Student Management System Phase 2 – Report

1. Description of your proposed platform

Give an overview of how your application works

Our platform is a student management system built for the CSE department at Qatar University. It supports three types of users: students, instructors, and administrators. Each user has access to specific features such as course registration, grade submission, and course validation.

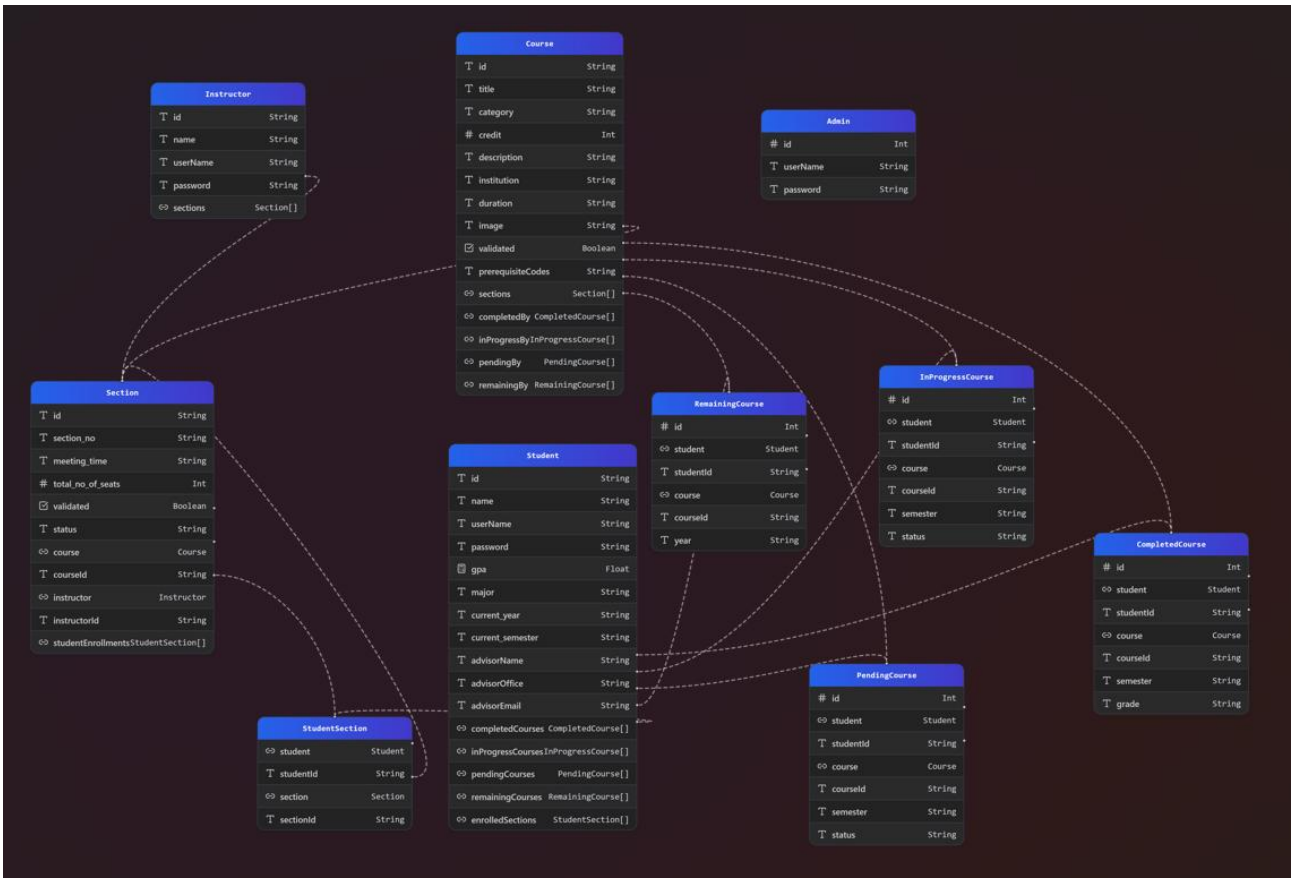
In Phase 1, we created the full interface using HTML, CSS, and JavaScript. Data was stored in JSON files. Students could log in, search for courses, register if they met the conditions, and view their academic progress. Instructors could access their classes and submit grades. Administrators were able to validate or remove courses and create new classes.

In Phase 2, we moved the data to a real relational database using Prisma ORM and SQLite. We built a repository using Prisma Client to handle data operations, with filtering and aggregation done directly in the database through queries.

We also added a new statistics page for administrator, built with Next.js and React. This page shows various academic statistics such as failure rates, GPA trends, course popularity, and instructor activity. The data is served through an API using Prisma queries.

