify" "1"
  WriteRegDWORD SHCTX "${UNINSTKEY}" "NoRepair" "1"

  ${GetSize} "$INSTDIR" "/M=uninstall.exe /S=0K /G=0" $0 $1 $2
  IntOp $0 $0 + ${ESTIMATEDSIZE}
```

```
  IntFmt $0 "0x%08X" $0
  WriteRegDWORD SHCTX "${UNINSTKEY}" "EstimatedSize" "$0"

  !if "${HOMEPAGE}" != ""
    WriteRegStr SHCTX "${UNINSTKEY}" "URLInfoAbout" "${HOMEPAGE}"
    WriteRegStr SHCTX "${UNINSTKEY}" "URLUpdateInfo" "${HOMEPAGE}"
    WriteRegStr SHCTX "${UNINSTKEY}" "HelpLink" "${HOMEPAGE}"
  !endif

  !insertmacro MUI_STARTMENU_WRITE_BEGIN Application
    Call CreateOrUpdateStartMenuShortcut
  !insertmacro MUI_STARTMENU_WRITE_END

  ${If} $PassiveMode = 1
  ${OrIf} ${Silent}
    Call CreateOrUpdateDesktopShortcut
  ${EndIf}

  !ifmacrodef NSIS_HOOK_POSTINSTALL
    !insertmacro NSIS_HOOK_POSTINSTALL
  !endif

  ${If} $PassiveMode = 1
    SetAutoClose true
  ${EndIf}
SectionEnd

Function .onInstSuccess
  ${If} $PassiveMode = 1
  ${OrIf} ${Silent}
```

```
    ${GetOptions} $CMDLINE "/R" $R0
    ${IfNot} ${Errors}
      ${GetOptions} $CMDLINE "/ARGS" $R0
      nsis_tauri_utils::RunAsUser "$INSTDIR\${MAINBINARYNAME}.exe" "$R0"
    ${EndIf}
  ${EndIf}
FunctionEnd

Function un.onInit
  !insertmacro SetContext

  !if "${INSTALLMODE}" == "both"
    !insertmacro MULTIUSER_UNINIT
  !endif

  !insertmacro MUI_UNGETLANGUAGE

  ${GetOptions} $CMDLINE "/P" $PassiveMode
  ${IfNot} ${Errors}
    StrCpy $PassiveMode 1
  ${EndIf}

  ${GetOptions} $CMDLINE "/UPDATE" $UpdateMode
  ${IfNot} ${Errors}
    StrCpy $UpdateMode 1
  ${EndIf}
FunctionEnd

Section Uninstall
```

```
  !ifmacrodef NSIS_HOOK_PREUNINSTALL
    !insertmacro NSIS_HOOK_PREUNINSTALL
  !endif

  !insertmacro CheckIfAppIsRunning

  Delete "$INSTDIR\${MAINBINARYNAME}.exe"

  Delete "$INSTDIR\uninstall.exe"
  RMDir "$INSTDIR"

  ${If} $UpdateMode <> 1
    !insertmacro DeleteAppUserModelId

    !insertmacro MUI_STARTMENU_GETFOLDER Application $AppStartMenuFolder
    !insertmacro IsShortcutTarget "$SMPROGRAMS\$AppStartMenuFolder\$
{PRODUCTNAME}.lnk" "$INSTDIR\${MAINBINARYNAME}.exe"
    Pop $0
    ${If} $0 = 1
      !insertmacro UnpinShortcut "$SMPROGRAMS\$AppStartMenuFolder\${PRODUCTNAME}.lnk"
      Delete "$SMPROGRAMS\$AppStartMenuFolder\${PRODUCTNAME}.lnk"
      RMDir "$SMPROGRAMS\$AppStartMenuFolder"
    ${EndIf}

    !insertmacro IsShortcutTarget "$SMPROGRAMS\${PRODUCTNAME}.lnk" "$INSTDIR\$
{MAINBINARYNAME}.exe"
    Pop $0
    ${If} $0 = 1
      !insertmacro UnpinShortcut "$SMPROGRAMS\${PRODUCTNAME}.lnk"
      Delete "$SMPROGRAMS\${PRODUCTNAME}.lnk"
```

```
  ${EndIf}
  ; CUSTOM: remove existing references to $DESKTOP
  ; Now remove from ALL USERS DESKTOP ($COMMONDESKTOP)
  !insertmacro IsShortcutTarget "$COMMONDESKTOP\${PRODUCTNAME}.lnk" "$INSTDIR\${MAINBINARYNAME}.exe"
  Pop $0
  ${If} $0 = 1
    !insertmacro UnpinShortcut "$COMMONDESKTOP\${PRODUCTNAME}.lnk"
    Delete "$COMMONDESKTOP\${PRODUCTNAME}.lnk"
  ${EndIf}
${EndIf}

