School Assistance Dashboard Report

1. **School Assistance Dashboard**
2. **Delegation of tasks:**

Everyone – All members participated in project documentation, presentation, and overall quality assurance. Each member was responsible for completing assigned components, and assisting others as needed.
Hosanna Pyles - Established the necessary basic assumptions for the project. Wrote sample code for calculator and corresponding unit tests and an explanation of the testing process.
Thomas Ewing - Worked on sequence diagrams, compared the project with similar products, and cited references.
Saulo Contreras - Worked out class diagrams, overall project timeline, and planned phases.
Awsaf Kabir - Created the Architecture diagram and assisted with the Cost Estimation calculation with Faiz Khan.
Faiz Khan - Created the use case diagrams, and assisted with the Cost estimation with Awsaf Kabir.
Govind Rangappa - Worked on task delegations for deliverable 1 as well as feedback from deliverable 1 and the conclusion in deliverable 2.
Joshua John - Determined the software process model and functional/non-functional requirements.

3. **Everything Submitted in Project Deliverable #1**

Assumptions:

a. Syllabi can always or almost always be expected to either be in or be able to be converted to .pdf or .docx format.
b. The eLearning class data is stored in a way that is compatible with an ASP.NET Core format of web application.
c. We will be given access to the UTD eLearning API calls.
d. We are legally allowed to access eLearning information.

Addressing Grader Feedback

Feedback given from Grader:

"Well done.

The tasks section only refers to mainly implementation tasks. Include tasks with respect to requirements gathering, design and testing. (for eg- who will create use case diagrams etc)"

We addressed our grader's feedback by detailing our task delegation a little more clearly and adding in design, testing, and requirements gathering tasks(Point 2). As a group we have also discussed how further tasks will only be found upon getting to each next deliverable and so our task delegation list is of a dynamic nature rather than set in stone from the start. So just like the last, this revised task delegation list(Point 2) is definitely subject to change with both additions and omissions.

Software Process Model

As a team, we decided to select the incremental model for our SDLC. One of the main reasons why we decided to pivot towards this model is that it encompasses increments and evaluation of those incremental pieces in our project. This also means whenever we have high uncertainty about a component we think may or may not be helpful to the system, we can scrutinize it before it gets pushed into the overall framework of the system. The benefits mean that we can produce an early usable product while incrementally adding features. These features don't necessarily have to be complex upon release. We can even afford to make minor output increments, while encompassing bug fixes and updates more easily with full feedback. This also fits since our project is quite flexible with what we want incorporated into our product.

Functional/Non-Functional Requirements

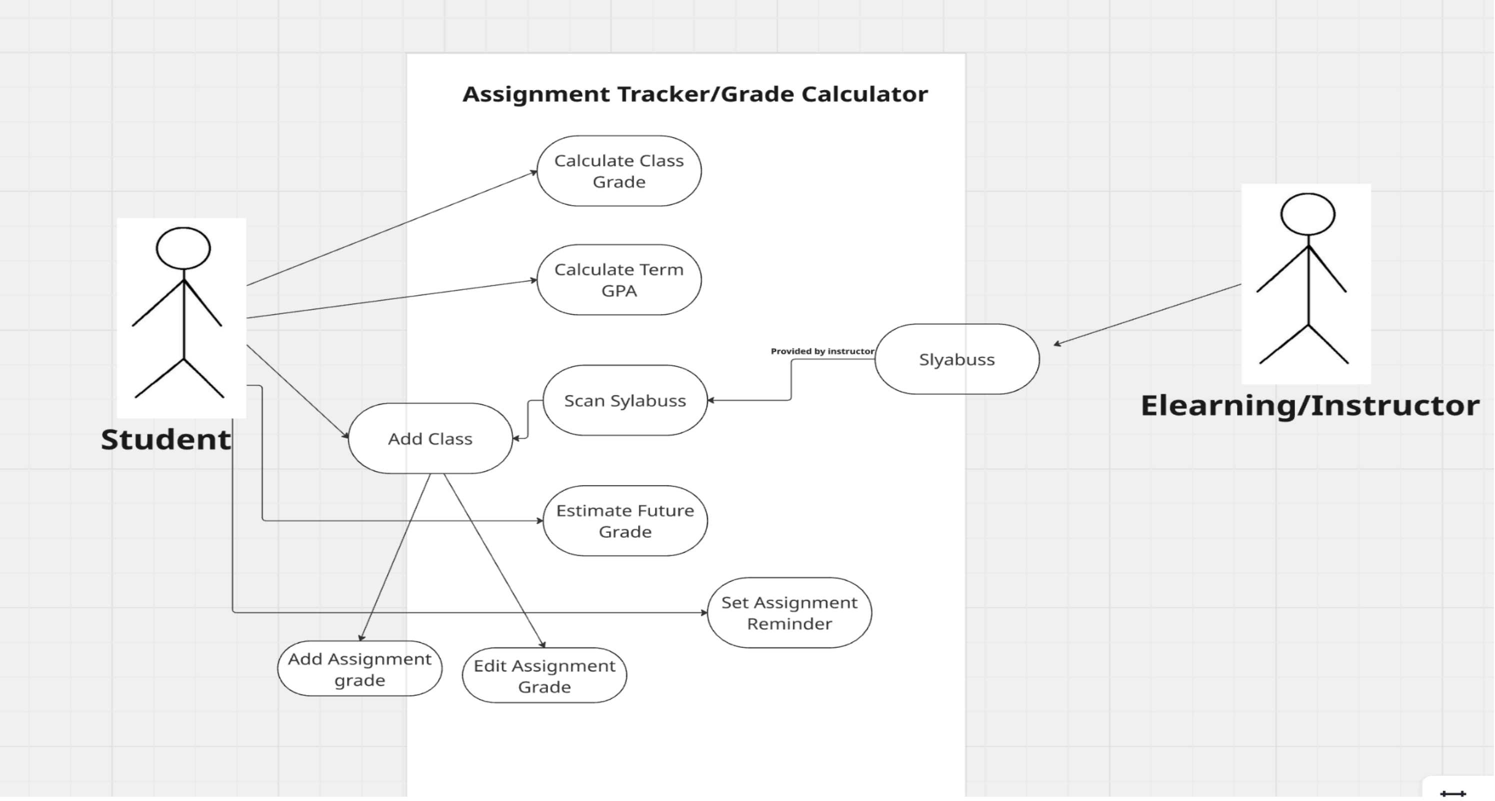
Functional requirements:

- The product must be able to calculate the overall grade based on the weights provided by the syllabus
- The product must allow you to keep track of your attendance for classes
- The product should be flexible to grading schemes of professors based on syllabuses, including unpredictable curves to grades, as part of the calculation
- The project must be capable of tracking future assignments when provided by users
- The product must possess the ability to forecast effects of hypothetical future assignment grades, so that students will know how they need to perform to get the grades they want.