Authentication is handled using NextAuth, with support for both GitHub and username/password login.

2. Data Model


Course
id String
title String
category String
credit Int
description String
institution String
duration String
image String
validated Boolean
prerequisiteCodes String
sections Section[]
completedBy CompletedCourse[]
inProgressByInProgressCourse[]
pendingBy PendingCourse[]
remainingBy RemainingCourse[]
Instructor
id String
name String
userName String
password String
sections Section[]
Admin
id Int
userName String
password String
Section
id String
section_no String
meeting_time String
total_no_of_seats Int
validated Boolean
status String
course Course
courseId String
instructor Instructor
instructorId String
studentEnrollments StudentSection[]
RemainingCourse
id Int
student Student
studentId String
course Course
courseId String
year String
InProgressCourse
id Int
student Student
studentId String
course Course
courseId String
semester String
status String
CompletedCourse
id Int
student Student
studentId String
course Course
courseId String
semester String
grade String
Student
id String
name String
userName String
password String
gpa Float
major String
current_year String
current_semester String
advisorName String
advisorOffice String
advisorEmail String
completedCourses CompletedCourse[]
inProgressCourses InProgressCourse[]
pendingCourses PendingCourse[]
remainingCourses RemainingCourse[]
enrolledSections StudentSection[]
StudentSection
student Student
studentId String
section Section
sectionId String
PendingCourse
id Int
student Student
studentId String
course Course
courseId String
semester String
status String

```prisma
model Course {
  id           String  @id // Using course
  title        String
  category     String
  credit       Int
  description String
  institution String
  duration     String
  image        String
  validated    Boolean @default(false)

  // Simple prerequisite implementation
  prerequisiteCodes String? // Comma-sepa

  // Relationships
  sections     Section[]
  completedBy  CompletedCourse[]
  inProgressBy InProgressCourse[]
  pendingBy    PendingCourse[]
  remainingBy  RemainingCourse[]
}
```

```prisma
model Section {
  id                String  @id // Using section ID like "CMPS151_L01"
  section_no        String
  meeting_time      String
  total_no_of_seats Int
  validated         Boolean @default(false)
  status            String // "in_progress" or "pending"

  // Relationships
  course   Course @relation(fields: [courseId], references: [id], onDelete: Cascade)
  courseId String

  // Instructor relationship - one instructor per section
  instructor   Instructor @relation(fields: [instructorId], references: [id])
  instructorId String

  // Student enrollments (many-to-many with explicit junction table)
  studentEnrollments StudentSection[]
}
```

```prisma
model StudentSection {
  // Junction table for student-section many-to-many relationship
  student   Student @relation(fields: [studentId], references: [id], onDelete: Cascade)
  studentId String
  section   Section @relation(fields: [sectionId], references: [id], onDelete: Cascade)
  sectionId String

  // Composite primary key
  @@id([studentId, sectionId])
}
```

```prisma
model Student {
  id               String @id // Using stud
  name             String
  userName         String @unique
  password         String
  gpa              Float
  major            String
  current_year     String
  current_semester String
  advisorName      String
  advisorOffice    String
  advisorEmail     String

  // Course status relationships
  completedCourses  CompletedCourse[]
  inProgressCourses InProgressCourse[]
  pendingCourses    PendingCourse[]
  remainingCourses  RemainingCourse[]

  // Section enrollments
  enrolledSections StudentSection[]
}
```

```
model Instructor {
  id       String @id // Using instructor ID lik
  name     String
  userName String @unique
  password String

  // Sections taught by this instructor
  sections Section[]
}

model Admin {
  id       Int    @id @default(autoincrement())
  userName String @unique
  password String
}
```

```
model CompletedCourse {
  id        Int     @id @default(autoincrement())
  student   Student @relation(fields: [studentId], references: [id], onDelete: Cascade)
  studentId String
  course    Course  @relation(fields: [courseId], references: [id], onDelete: Cascade)
  courseId  String
  semester  String
  grade     String

  @@unique([studentId, courseId], name: "studentId_courseId")
}

model InProgressCourse {
  id Int @id @default(autoincrement())

  // Student relationship
  student   Student @relation(fields: [studentId], references: [id], onDelete: Cascade)
  studentId String

  // Course relationship
  course    Course @relation(fields: [courseId], references: [id], onDelete: Cascade)
  courseId String

  // Additional fields
  semester String
  status   String @default("In Progress")

  // Ensure a student doesn't have duplicate in-progress course entries
  @@unique([studentId, courseId], name: "studentId_courseId")

}
```

