COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

FLORIDA ATLANTIC UNIVERSITY

PRINCIPLES OF SOFTWARE ENGINEERING

CEN4010 – Spring 2023

MILESTONE 4 – Beta Launch & Final Project Reviews

(#23) TEAM NIGHTSHIFT:

Name	Email
Adam Clark	cclark36@fau.edu
Quang Le	leq2019@fau.edu
Nicholas Moussa	nmoussa2017@fau.edu
Mahmood Sakib	msakib2022@fau.edu
Kyla Tolentino	ktolentino2021@fau.edu

Revision History	Date
n/a	*n/a*

➢ **Submitted on 04/14/23 by Quang Le**

Table of Contents

Figure 1 – Grade PlanAlyzer Logo

Product Summary

As students, we all know from personal experience that there are always moments toward the end of any given semester in which we inevitably whip out our calculators to perform a series of "doomsday" GPA number crunchings. Questions arise such as "What do I need to score on this final to pass the course?" or, hopefully, "What's the absolute *worst* I can do and still retain my 'A'?" Such questions are important in prioritizing one's time and establishing goals, especially when coinciding project and exam deadlines roll in for those already juggling multiple courses and responsibilities from family to work! "But *where* are the answers to all these pressing issues?" you might ask. Why, the how, the what, the when, and the where are all within the Grade PlanAlyzer, of course – it just needs you!

The Grade PlanAlyzer application is our solution to students' woes by creating a platform tailored to providing them a more robust, proactive, and quantitative approach to academic planning throughout each semester. Many of the most intrinsic advantages of the Grade PlanAlyzer stem from its focus upon facilitating academic performance via an account-based platform, allowing a user to rapidly access grade information between courses using a

singular click. By retaining all of one's pertinent academic information in a personal account, the student not only avoids having to manually transcribe large quantities of data on a case-by-case basis to calculate grading hypotheticals, but is also afforded the luxury of attaining more dynamic and impactful results via the application of data analytics. In other words, the student will remain keenly aware of one's academic standing throughout the entirety of a given semester such that they may make informed decisions to guide their path forward, rather than defensively access such information to backpedal a semester gone awry. The Grade PlanAlyzer will also enable students to optimize their time management skills and dedicate a more appropriate amount of time to each assigned task. Furthermore, this approach will enable them to ensure that they are allocating their efforts efficiently and effectively, ultimately resulting in better overall performance and academic outcomes.

Major Functions

- **Course Info** – the user shall be able to add/edit/delete their own course information as needed throughout a given semester
- **Look Ahead** – the user shall be able to view and sort all upcoming assignment deadlines within a designated interval alongside a chart of their relative grade impact by weight
- **Infographics** – the user shall be able to view infographics detailing their semester progress at either an overall or individual course basis
- **Grade Entry** – the user shall be able to input new (or edit existing) assignment grades as they are received
- **What-If** – the user shall be able to test hypothetical grading scenarios using slider tools which represent the average scores of ungraded assignments on a categorical basis

The What-If? tool offered by the Grade PlanAlyzer is particularly advantageous in that its calculations will always be derived from the latest empirical dataset at one's disposal. Many competitors, on the other hand, simply attempt to provide a crude letter grade estimate based solely upon input of hypothetical score averages by weighted category. Moreover, such tools are further limited due to their complete lack of customization options to account for various calculation intricacies, such as a unique grading scale, extra credit, or even complex rules such as replacing the lowest deliverable score in a category with the average of the remainder. However, the Grade PlanAlyzer maintains the real-time authenticity of such What-If? estimates by automatically adjusting its calculations to display results under the context of remaining scoring opportunities after factoring in the weight of what events have already transpired.

Grade PlanAlyzer URL

https://q-wrld97.github.io/cen4010_sp23_g23/index.html

Usability Test Plan

Test Objective

The objective of this usability test plan is to observe students performing the 'Add Course' use case of the Grade PlanAlyzer in order to identify problems in the design of the product, learn about the target user's behavior and preference, and uncover opportunities for process improvement.

