Software Engineering
CSC 131
Section 2
Fall 2023
Software Requirements and Design Brief
Dynamic PDF Generation for E-commerce documents

Table of Contents:

History Revision Table:

Revision #	Date	Reviser(s)	Description
1	10/2	Bilal	Added Table of contents
2	10/3	Bilal	System context diagram and description
3	11/1	Dylan	Functional/ Non-Functional requirements.
4	11/4	Ayush	Added Sequence diagram.
5	11/13	Dylan	Added ERD Diagram.
6	12/10	Bilal	Added Use Case Model Revised Functional/ Non-Functional requirements
7	12/11	anthony	Revised and added technology and tools
8	12/13	Dylan	Added description to ERD.
9	12/13	Evan	Revised the code files for the terminal and index files
10	12/13	Ayush	Added completed 4+1 view model for architectural design
11			
13			
14			
15			
16			
17			
18			
19			

Software Engineering Team

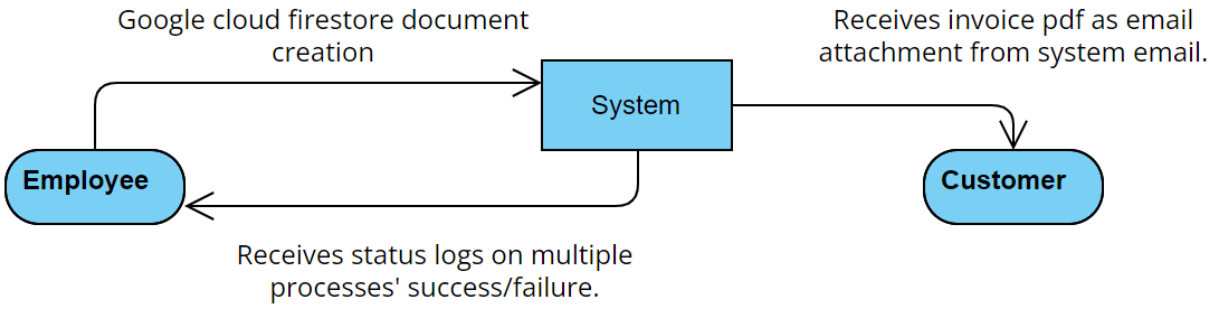
Project Leader: Ayush Thapa

Team Members: Bilal Baloch, Dylan Khon, Anthony Torres, Evan Selby

Project Scope- System Context Diagram (SCD) with brief description

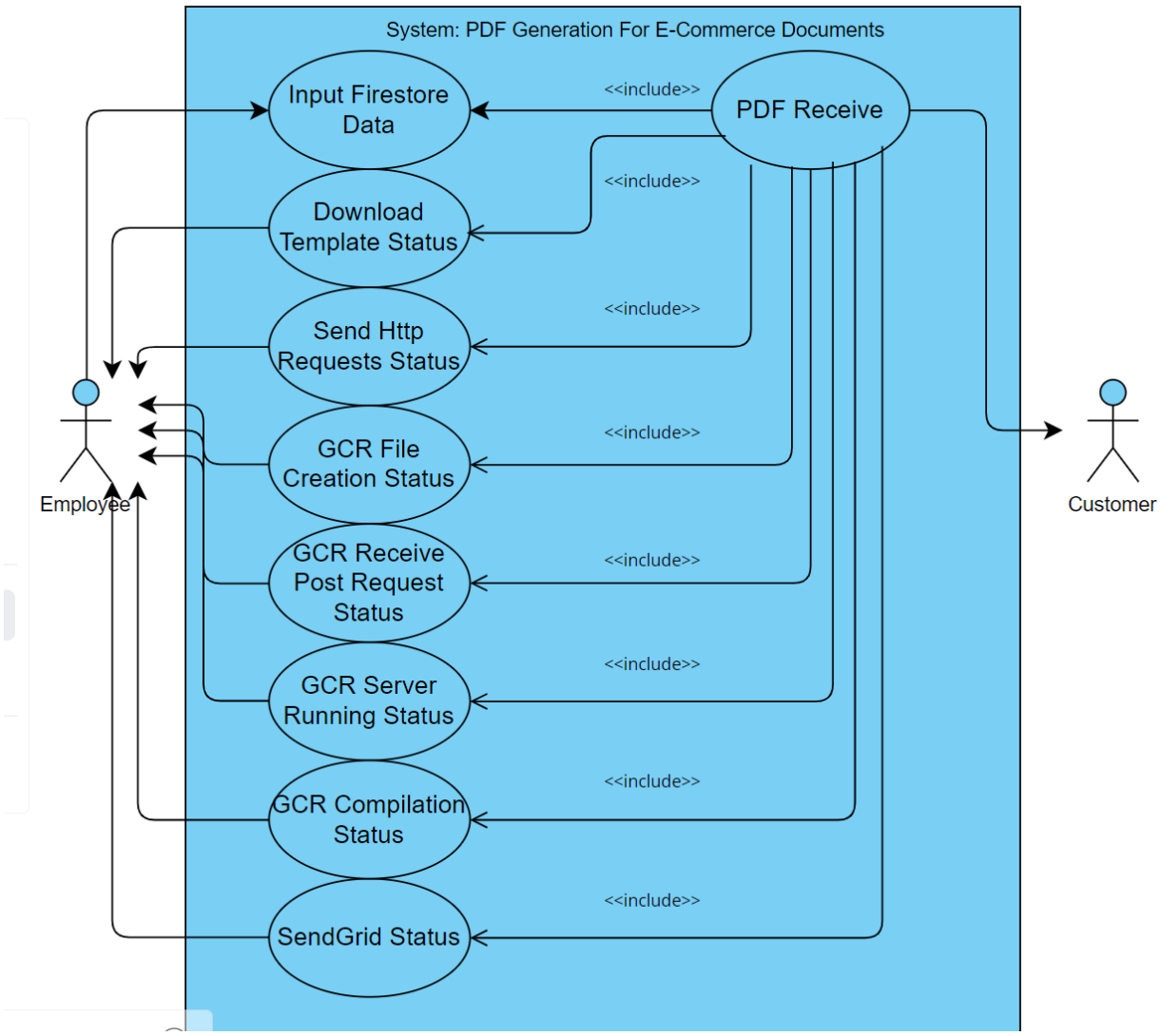
Description:

This System Context Diagram establishes the information boundary between the system being implemented and the environment in which the system operates. The system here is labeled as “Dynamic PDF Generation for E-commerce Documents System. The environment in which the system operates is the Google Cloud Platform, which utilizes google cloud storage, firestore, cloud run and cloud functions. The information entry point is firestore, and exit is a pdf attachment sent in an email. The external producers of information are employees