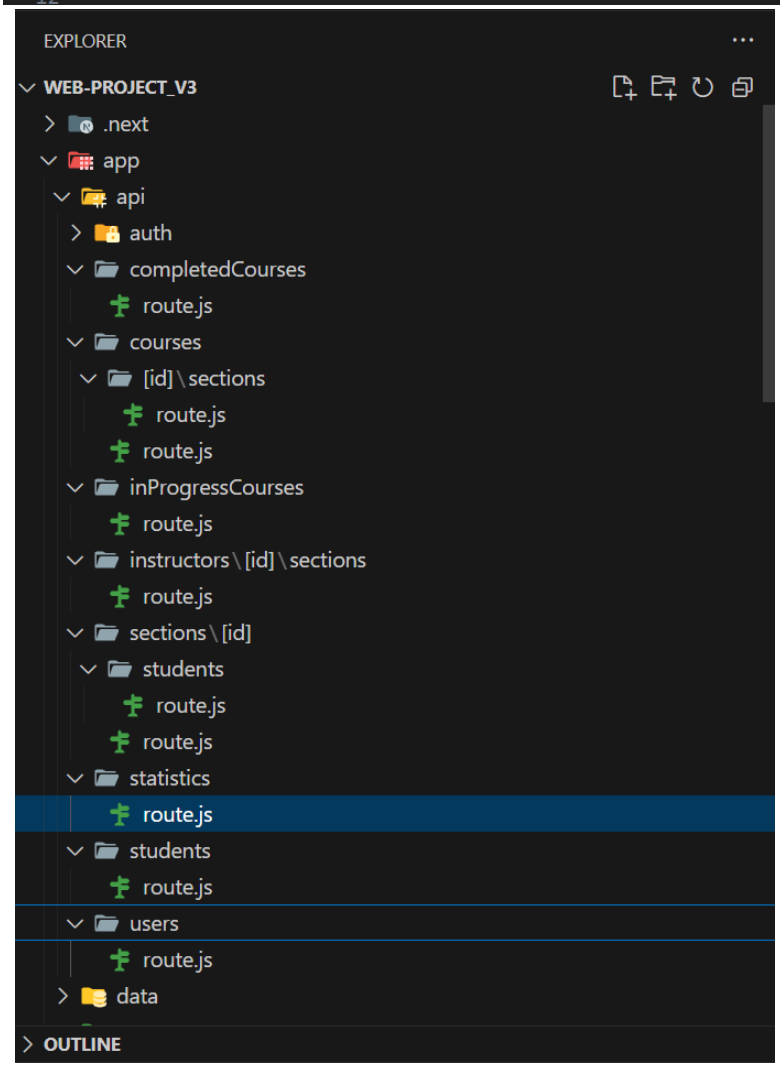
```prisma
model PendingCourse {
  id Int @id @default(autoincrement())

  // Student relationship
  student   Student @relation(fields: [studentId], references: [id], onDelete: Cascade)
  studentId String

  // Course relationship
  course   Course @relation(fields: [courseId], references: [id], onDelete: Cascade)
  courseId String

  // Additional fields
  semester String
  status   String @default("Registration Confirmed")

  // Ensure a student doesn't have duplicate pending course entries
  @@unique([studentId, courseId], name: "studentId_courseId")

}

model RemainingCourse {
  id Int @id @default(autoincrement())

  // Student relationship
  student   Student @relation(fields: [studentId], references: [id], onDelete: Cascade)
  studentId String

  // Course relationship
  course   Course @relation(fields: [courseId], references: [id], onDelete: Cascade)
  courseId String

  // Additional fields
  year String

  // Ensure a student doesn't have duplicate remaining course entries
  @@unique([studentId, courseId], name: "studentId_courseId")

}
```

3. Web API, Server Actions and repository

`GET /api/statistics`: Calls `getStatistics()` from `StatisticsRepo` and returns the statistics in JSON.

app > api > statistics > route.js > ...