!if "${INSTALLMODE}" == "both"
  DeleteRegKey SHCTX "${UNINSTKEY}"
!else if "${INSTALLMODE}" == "perMachine"
  DeleteRegKey HKLM "${UNINSTKEY}"
!else
  DeleteRegKey HKCU "${UNINSTKEY}"
!endif

DeleteRegValue HKCU "${MANUPRODUCTKEY}" "Installer Language"

${If} $DeleteAppDataCheckboxState = 1
${AndIf} $UpdateMode <> 1
  SetShellVarContext current
  RmDir /r "$APPDATA\${BUNDLEID}"
  RmDir /r "$LOCALAPPDATA\${BUNDLEID}"
${EndIf}

!ifmacrodef NSIS_HOOK_POSTUNINSTALL
```

```
    !insertmacro NSIS_HOOK_POSTUNINSTALL
  !endif

  ${If} $PassiveMode = 1
  ${OrIf} $UpdateMode = 1
    SetAutoClose true
  ${EndIf}
SectionEnd

; (We keep RestorePreviousInstallLocation, but we won't call it anymore)
Function RestorePreviousInstallLocation
  ReadRegStr $4 SHCTX "${MANUPRODUCTKEY}" ""
  StrCmp $4 "" +2 0
    StrCpy $INSTDIR $4
FunctionEnd

Function Skip
  Abort
FunctionEnd

Function SkipIfPassive
  ${IfThen} $PassiveMode = 1  ${|} Abort ${|}
FunctionEnd
Function un.SkipIfPassive
  ${IfThen} $PassiveMode = 1  ${|} Abort ${|}
FunctionEnd
; CUSTOM: Desktop Shortcut for ALL USERS
; Replace $DESKTOP with $COMMONDESKTOP below:
Function CreateOrUpdateDesktopShortcut
```

```
  !insertmacro IsShortcutTarget "$COMMONDESKTOP\${PRODUCTNAME}.lnk" "$INSTDIR\
$OldMainBinaryName"
  Pop $0
  ${If} $0 = 1
    !insertmacro SetShortcutTarget "$COMMONDESKTOP\${PRODUCTNAME}.lnk" "$INSTDIR\$
{MAINBINARYNAME}.exe"
    Return
  ${EndIf}

  ${If} $WixMode = 0
    ${If} $UpdateMode = 1
    ${OrIf} $NoShortcutMode = 1
      Return
    ${EndIf}
  ${EndIf}

  CreateShortcut "$COMMONDESKTOP\${PRODUCTNAME}.lnk" "$INSTDIR\$
{MAINBINARYNAME}.exe"
  !insertmacro SetLnkAppUserModelId "$COMMONDESKTOP\${PRODUCTNAME}.lnk"
FunctionEnd

Function CreateOrUpdateStartMenuShortcut
  StrCpy $R0 0

  !insertmacro IsShortcutTarget "$SMPROGRAMS\$AppStartMenuFolder\${PRODUCTNAME}.lnk"
"$INSTDIR\$OldMainBinaryName"
  Pop $0
  ${If} $0 = 1
    !insertmacro SetShortcutTarget "$SMPROGRAMS\$AppStartMenuFolder\$
{PRODUCTNAME}.lnk" "$INSTDIR\${MAINBINARYNAME}.exe"
    StrCpy $R0 1
```

```
  ${EndIf}

  !insertmacro IsShortcutTarget "$SMPROGRAMS\${PRODUCTNAME}.lnk" "$INSTDIR\
$OldMainBinaryName"
  Pop $0
  ${If} $0 = 1
    !insertmacro SetShortcutTarget "$SMPROGRAMS\${PRODUCTNAME}.lnk" "$INSTDIR\$
{MAINBINARYNAME}.exe"
    StrCpy $R0 1
  ${EndIf}

  ${If} $R0 = 1
    Return
  ${EndIf}

  ${If} $WixMode = 0
    ${If} $UpdateMode = 1
    ${OrIf} $NoShortcutMode = 1
      Return
    ${EndIf}
  ${EndIf}

  !if "${STARTMENUFOLDER}" != ""
    CreateDirectory "$SMPROGRAMS\$AppStartMenuFolder"
    CreateShortcut "$SMPROGRAMS\$AppStartMenuFolder\${PRODUCTNAME}.lnk" "$INSTDIR\
${MAINBINARYNAME}.exe"
    !insertmacro SetLnkAppUserModelId "$SMPROGRAMS\$AppStartMenuFolder\$
{PRODUCTNAME}.lnk"
  !else
    CreateShortcut "$SMPROGRAMS\${PRODUCTNAME}.lnk" "$INSTDIR\$
{MAINBINARYNAME}.exe"
```

```
    !insertmacro SetLnkAppUserModelId "$SMPROGRAMS\${PRODUCTNAME}.lnk"
  !endif
FunctionEnd
```