Test Plan

1. System Layout: Any desktop or laptop configuration with an active internet connection

2. Intended User: College student

3. Starting Point: The user has just opened a brand new Course Form

4. URL of Test Environment:
https://q-wrld97.github.io/cen4010_sp23_q23/Dashboard/CourseInfo/html/courseForm.html

5. Task To be Accomplished:
Using a course syllabus as an external resource, the user will input the following required information pertaining a course of their choosing:

- Semester & Year
- Three-Letter Course Acronym
- Term Length
- Credit Hours
- Meeting Days
- Grading Scale
- Category Weights & Quantities
- Assignment Dates

6. Completion Criteria:

- ✓ The user has successfully submitted the Course Form.
- ✓ The Course Form layout accurately & comprehensively encapsulated all course aspects
- ✓ The UX was intuitive enough to not require any supplemental guidance.

Questionnaire

Table 1 – 'Add Course' Usability Questionnaire Form Using Likert Scale

	Strongly Agree	*Agree*	*Undecided*	*Disagree*	*Strongly Disagree*
1. The UI was visually pleasing	☐	☐	☐	☐	☐
2. The form was difficult to use	☐	☐	☐	☐	☐
3. The form was able to accommodate the unique grading features of your course	☐	☐	☐	☐	☐

Results & Suggestions

Five college students from various academic backgrounds, hereafter referred to as U1-U5, were consulted for feedback via both the aforementioned questionnaire and request for general commentary. All usability tests were conducted on the evening of April 13th, 2023, and their results have been quantitatively tabulated in *Table 2*, below.

Table 2 – Results Summary Table of 'Add Course' Usability Questionnaire

	Strongly Agree	*Agree*	*Undecided*	*Disagree*	*Strongly Disagree*
1. The UI was visually pleasing	20%	60%	-	20%	-
2. The form was difficult to use	-	40%	20%	20%	20%
3. The form was able to accommodate the unique grading features of your course	60%	20%	20%	-	-

The results of this survey indicate a strong general user approval of both the functional and aesthetic nature of the Course Form, whereas the consensus proved to be far more ambiguous regarding its ease of use. Moreover, the widespread distribution of opinions regarding the form's usability reflects a high degree of user uncertainty. Therefore, enhancing the instructional clarity of the Course Form remains a key area of opportunity for process improvement. In addition, excising the 'Term Length' and 'Class Days' field requirements may further ameliorate perceptions of superfluity and warrants further consideration. To aid in future decision-making and implementation, a few user suggestions are listed as follows:

> "In my opinion, on the new course form, it would look better if there wasn't so much empty space. So, either make everything a bit bigger or having the information split on the left and right side, if that makes sense?" – *U1*

> "The grade input screen could be modified to maybe 2 or 3 pages, that way the selections can be bigger and more fleshed out. Dashboard is very nice, maybe add an option for light mode?" – *U2*

> "Too much info. Account for human error – too many manual entries can be entered wrong and, thus, not as reliable. After submitting the Course Form it was not clear what to do next!" – *U3*

> "Include text to explain the icon functions and form sections, either by a pop-up when the mouse hovers the icon, or as a header on the page itself. (Default dark-mode, *or I riot*!!!)" – *U4*

> "The UI was pleasant, but too compact! I did not know how to use the percentage part of the form, so also I don't know how to use this for school. Maybe add instructions?" – *U5*

QA Test Plan

Test Case #: 1.1 **Test Case Name:** Add Course–Positive Case

System: Course Info **Subsystem:** Course Form

Test Objective: To check the functionality of the 'Add Course' feature such that, after entering a correct set of course information, a new course may be added to the system

Hardware & Software Setup: 1.Desktop
2. OS: Windows 11

Preconditions: 1. System is available
2. User has an active internet connection.
3. User has a permanent account and is logged into the system