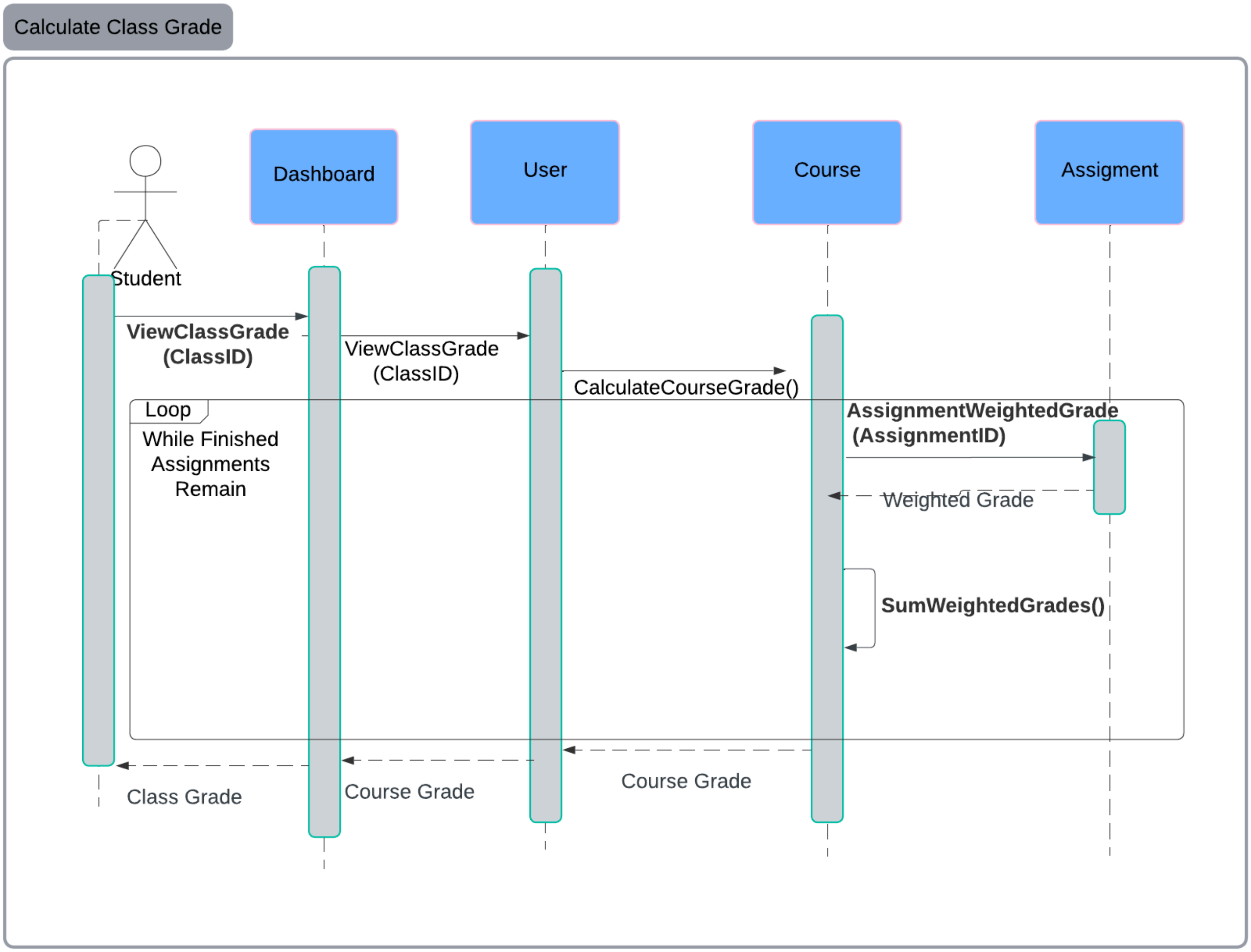
Non-functional requirements:

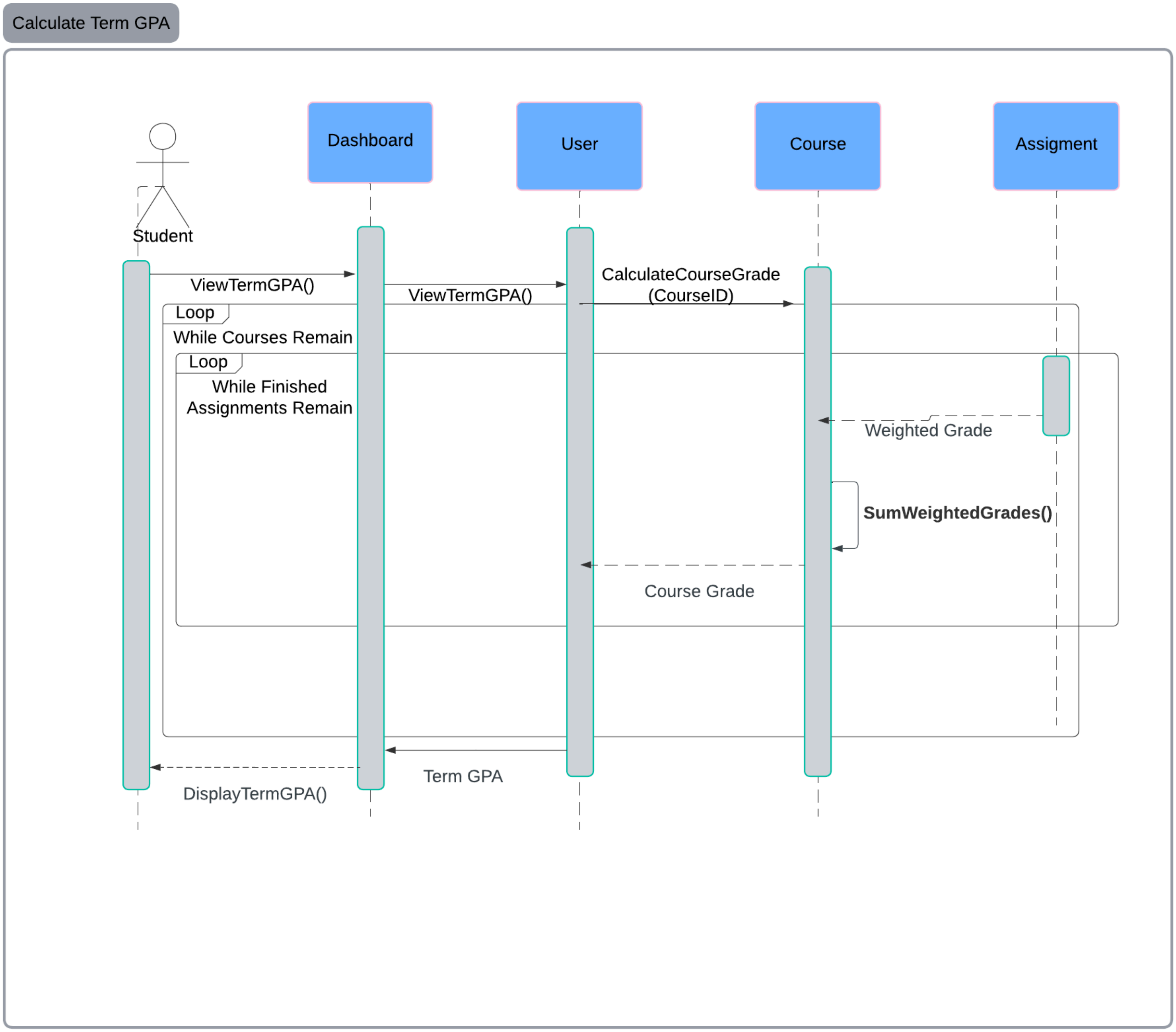
- Usability: the system must be simpler to use than the current Elearning system
- Scalability: the system should scale up and down to different operating systems; both laptops and mobile devices
- Portability: should be accessible/easy-to-use on platforms outside the main website
- Reliability: the system should not crash when permissions or information are unavailable