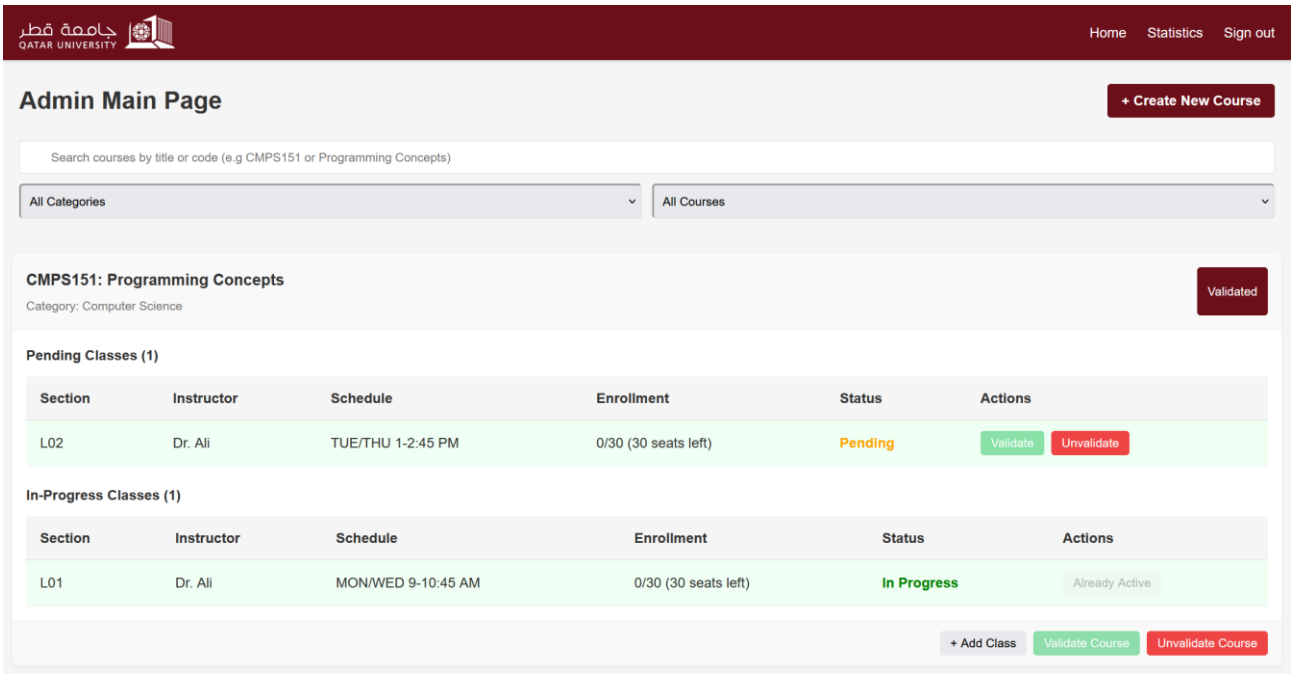
```
1  import repo from "../../repo/StatisticsRepo";
2
3  export async function GET(request) {
4    try{
5      const statistics = await repo.getStatistics()
6      return Response.json({ success: true, data: statistics });
7    } catch (error) {
8      console.error("Error fetching statistics:", error);
9      return Response.json({ success: false, message: "Internal Server Error" }, { status: 500 });
10   }
11 }
12
```