Step	Action	Test Input	Expected System Response	Test Browser	Test Result
1	Navigate to Course Info Tab	Click "Course Info" icon	The system displays the Course Info tab.	Firefox 112.0	PASS
2	Add a New Course	Click "+" button	The system displays a blank Course Form.	Firefox 112.0	PASS
3	Provide Semester Term & Year	Term: Dropdown -> Spring Year: 2023	Fields should be editable and accept the information.	Firefox 112.0	PASS
4	Provide Credit Hours	Credit Hours: Dropdown -> 3 Credit	Fields should be editable and accept the information.	Firefox 112.0	PASS
5	Provide Grading Scale	Click first radial option	Fields should be editable and accept the information.	Firefox 112.0	PASS
6	Provide Weighted Categories	Check Exams, Quiz, Project	Fields should be editable and accept the information. For each checked category, three nested fields should appear: Weight%, Qty, Equally Weighted (Y/N)?	Firefox 112.0	PASS
7	Provide Category Weights	Exams: 50, Quiz: 20, Project: 30	Fields should be editable and accept the information.	Firefox 112.0	PASS
8	Provide Quantity for Each Category	Exams: 2, Quiz: 4, Project: 1	Fields should be editable and accept the information.	Firefox 112.0	PASS
9	Provide Equally Weighted Response	Exams: Dropdown -> No Quiz&Project: Dropdown -> Yes	Fields should be editable and accept the information. For the Exam category, because 'No' is selected, two more nested fields should appear: Exam1%, Exam2%	Firefox 112.0	PASS
10	Provide Weight Distribution for Each Field Generated When Weights are Not Equal	Exam1%: 20, Exam2%: 30	Fields should be editable and accept the information.	Firefox 112.0	PASS
11	Submit Course Form	Click "Submit" button	The system should perform a logic check to ensure all necessary fields have been properly addressed, all category weights sum to 100, and all sub-weights within a category add up to its designated category weight. The system should notify the user of successful form submission, then redirect user to the Deadline Form.	Firefox 112.0	PASS

Test Case #: 1.2 **Test Case Name:** Add Course–Positive Case

System: Course Info **Subsystem:** Course Form

Test Objective: To check the functionality of the 'Add Course' feature such that, after entering a correct set of course information, a new course may be added to the system

Hardware & Software Setup: 1. Laptop
2. OS: MacOS 13.0

Preconditions: 1. System is available
2. User has an active internet connection.
3. User has a permanent account and is logged into the system

Step	Action	Test Input	Expected System Response	Test Browser	Test Result
1	Navigate to Course Info Tab	Click "Course Info" icon	The system displays the Course Info tab.	Chrome 110.0	PASS
2	Add a New Course	Click "+" button	The system displays a blank Course Form.	Chrome 110.0	PASS
3	Provide Semester Term & Year	Term: Dropdown -> Spring Year: 2023	Fields should be editable and accept the information.	Chrome 110.0	PASS
4	Provide Credit Hours	Credit Hours: Dropdown -> 3 Credit	Fields should be editable and accept the information.	Chrome 110.0	PASS
5	Provide Grading Scale	Click first radial option	Fields should be editable and accept the information.	Chrome 110.0	PASS
6	Provide Weighted Categories	Check Exams, Quiz, Project	Fields should be editable and accept the information. For each checked category, three nested fields should appear: Weight%, Qty, Equally Weighted (Y/N)?	Chrome 110.0	PASS
7	Provide Category Weights	Exams: 50, Quiz: 20, Project: 30	Fields should be editable and accept the information.	Chrome 110.0	PASS
8	Provide Quantity for Each Category	Exams: 2, Quiz: 4, Project: 1	Fields should be editable and accept the information.	Chrome 110.0	PASS
9	Provide Equally Weighted Response	Exams: Dropdown -> No Quiz&Project: Dropdown -> Yes	Fields should be editable and accept the information. For the Exam category, because 'No' is selected, two more nested fields should appear: Exam1%, Exam2%	Chrome 110.0	PASS
10	Provide Weight Distribution for Each Field Generated When Weights are Not Equal	Exam1%: 20, Exam2%: 30	Fields should be editable and accept the information.	Chrome 110.0	PASS
11	Submit Course Form	Click "Submit" button	The system should perform a logic check to ensure all necessary fields have been properly addressed, all category weights sum to 100, and all sub-weights within a category add up to its designated category weight. The system should notify the user of successful form submission, then redirect user to the Deadline Form.	Chrome 110.0	PASS