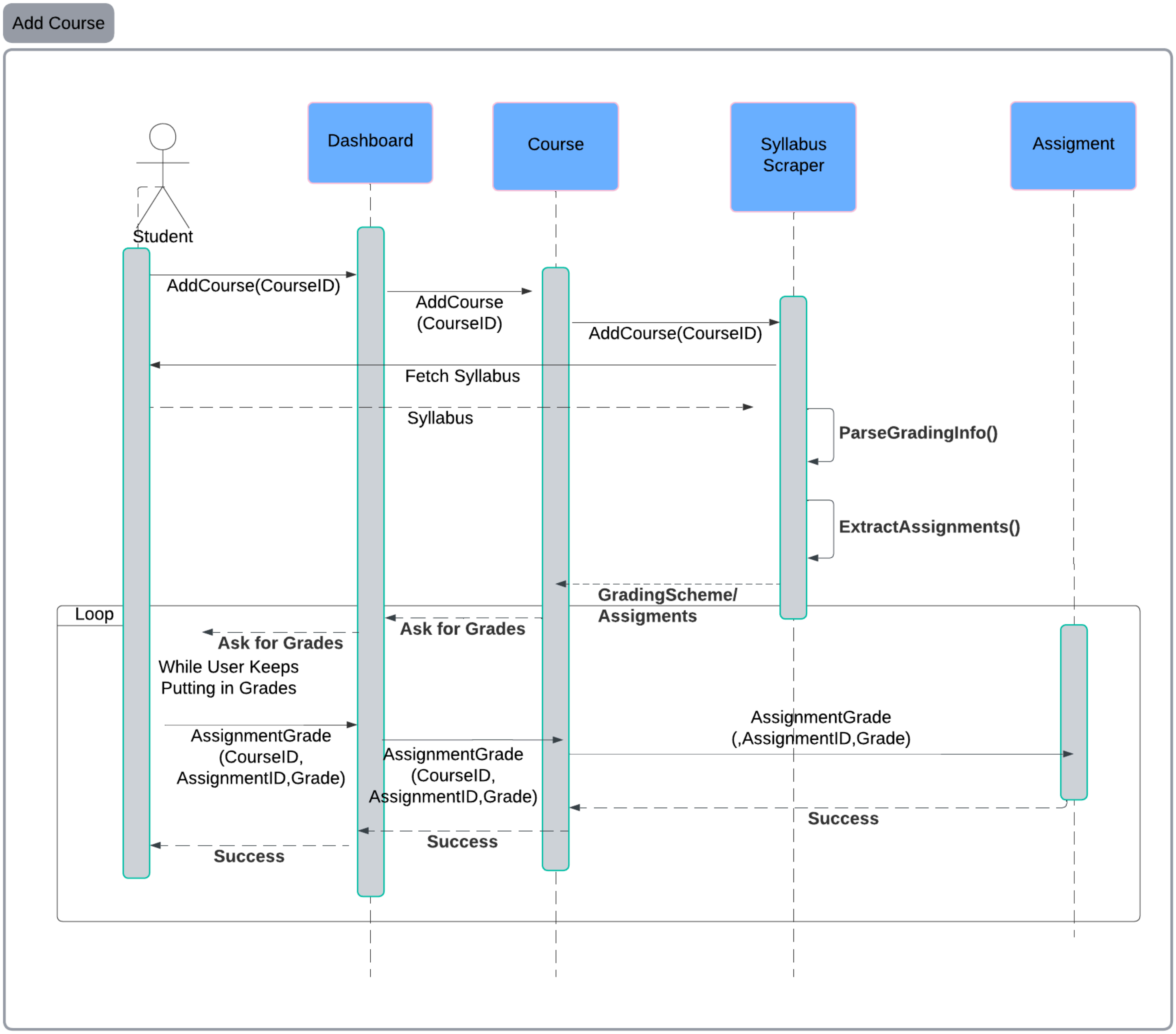
Use Case Diagram

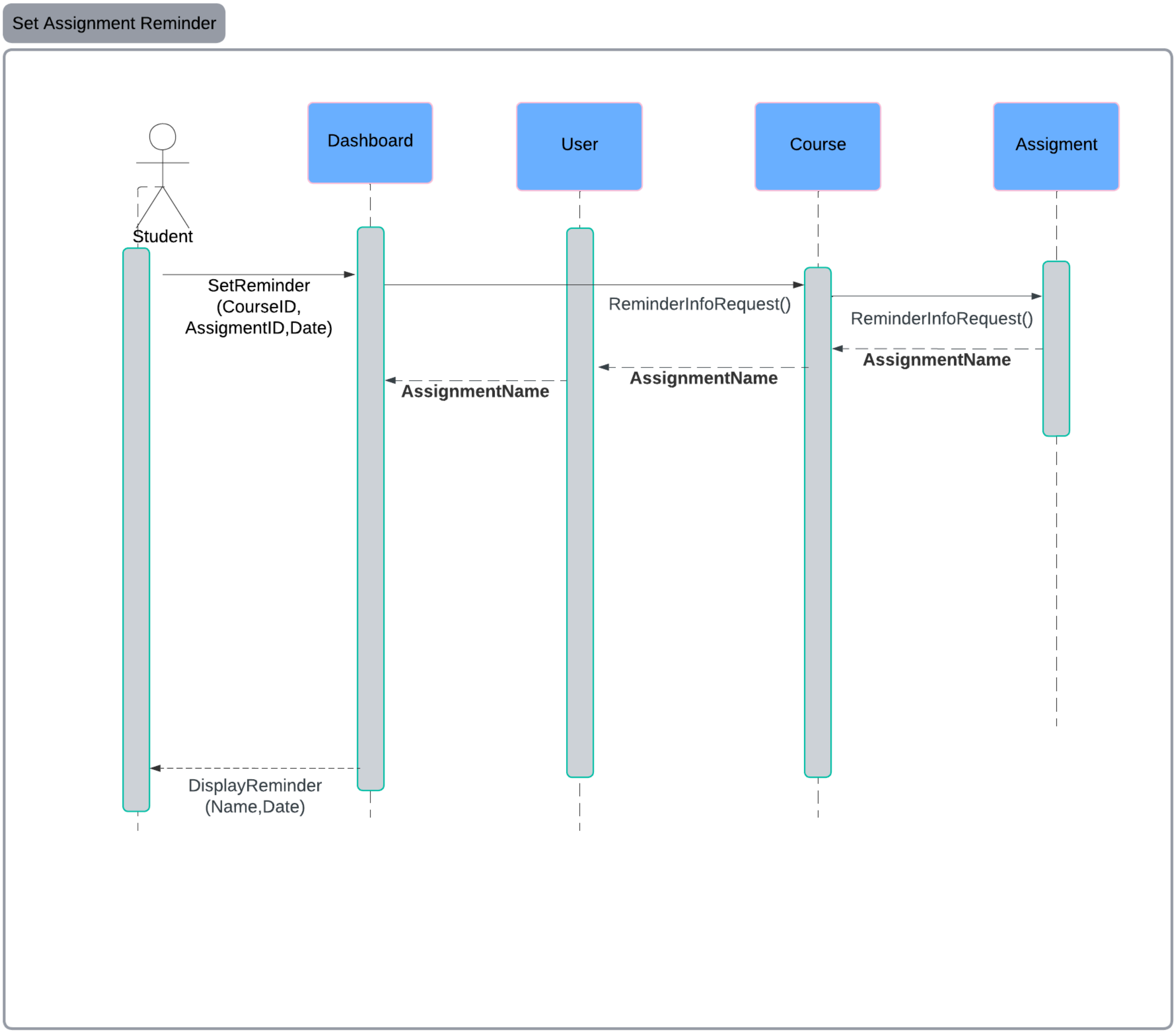


Sequence Diagrams (See Below)

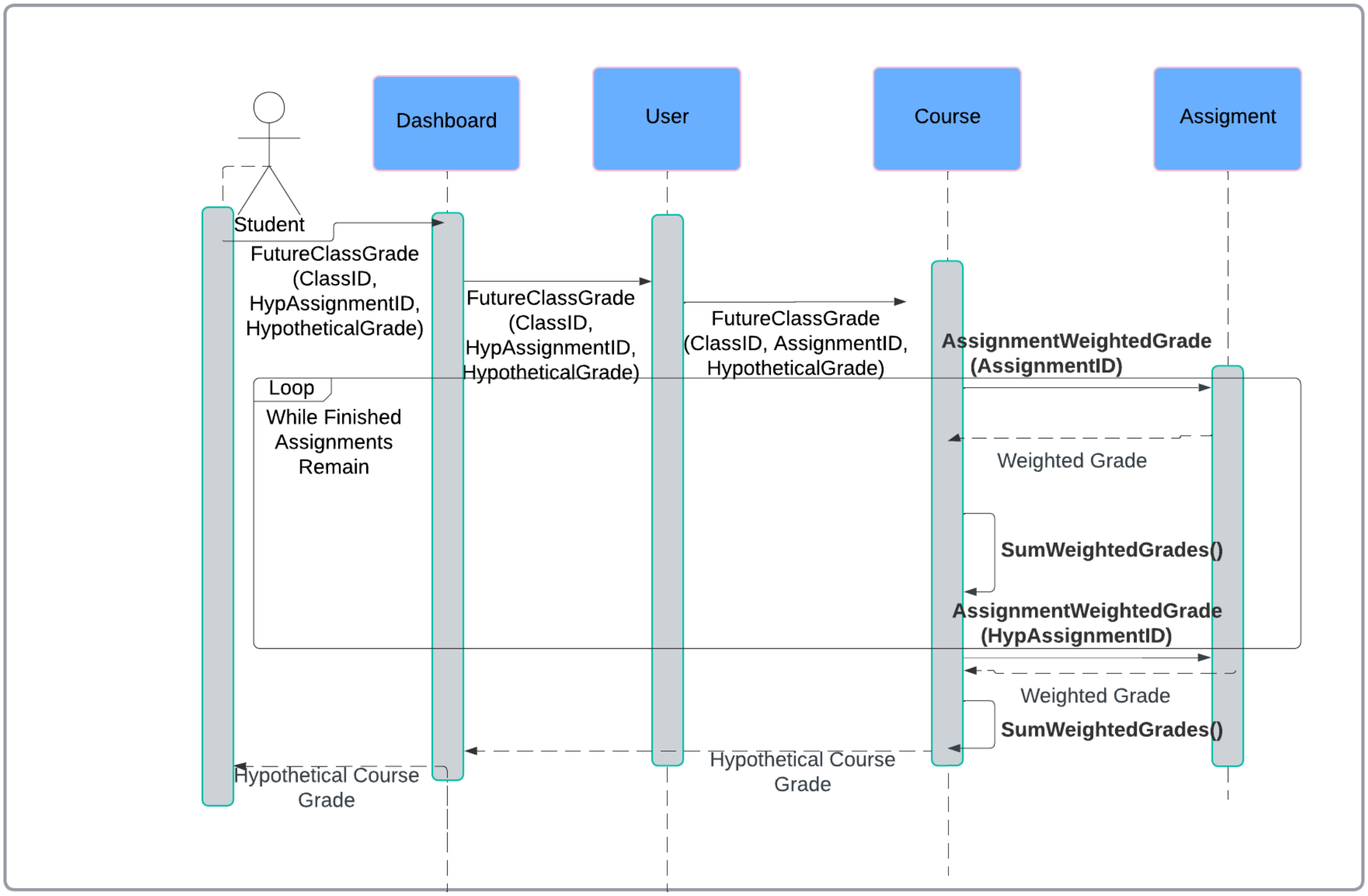



Calculate Term GPA
Dashboard
User
Course
Assigment
Student
ViewTermGPA()
ViewTermGPA()
CalculateCourseGrade
(CourseID)
Loop
While Courses Remain
Loop
While Finished
Assignments Remain
Weighted Grade
SumWeightedGrades()
Course Grade
Term GPA
DisplayTermGPA()