who input customer information into google cloud firestore, which triggers the system. The consumer of the information is the employee who receives status logs on multiple processes’ success/failures, and the customer, who receives an invoice pdf in an email attachment. The external entities that interact with the system are employees, and customers of async.



Object Oriented Requirements Analysis (OOA)- UML Modeling

Use Case Diagram:



Use Cases/Supplementary Specification/Glossary

Link for the glossary
https://docs.google.com/presentation/d/1tdcHxOWCjYsbGzSEOHPgr9_3iGlb1tuUfQLyfyxC-SE/edit#slide=id.p

System Requirements

Functional:

FR1 - User inputs data into the firestore database.
FR2 - User enters an empty template to the Google cloud storage.
FR3 - Obtain a blank template from Google Cloud Storage.
FR4 - Query data from Google Firestore into a .tex format.
FR5 - Fill in data from firestore into the blank template.
FR6 - Generate the filled template with text wrapping and adjustments.
FR7 - Save completed template in Google Cloud Storage.
FR8 - Use SendGrid to email the PDF along with a confirmation message.
FR9 - Update the flag in Google Firestore following email.

Non-Functional:

Error-Handling

Detects errors upon creation and notifies if the function was successful.

Reliability

Flexibility

Handles other forms of template population other than an invoice receipt.

Dynamic

The template handles text-wrapping in the case the item name exceeds that of the row size, continuing the name below.

Efficiency

PDF is generated in a timely manner once the function is triggered.