Code Review

The team's chosen coding style adheres to the Kernighan & Ritchie (K&R) format. The sample segment of code below was submitted for peer review and depicts the account deletion function employed by the Node.js server should a user fail to verify within a 24-hour window.

```
const deleteUser = cron.schedule('0 * * * *',  () => {

  // Get all users from Firebase auth
  auth.listUsers().then((listUsersResult) => {
    listUsersResult.users.forEach((userRecord) => {
      const userUid = userRecord.uid;
      const userCreationTime = userRecord.metadata.creationTime; // creation timestamp in milliseconds

      // Check if the user is not email-verified
      if (!userRecord.emailVerified) {
        const timeSinceCreation = Date.now() - new Date(userCreationTime).getTime(); // elapsed time in milliseconds
        const timeThreshold = 24 * 60 * 60 * 1000; // 24 hours in milliseconds

        // Check if the user account is older than 24 hours and not verified
        if (timeSinceCreation >= timeThreshold) {
          // Delete the user account from Firebase auth
          auth.deleteUser(userUid)
            .then(() => {
              console.log(`Successfully deleted user ${userUid} from Firebase auth.`);
            })
            .catch((error) => {
              console.log(`Error deleting user ${userUid} from Firebase auth: ${error}`);
            });

          // Delete the user data from Firestore
          db.collection('users').doc(userUid).delete()
            .then(() => {
              console.log(`Successfully deleted user data for ${userUid} from Firestore.`);
            })
            .catch((error) => {
              console.log(`Error deleting user data for ${userUid} from Firestore: ${error}`);
            });
        }
      }
    });
  });
});
```

Figure 2 – Sample Code Function 'deleteUser' Written in JavaScript

Peer Review

1. **Readability**: code is easy to read and understand, and maintains proper indentation, bracketing, and spacing between blocks of code and components therein
2. **Naming conventions**: consistent use of camelCase for variable names, typically confined to 7-20 characters in length for excellent readability
3. **Descriptive variable names**: variables have meaningful and intuitive names, such as "userUid", "userCreationTime", "timeSinceCreation" and "timeThreshold", which helps in understanding the purpose of each variable

4. **Proper error handling**: uses ".catch()" statements to handle potential errors when interacting with Firebase; this further assists with debugging in the console
5. **Use of promises:** uses promises to handle asynchronous operation when writing to and reading from firebase to provide more stability.
6. **Functional approach**: the example code is compartmentalized as a single function, 'deleteUser', which is responsible for the entire process of identifying and deleting unverified user accounts; this modular approach makes the code reusable and easier to maintain
7. **Separation of concerns**: the code separates the tasks of deleting the user account from Firebase authentication and deleting the user data from Firestore; this separation of concerns allows for better organization and easier debugging in case issues
8. **Clear control flow**: the code uses nested 'if' statements to check the condition for deleting a user account, allowing for clear and logical control flow; the nested structure aids in understanding the sequence of events which must take place for an account to be eligible for deletion