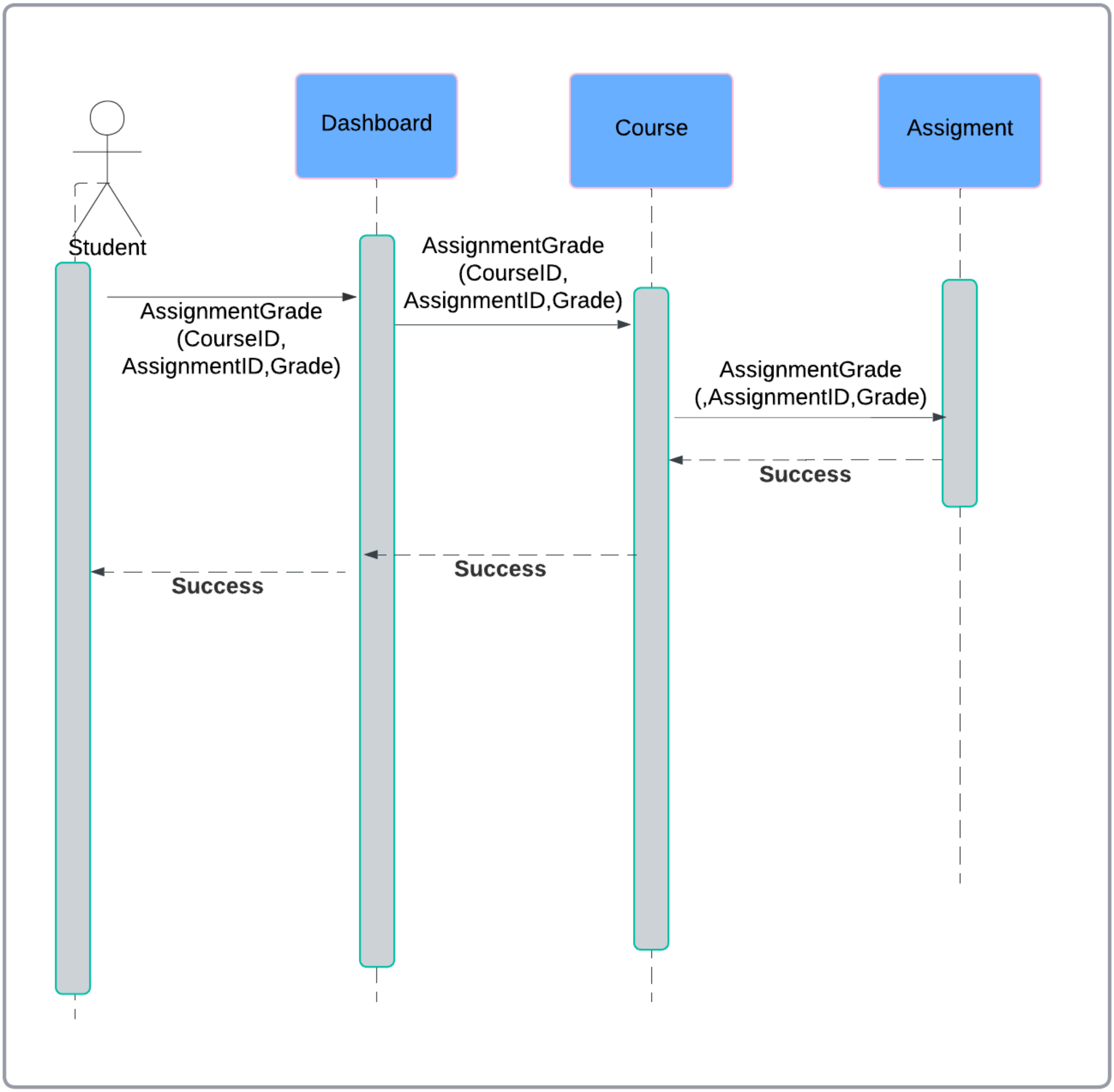

Add Course
Student
Dashboard
Course
Syllabus Scraper
Assigment
AddCourse(CourseID)
AddCourse (CourseID)
AddCourse(CourseID)
Fetch Syllabus
Syllabus
ParseGradingInfo()
ExtractAssignments()
GradingScheme/ Assigments
Loop
Ask for Grades
Ask for Grades
While User Keeps Putting in Grades
AssignmentGrade (CourseID, AssignmentID,Grade)
AssignmentGrade (CourseID, AssignmentID,Grade)
AssignmentGrade (,AssignmentID,Grade)
Success
Success
Success

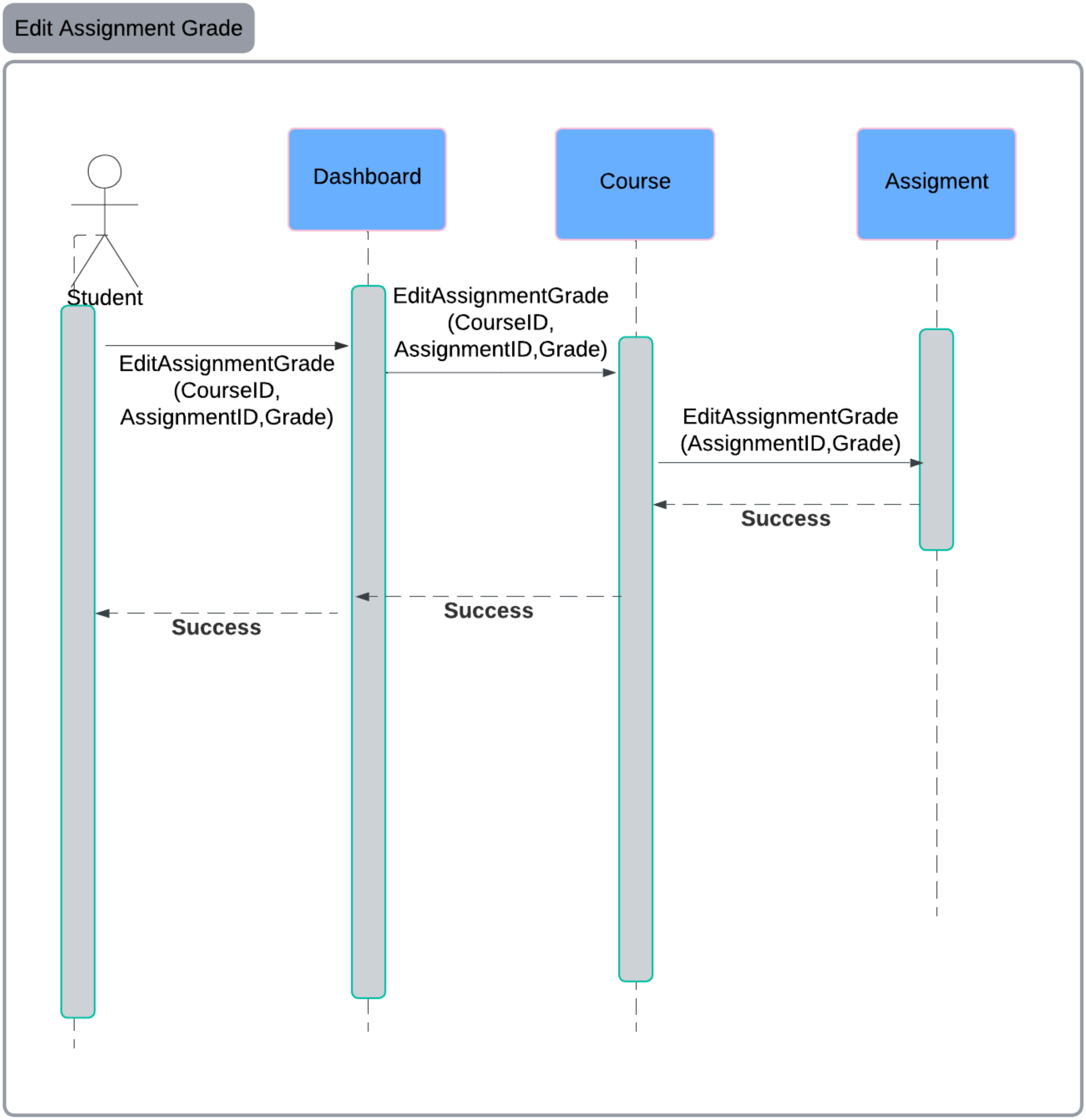

Set Assignment Reminder
Student
Dashboard
User
Course
Assigment
SetReminder
(CourseID,
AssigmentID,Date)
ReminderInfoRequest()
ReminderInfoRequest()
AssignmentName
AssignmentName
AssignmentName
DisplayReminder
(Name,Date)