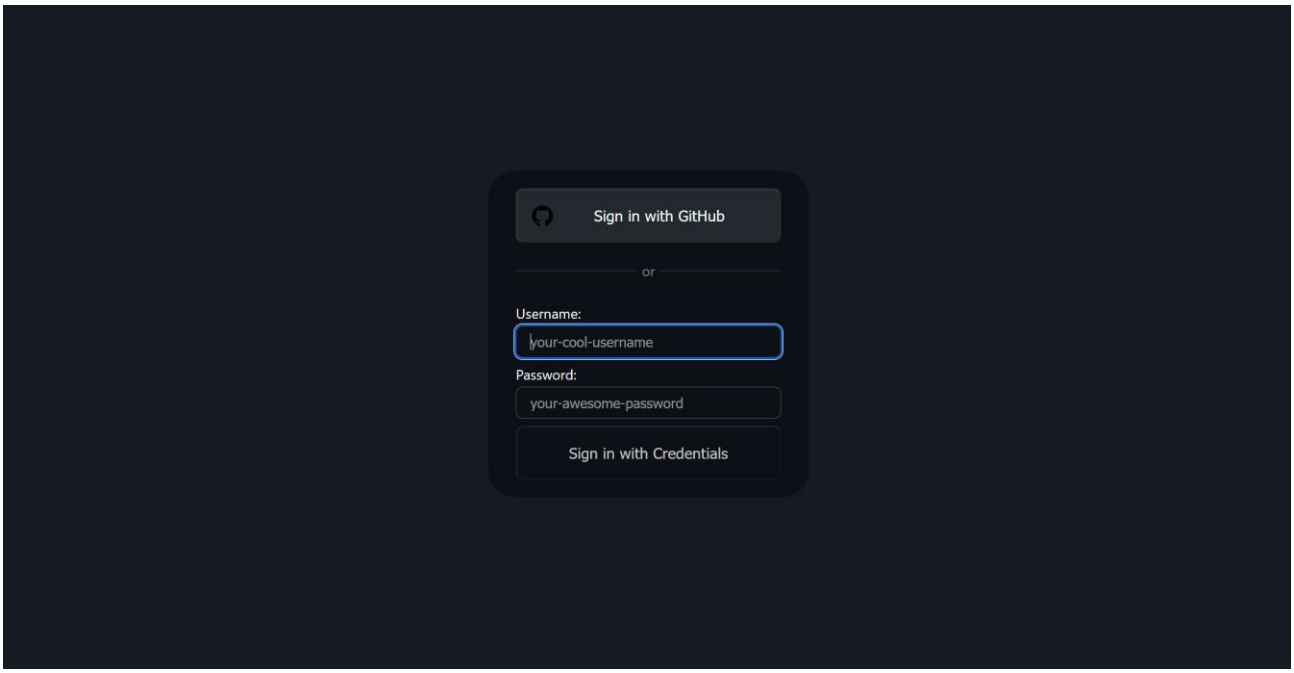


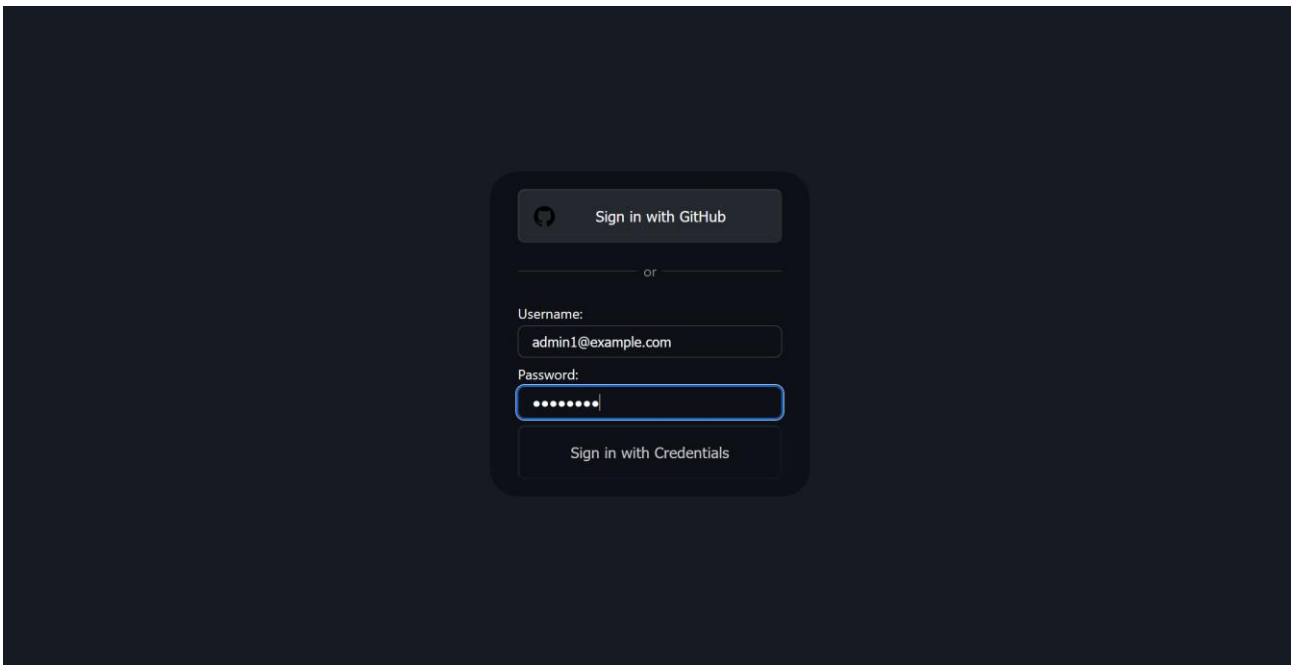
4. Implemented statistics use case

4.1. User Interface

We put the statistics page under the Admin role only not the student nor the instructor