Overall, the code is exceptionally well-organized! The primary means of improvement would be to prepend all functions with additional header commentary detailing its name, preconditions, postconditions, and description. An example is shown below:

```
/*
//*********************************************************************************************
Name: deleteUser

Precondition: User has created an account and 24 hrs have passed without confirmation

Postcondition: If an account remains unverified over 24 hours since account creation, delete the
    user account from the database.  Otherwise, no further action is required.

Description: Every hour, look for users that have registered but have not verified their accounts.
    If it has been over 24 hours since account creation, delete the user account and database.
    We first iterate through all user ID and Time of Creation(TOC) from the authentication in
    Firebase. Then, for each user, we check using .emailVerified method to verify whether the
    given user is currently not verified. Prior to comparison, we calculate TimeSinceCreation
    as the difference between the current time and TOC. Additionally, we declare a variable
    called timeThreshold that holds the 24-hour threshold. If TimeSinceCreation > timeThreshold,
    we proceed with the "if" statement to delete the user in authentication and Firestore.
//*********************************************************************************************
*/
```

Figure 3 – Suggested Header Commentary for 'deleteUser' Function

Self-Check: Best Practices for Security

Major Protected Assets

As part of our commitment to protecting user data, our "grade plan analyzer" application implements robust security measures to safeguard the major assets we handle. Firstly, we collect only minimal personally identifiable information (PII), limited to usernames, and in case of a potential attack, only user email addresses may be at risk. This aligns with our commitment to minimizing the use of PII and ensuring compliance with data privacy regulations. Furthermore, our Firestore database is configured to associate each class and grade information with a unique user ID, ensuring that Insecure Direct Object Reference (IDOR) attacks and user manipulation attempts are mitigated. This granular access control ensures that only authorized users can access and modify their own data. In addition, our application leverages the robust security features of Firestore, including protection against SQL injection (SQLi) attacks. This ensures the integrity and confidentiality of the stored data, preventing unauthorized extraction without proper authorization. Moreover, our application implements browser-side checks to verify that the user's unique ID matches the data retrieved from the Firestore database, adding an extra layer of security to detect and prevent unauthorized access attempts. By implementing these security measures, we are dedicated to safeguarding the confidentiality, integrity, and availability of user data, ensuring the highest level of security for our application, and minimizing the potential risk to user PII in case of any security breach. This underscores our commitment to data privacy and protection for our users and stakeholders.

Database Password Encryption

As part of our robust security measures, all user passwords are encrypted using a strong encryption algorithm script, with a unique salt for each user, before being stored in the authentication database. For example, when a user creates an account or changes their password, the password is passed through the encryption algorithm with the salt, and the resulting encrypted password is then stored in the database. Additionally, we prioritize the secure transmission of passwords over the internet. When users enter their passwords, the passwords are sent over the internet using Secure Socket Layer (SSL) or Transport Layer Security (TLS) encryption protocols, which encrypt the data during transmission to prevent interception and eavesdropping by unauthorized parties. This ensures that passwords are securely transmitted from the user's device to our servers, protecting them from potential interception and unauthorized access. Here's an example of how a password might be stored in and transmitted securely over the internet using SSL:

Original Password: "P@55w0!d12345"

Salt: "&Rj0$gFG"

Encrypted Password:
"$2a$12$R#7^b2@9DpfI/BoydFyO1e9kbSdN7L4blO1NToaMw4M4t5Cjlrju"

By implementing strong encryption algorithms with unique salts for password storage and ensuring secure transmission over the internet using SSL or TLS, we prioritize the security and protection of user credentials, minimizing the risk of unauthorized access and maintaining the confidentiality of user data.

Input Data Validation

In our application, since we do not employ a search bar, there is no need to perform validation for search bar input as it is not applicable to our application's functionality. It's essential to always tailor input data validation to the specific requirements and inputs used within a given application to ensure data integrity and security. A few examples of data validation are illustrated below.

```
// Validate that email is in the correct format
function validateEmail(email) {
  expression = /^[^@]+@\w+(\.\w+)+\w$/
  if (expression.test(email) == true) { // email is valid
    return true
  } else { // email is invalid
    return false
  }
}
// Validate password is in the correct format: (1 number, 1 special, 6 characters, 1 uppercase, 1 lowercase)
function validatePassword(password) {
  var re = /^(?=.*[0-9])(?=.*[!@#$%^&*])[a-zA-Z0-9!@#$%^&*]{6,}$/;;
  if (re.test(password)==true) { // password is valid
    return true
  } else { // password is invalid
    return false
  }
}
```

Figure 4 – Email and Password Input Validation Code

Shown in *Figure 4* above, two examples of input data validation being performed include validation for email format and password format.