Calculate Future Class Grade

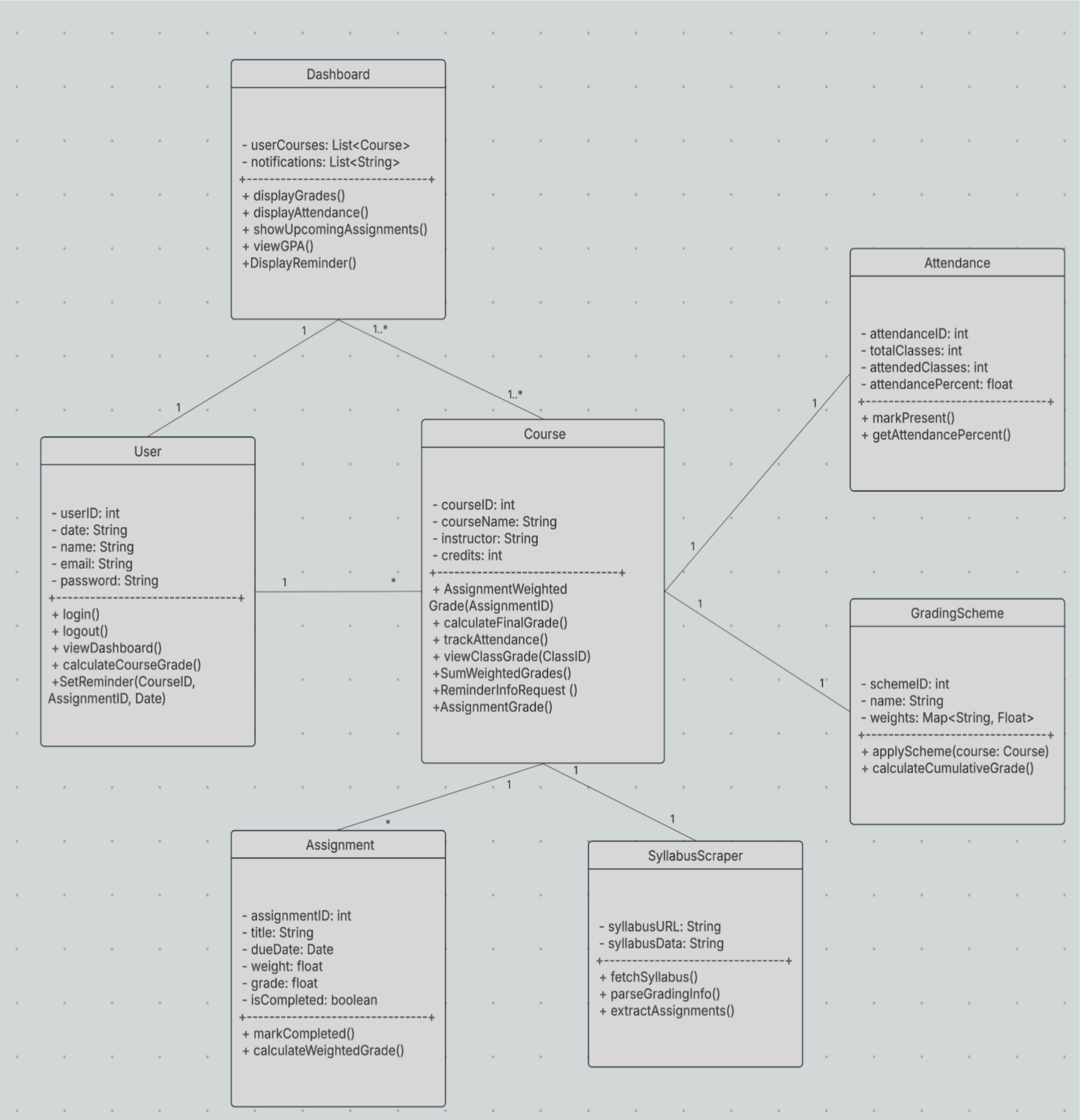

Dashboard
User
Course
Assigment
Student
FutureClassGrade (ClassID, HypAssignmentID, HypotheticalGrade)
FutureClassGrade (ClassID, HypAssignmentID, HypotheticalGrade)
FutureClassGrade (ClassID, AssignmentID, HypotheticalGrade)
AssignmentWeightedGrade (AssignmentID)
Loop
While Finished Assignments Remain
Weighted Grade
SumWeightedGrades()
AssignmentWeightedGrade (HypAssignmentID)
Weighted Grade
SumWeightedGrades()
Hypothetical Course Grade
Hypothetical Course Grade

Add/Edit Assignment Grade


Dashboard
Course
Assigment
Student
AssignmentGrade (CourseID, AssignmentID,Grade)
AssignmentGrade (CourseID, AssignmentID,Grade)
AssignmentGrade (,AssignmentID,Grade)
Success
Success
Success


Edit Assignment Grade
Dashboard
Course
Assigment
Student
EditAssignmentGrade
(CourseID,
AssignmentID,Grade)
EditAssignmentGrade
(CourseID,
AssignmentID,Grade)
EditAssignmentGrade
(AssignmentID,Grade)
Success
Success
Success

Class Diagram - GOVIND



Dashboard

- userCourses: List<Course>
- notifications: List<String>
+-------------------------------+
+ displayGrades()
+ displayAttendance()
+ showUpcomingAssignments()
+ viewGPA()
+DisplayReminder()

User

- userID: int
- date: String
- name: String
- email: String
- password: String
+-------------------------------+
+ login()
+ logout()
+ viewDashboard()
+ calculateCourseGrade()
+SetReminder(CourseID, AssignmentID, Date)

Course

- courseID: int
- courseName: String
- instructor: String
- credits: int
+-------------------------------+
+ AssignmentWeighted Grade(AssignmentID)
+ calculateFinalGrade()
+ trackAttendance()
+ viewClassGrade(ClassID)
+SumWeightedGrades()
+ReminderInfoRequest ()
+AssignmentGrade()

Attendance

- attendanceID: int
- totalClasses: int
- attendedClasses: int
- attendancePercent: float
+-------------------------------+
+ markPresent()
+ getAttendancePercent()

GradingScheme

- schemeID: int
- name: String
- weights: Map<String, Float>
+-------------------------------+
+ applyScheme(course: Course)
+ calculateCumulativeGrade()