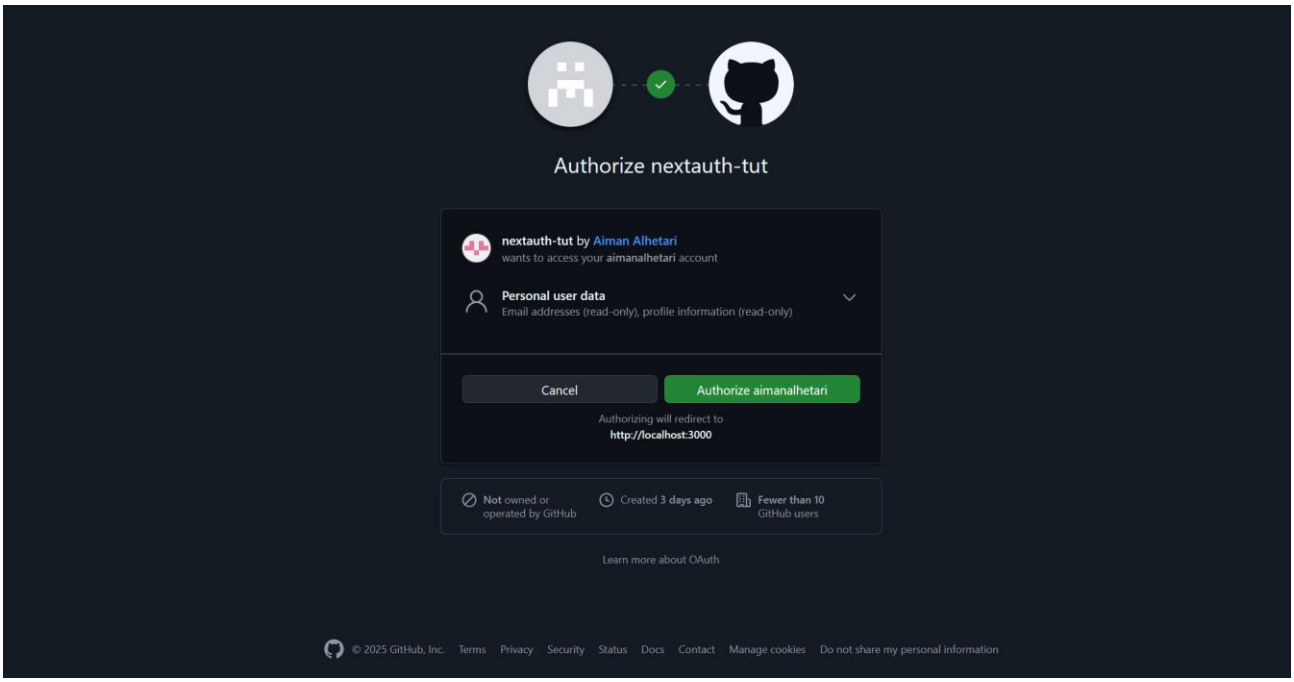




CredentialsProvider:

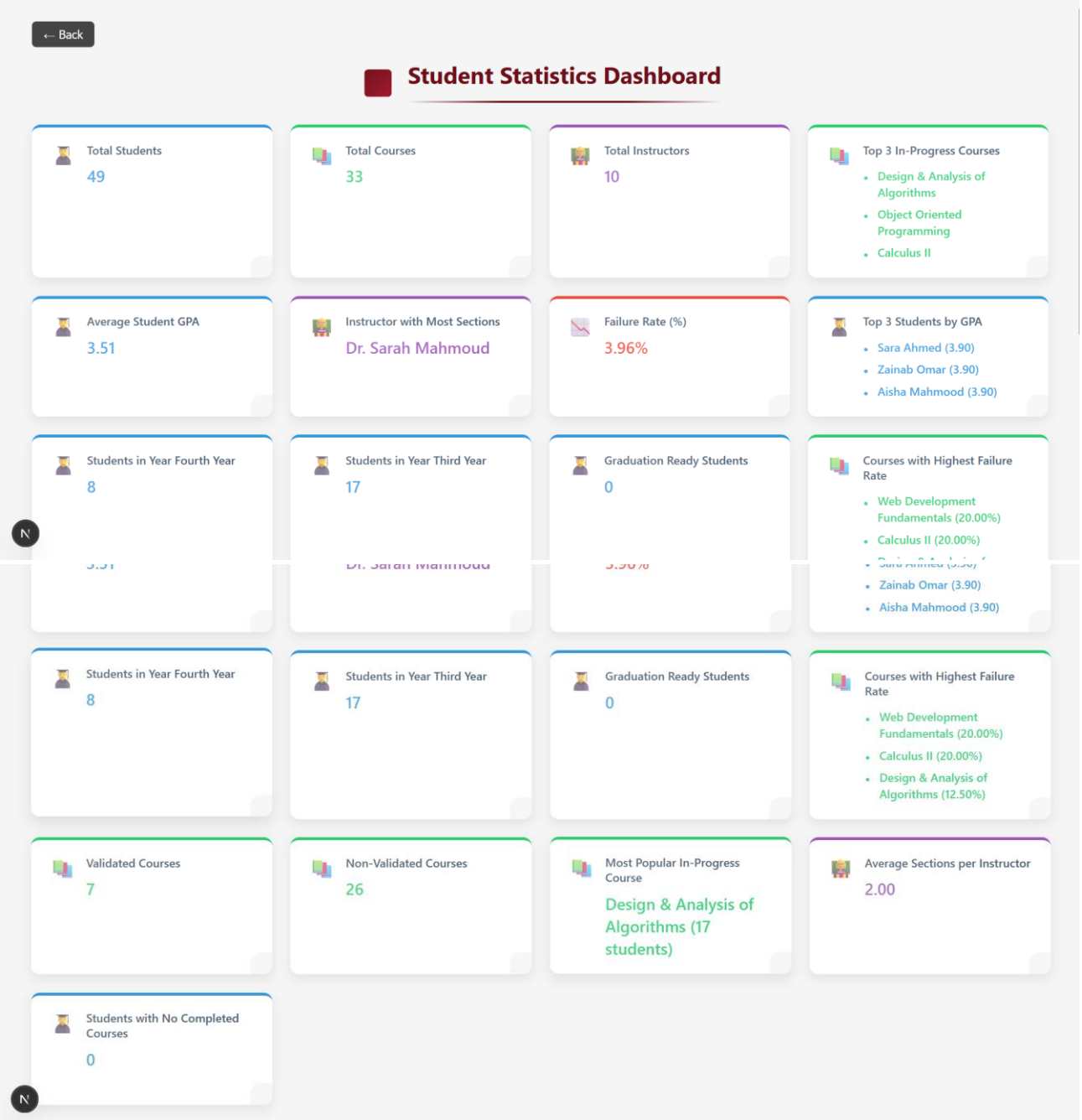
username: admin1@example.com

Password: admin123



GitHubProvider

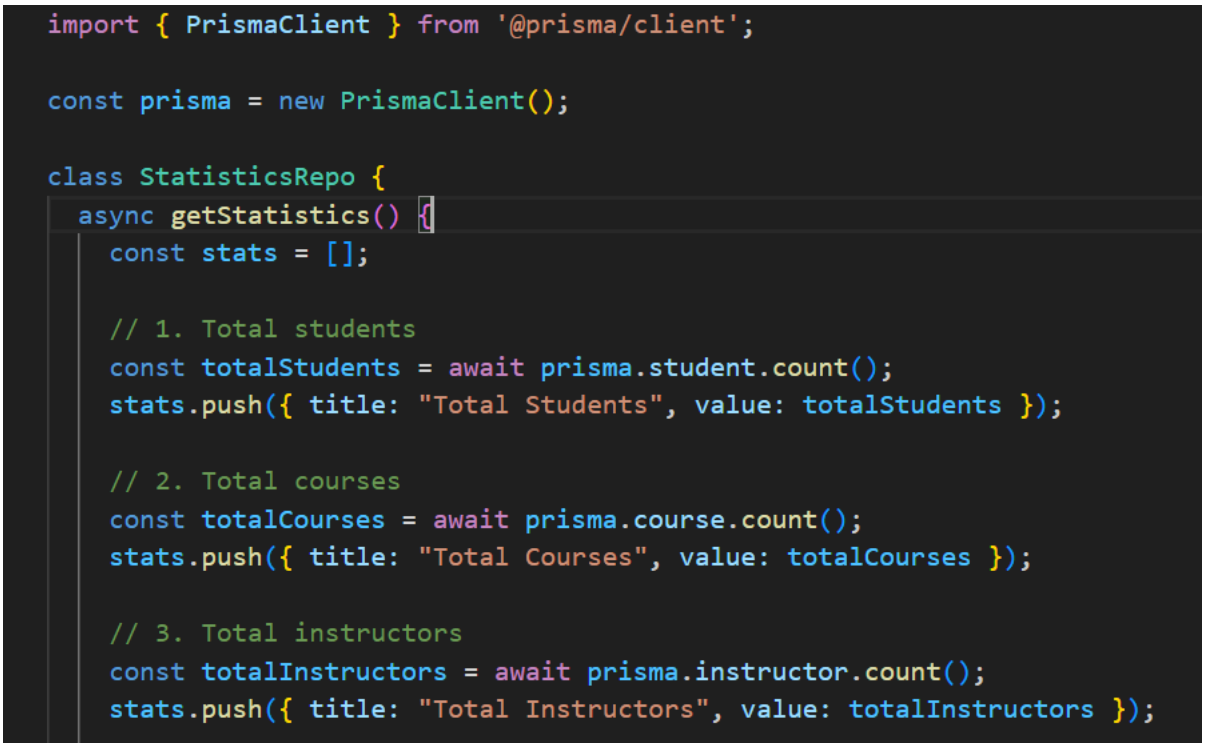
The frontend dashboard displays a list of statistics in a card/grid format. Each card shows the title and value returned by the getStatistics() method.