I. **Email format validation:** The function 'validateEmail(email)' uses a regular expression (/^[^@]+@\w+(\.\w+)+\w$/) to validate whether the email input follows the correct format. If the email format is valid, the function returns true; otherwise, it returns false.

II. **Password format validation:** The function 'validatePassword(password)' uses a regular expression (/^(?=.*[0-9])(?=.*[!@#$%^&*])[a-zA-Z0-9!@#$%^&*]{6,}$/) to validate whether the password input meets certain criteria, including having at least one number, one special character, one uppercase letter, one lowercase letter, and a minimum length of six characters. If the password format is valid, the function returns true; otherwise, it returns false.

Self-Check: Adherence to Original Non-Functional Specs

Interoperability Requirements

Browser Compatibility – The system will be a we-based web app that operates on major browsers, including Google Chrome, Mozilla Firefox, Safari, Opera, Brave, and Internet Explorer.

➔ **DONE**

Operating System Compatibility – The system will operate on multiple operating systems, including Windows, Linux, and OS X.

➔ **DONE**

Mobile Operating Sytem Compatibility – The system will operate on all major mobile operating systems, including iOS and Android.

➔ **ISSUE** – Potential time constraints exist with respect to fully polishing the mobile-specific UI as it is a low-priority action item; however, the website itself will still be useable via mobile device.

Storage Requirements:

Back-End Maximum Database Load:

- *Storage Capacity* – 1 GiB
- *Document Reads* – 50,000 per day
- *Document Writes* – 20,000 per day
- *Document Deletes* – 20,000 per day
- *Network Egress* – 10 GiB per month

➔ **DONE**

Back-End Usage Load for Authentication and Authorization:

- *Daily Active Users* – 3000 per day
- *SMS Sent* – 10 per day per user
- *Multi-Factor Authentications* – 10 per day per user

➔ **DONE**

Security Requirements

Our user authentication/authorization system will maintain the private integrity of site users' academic information and preferences on an individual basis.

Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to any app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more. The Firebase Authentication software development kit (SDK) provides methods to create and manage users that use their email addresses and passwords to sign in. Firebase Authentication also handles sending password reset emails.

Firebase Security Rules leverage extensible, flexible configuration languages to define what data users can access for Realtime Database, Cloud Firestore, and Cloud Storage. Firebase Realtime Database Rules leverage JSON in rule definitions, while Cloud Firestore Security Rules and Firebase Security Rules for Cloud Storage leverage a unique language built to accommodate more complex rules-specific structures. Firebase provides a rule-based access control system that allows developers to enforce fine-grained access control to data stored in Firebase services and restrict access to sensitive data to ensure that only authorized users can access.

➔ **DONE**

Supportability Requirements

Coding Standards – Our system will be coded in a range of 75-80% of coding standards for HTML 5 and CSS3. The code will be routinely reviewed, tested, and validated via third-party compliance software.

➔ **ON TRACK**

Naming Conventions – HTML classes and id tags will be coded in lowercase except in naming situations involving multiple words, for which the camelCase convention will be applied. Firebase collections, documents, and fields will also adhere to this convention.

➔ **ON TRACK**

Availability Requirements

Accessible Times – Our system should be available for use 24 hours per day, 7 days per week, unless catastrophic unforeseen circumstances arise due to our application being hosted upon GitHub.

➔ **DONE**

Downtime Impact – General downtime will be largely non-existent, save for brief events in which our team updates the software, which should only result in about 5 to 10 minutes of unavailability.

➔ **DONE**

Report Issue – There will be a support feature accessible via the navigation bar which will pull up a template that allows the user to express concerns/report bugs which will automatically be routed to the team's email account.

➔ **ON TRACK**