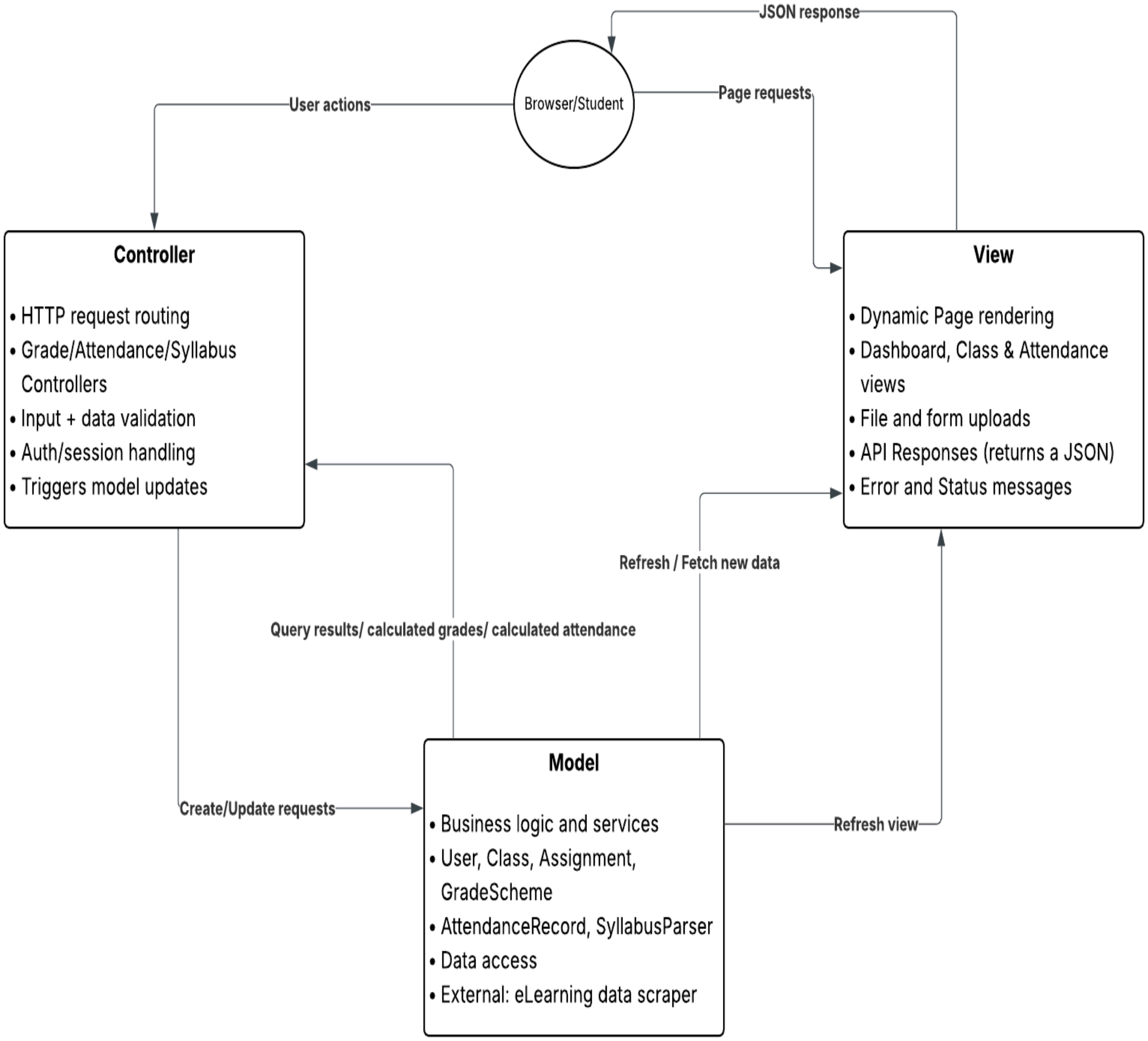
Assignment

- assignmentID: int
- title: String
- dueDate: Date
- weight: float
- grade: float
- isCompleted: boolean
+-------------------------------+
+ markCompleted()
+ calculateWeightedGrade()

SyllabusScraper

- syllabusURL: String
- syllabusData: String
+-------------------------------+
+ fetchSyllabus()
+ parseGradingInfo()
+ extractAssignments()

Architectural Design


JSON response
Browser/Student
User actions
Page requests
Controller
• HTTP request routing
• Grade/Attendance/Syllabus Controllers
• Input + data validation
• Auth/session handling
• Triggers model updates
View
• Dynamic Page rendering
• Dashboard, Class & Attendance views
• File and form uploads
• API Responses (returns a JSON)
• Error and Status messages
Refresh / Fetch new data
Query results/ calculated grades/ calculated attendance
Create/Update requests
Refresh view
Model
• Business logic and services
• User, Class, Assignment, GradeScheme
• AttendanceRecord, SyllabusParser
• Data access
• External: eLearning data scraper

4. Project Scheduling, Cost, Effort, and Pricing Estimation, Project duration and staffing

a) Project Scheduling

Initialization & Scope (Aug 25 – Sep 5)

- Define goals, feasibility, and requirements.

Planning & Production Phase 1 (Sep 8 – Sep 26)

- Architecture, core diagrams, early feature development.
- Focus: system architecture, grade calculator, attendance.

Planning & Production Phase 2 (Sep 29 – Oct 17)

- Continue design + build features.
- Focus: UI integration, syllabus scraping, data handling.

Launch / Final Integration (Oct 20 – Oct 31)

- Final development, system-wide testing, UI polish, presentation prep.

Closing & Post-Launch (Nov 3 – Nov 14)

- Final documentation, retrospective, cleanup.

Project Timeline

We begin the project on August 25th, and look to conclude around November 14th, which aligns with a standard semester timeline and gives us enough space for incremental development and testing. Our work week follows a Monday–Friday schedule, with 1 - 3 hour work days, and we will not formally count weekends in our plan, although members may choose to work individually during that time

Phase 1 is Initialization and Scope, lasting the first two weeks. Here, we gather requirements, define the system's goals, validate our assumptions about syllabus formats and API access, and produce the initial documentation. This gives us a clear foundation before development begins. our next phase is split into two parts

Our next step, planning and production is split into two separate phases, each taking about 3 weeks. The first half will be looking to create our architectural design and design our first working models of the grade calculator and attendance tracker, while updating any diagrams or documentation as problems may arrive.

In the second half we look to create our user-friendly UI, get a model of the syllabus scraper working, fully integrating all models into one cohesive application, and updating task delegation based on implementation discoveries.

Phase 4 is the Launch Phase, also taking two weeks, where the whole team focuses on integration, testing, bug fixing, and refining the user experience. We ensure all features work together smoothly and prepare the project for presentation.

Finally, Phase 5 is the Closing and Post-Launch, which will be 2 weeks, where we finalize documentation, clean up the repository, prepare our final product, and reflect on lessons learned throughout the project for future endeavors and future maintenance

b) Cost, Effort, and Pricing Estimation

With the team, we chose to implement the Application Composition Model. A sub-model of COCOMO II Software Cost estimation suite, this model estimates Effort based on calculating the Object Points and productivity rate.[1]

Our application consists of 10 primary user-facing screens:

1. login page
2. home page
3. attendance tracker
4. attendance editor
5. grade calculator page
6. grade editor page
7. syllabus upload page
8. parsed syllabus page
9. assignment tracker
10. Settings.

It also includes 3 generated reports

1. attendance summaries
2. grade summaries
3. predicted results

5 core logic modules

1. grade calculations
2. attendance logic
3. syllabus scraping
4. assignment reminders
5. data handling.

Altogether, these components total 18 Object Points. Using the Application Composition productivity rate of 10 OP per person-month, the estimated effort for the project is 1.8 person-months.