4.2. Implemented queries

Implemented Queries in Repository:

1. Total Students
2. Total Courses
3. Total Instructors
4. Top 3 In-Progress Courses
5. Average Student GPA
6. Instructor with Most Sections
7. Failure Rate
8. Top 3 Students by GPA
9. Students per year (excluding Second Year)
10. Graduation Ready Students
11. Top 3 Courses by Failure Rate
12. Validated vs Non-Validated Courses
13. Most Popular In-Progress Course
14. Average Sections per Instructor
15. Students with No Completed Courses



```
import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

class StatisticsRepo {
  async getStatistics() {
    const stats = [];

    // 1. Total students
    const totalStudents = await prisma.student.count();
    stats.push({ title: "Total Students", value: totalStudents });

    // 2. Total courses
    const totalCourses = await prisma.course.count();
    stats.push({ title: "Total Courses", value: totalCourses });

    // 3. Total instructors
    const totalInstructors = await prisma.instructor.count();
    stats.push({ title: "Total Instructors", value: totalInstructors });
```

```
// 4. Number of students currently in progress per course
const inProgress = await prisma.inProgressCourse.groupBy({
  by: ["courseId"],
  _count: {
    courseId: true
  },
  orderBy: {
    _count: {
      courseId: "desc"
    }
  },
  take: 3,
});

const inProgressNames = await Promise.all(
  inProgress.map((c) =>
    prisma.course.findUnique({
      where: { id: c.courseId },
      select: { title: true },
    })
  )
);

stats.push({
  title: "Top 3 In-Progress Courses",
  value: inProgressNames.map((c) => c.title).join(", "),
});

// 5. Average GPA of all students
const avgGpa = await prisma.student.aggregate({
  _avg: { gpa: true },
});
stats.push({
  title: "Average Student GPA",
  value: avgGpa._avg.gpa?.toFixed(2) || "N/A",
});
```

```
// 6. Instructor with Most Sections
const topInstructor = await prisma.section.groupBy({
  by: ["instructorId"],
  _count: {
    instructorId: true
  },
  orderBy: {
    _count: {
      instructorId: "desc"
    }
  },
  take: 1,
});

if (topInstructor.length > 0) {
  const instructor = await prisma.instructor.findUnique({
    where: { id: topInstructor[0].instructorId },
    select: { name: true },
  });
  stats.push({
    title: "Instructor with Most Sections",
    value: instructor?.name || "N/A",
  });
} else {
  stats.push({
    title: "Instructor with Most Sections",
    value: "N/A",
  });
}
```

```
// 7. Failure Rate (% of F grades)
const totalGrades = await prisma.completedCourse.count();
const totalFails = await prisma.completedCourse.count({
  where: { grade: "F" },
});
const failureRate =
  totalGrades > 0 ? ((totalFails / totalGrades) * 100).toFixed(2) : "0.00";
stats.push({
  title: "Failure Rate (%)",
  value: `${failureRate}%`,
});

// 8. Students with Highest GPA (Top 3)
const topStudents = await prisma.student.findMany({
  orderBy: { gpa: "desc" },
  take: 3,
  select: { name: true, gpa: true },
});
stats.push({
  title: "Top 3 Students by GPA",
  value: topStudents.map((s) => `${s.name} (${s.gpa.toFixed(2)})`).join(", "),
});

// 9. Number of students per year
const yearGroups = await prisma.student.groupBy({
  by: ["current_year"],
  _count: {
    _all: true
  },
});
yearGroups.forEach((group) => {
  if (group.current_year !== "Second Year") {
    stats.push({
      title: `Students in Year ${group.current_year}`,
      value: group._count._all,
    });
  }
});
```

```
// 10. Number of students who completed all their courses (graduation ready)
const graduationReadyStudents = await prisma.student.findMany({
  where: {
    remainingCourses: {
      none: {}
    }
  },
  select: { id: true }
});
stats.push({
  title: "Graduation Ready Students",
  value: graduationReadyStudents.length,
});
```

4.3. Data used in the statics

Student Table: for GPA, current year, remaining/completed courses

Course Table: for titles and validation

Instructor Table: for names and section count

InProgressCourse Table: for current enrollments

CompletedCourse Table: for grades and completions

Section Table: for instructor-course relationships

4.4. Conducted tests

Test for Statistics API