Effort

- Total Count = 10 + 3 + 5 = 18 OP
- Effort = OP / Productivity
- 18 / 10 = 1.8 person-month
- **Effort = 1.8 person-months** (based on 18 Object Points ÷ productivity 10)

Pricing Estimation

- **Development Cost ≈ $18,000** (1.8 person-months × $10,000 per month)

c)

- **Hardware Costs ≈ $1,290** (cloud hosting, laptop depreciation, backup drives)

d)

- **Software Costs ≈ $60** (Acrobat tools; IDEs/GitHub free for students)

e)

- **Misc Costs ≈ $400** (training material, documentation, deployment)
- **Calculated Total ≈ $19,750**
- **Predicted Mark-Up = 1.20 × Total ≈ $23,700**
- **Final Estimated Cost ≈ $23,000–$24,000**

5. Test plan

We will be making use of thorough unit testing. Every individual code module will have a corresponding unit test that can be used as we progress in order to ensure our code is continuing to function as intended without any significant errors. Our test cases cover: initializing grades for a class without considering weighting, initializing grades for a class with weighting, adding an unweighted grade to the class grades, adding a weighted grade to the class grades, calculating the class' grade unweighted, and calculating weighted class grade.

The initialization tests each create an instance of the module containing the class grade calculations code using a collection of objects holding grade data, one with and one without weighing information provided. Weighted class grade initialization must also provide a collection of the weights. Then they use a function that returns the initialized collection with all the class grades and checks that they are both not null and the correct length.

The tests for adding a grade start with the same initialization process as the initialization tests, including weighting as relevant to the respective tests. If the initialization tests have been passed, we can include it here with the assumption it can be successfully used as intended. The module instance's function for adding a grade is then run with the information necessary to create the relevant grade information object. The function for adding a grade is a boolean function for testing purposes, and is checked for a result of true as a result of false indicates an error. The grades collection is returned via a function again to check that the number of objects matches the expected updated total.

The tests for calculating overall class grade once again start with the same initialization process as the initialization tests. The function for calculating overall class grade is run and the result is stored and checked against the expected result. The process for the weighted versus unweighted overall grade is entirely different, so it's important to check the result with a small margin of error (0.001).

6. Comparison with Similar Concepts

School Assistance Dashboard (Our Project) - Is a Website based planning tool that allows for precise grading with the ability to incorporate custom grading schemes for multiple classes. Attendance tracking and hypothetical grade forecasting are also prominent abilities of the product. This is all accomplished through a MVC architectural pattern with incremental development. Primary non-functional requirements for the project are that the product be easier to use than Elearning, accessible outside of the main website and be available across different operating systems.

GradeWay- An app for phones that integrates with an external server (Home Access Center Systems) which shows students their course averages alongside individual grades. The app also allows for viewing grades by category and with the addition of hypothetical future grades and their impacts to weighted/unweighted GPA.[2]

MyGrades Tracker - Another app that lets students insert their assignment grades for various courses as well as the custom grading schemes for these courses. The app then calculates grades and visualizes progress in a variety of charts for these courses. Topping it off, the app also has a standalone software capability with on device storage of relevant information. [3]

Gradekit - Another app that asks students to sign in before remembering their login. It interacts with the school's chosen platform like PowerSchool or Infinite Campus and automatically accesses their grades before displaying them. There is additional functionality in that it supports forecasting hypothetical grades to help future planning.[4] [5]

The three external products, GradeWay, MyGrades Tracker, and Gradekit offer functionality similar to our own in their core use cases. These functionalities are the calculation of course grades based on custom grading schemes for multiple courses. Many of the external products also offer adjacent functionality like our hypothetical forecasting using submitted grades and use of dashboards and graphs/charts to convey information. The more significant use cases that our product offers that do not seem to be as commonly replicated in others is the attendance tracking and ability to set reminders for future assignments. One use case that is offered by GradeWay but not in our own product is the ability to show grades by category in addition to individual courses. This is something that our School Assistance Dashboard could release in a future update since it seems helpful for student analysis which is the end goal for our product.

Beyond use cases offered, there are differences in the system implementation. This primarily presents itself in the way our product is accessed as every external product is marketed as an app instead of a website like ours. Every product is still connected to an external server but the others do so through a mobile app. Additionally, Gradekit has a feature where the product can function on a standalone basis by storing current information on the device to pull from when internet access is not available. These two features are both something that our product could expand into in future updates. Lastly, Gradekit is not student led like many of the other products but is led by educational organizations and accessed by students. School Assistance Dashboard currently is not being developed in this direction but could benefit from growing cooperation with education groups to make the automatic inclusion of grades and gradings schemes more easily accessible in the future.

7. Conclusion

Overall, our team successfully delivered the core features of the School Assistance Application as originally planned, specifically the grade calculator, attendance tracker, assignment tracking, and syllabus based automation. Throughout the project, we consistently aligned our work with the initial problem statement while refining our approach as we understood the technical constraints and user needs better

i. Changes Made Compared to the Original Plan

While our final application aligns majorly with our original idea, a few adjustments were made during project development:

- **Refinement of Syllabus Scanning Scope:**
  Originally, we aimed to scrape both eLearning and uploaded syllabi. Due to restricted API access and feasibility concerns, we shifted toward focusing on upload based syllabus scraping, which became the primary input method in our final design.

- **Revised Architectural Approach:**
  Our early guidelines did not specify the MVC structure in detail, but as development progressed, we formalized an ASP.NET MVC architecture to better support modularity, integration, and testing.

- **Incremental Feature Prioritization:**
  Some optional quality of life enhancements (i.e. full assignment reminders and extra analytics) were not prioritized in favor of delivering stable MVP components like the Grade Calculator, Attendance Tracker, and Syllabus Scraper MVP.

- **More Detailed Timeline and Cost Estimation:**
  The original plan did not include a full COCOMO II driven cost estimation. This was introduced later to satisfy later learnt course requirements and provide a more thorough project assessment.

ii.Justification for These Changes

These changes were made to ensure the project remained feasible, stable, and aligned with technical constraints:

8. API restrictions made a full eLearning integration impractical within the semester timeline, so focusing on syllabus uploads ensured the feature remained functional and realistic.

9. Formalizing the MVC architecture improved maintainability and ensured smooth integration across modules created by different team members.

10. Prioritizing core features allowed us to guarantee that all essential functionality worked reliably before adding stretch features.

11. Expanding the cost and effort analysis helped meet course expectations and provided a more accurate view of real-world development considerations.

12. References

[1] OpenAI, “ChatGPT (GPT-5.1 model),” OpenAI, San Francisco, CA, USA. Accessed: Nov. 23, 2025.

[2] “Gradeway for HAC - apps on Google Play,” Google, https://play.google.com/store/apps/details?id=com.srujan.up_grade_app&hl=en_US (accessed Nov. 23, 2025).

[3] M. N. Memon, “MyGrades Tracker App,” App Store, https://apps.apple.com/us/app/mygrades-tracker/id1662229504 (accessed Nov. 23, 2025).

[4] A. Truong, “Gradekit: Track grades & GPA App,” App Store, https://apps.apple.com/us/app/gradekit-track-grades-gpa/id947291514 (accessed Nov. 23, 2025).

[5] GradeKit, https://gradekit.github.io/#/ (accessed Nov. 23, 2025).