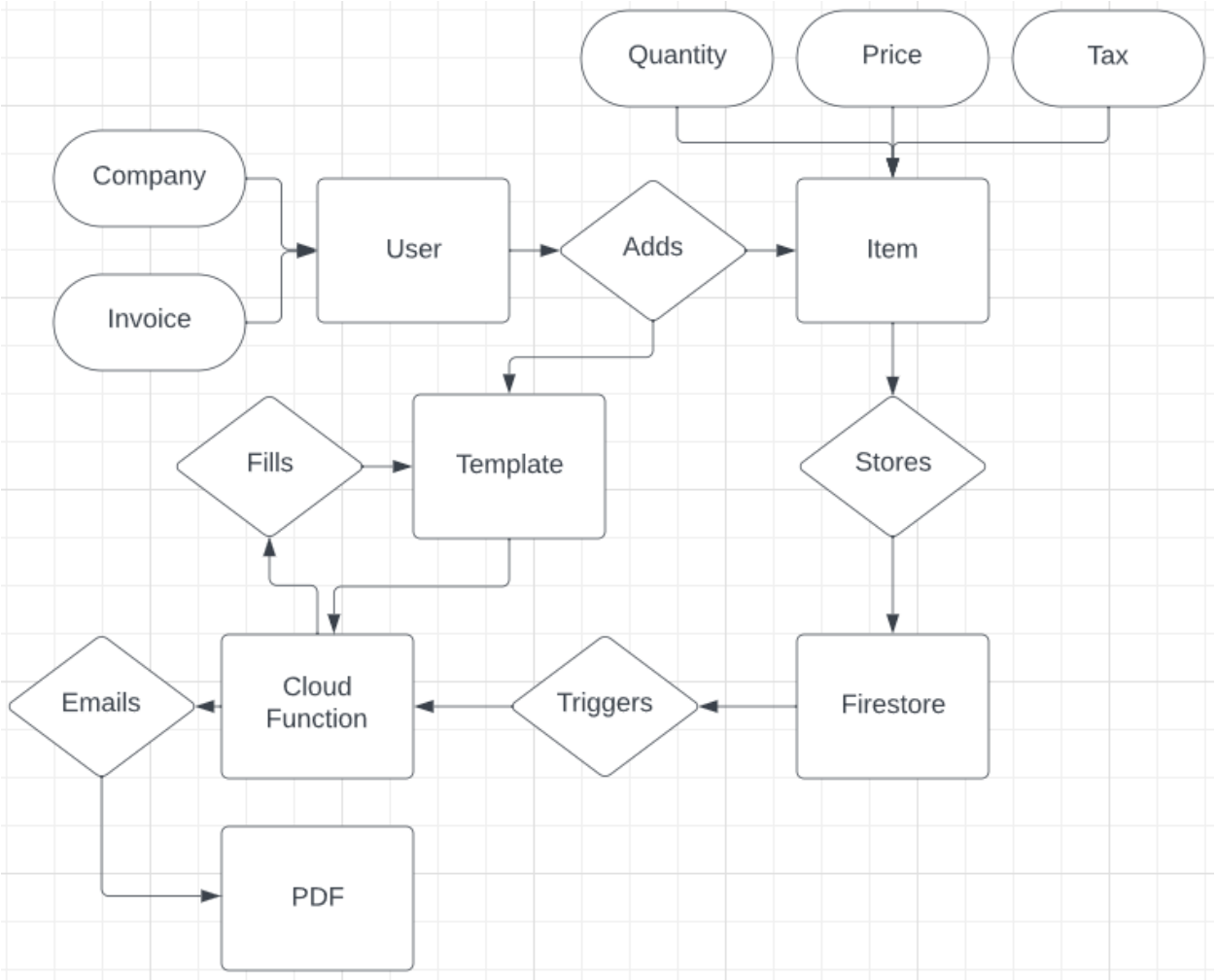
Quality

Code functions are provided with a detailed description and are well organized. Additionally generates a completed and accurate template.

Data Design

The Entity Relationship Diagram depicts the various entities found in the system and the relationship between them. Each entity may have attributes associated with them, in the case of our project the item entered into the firestore database holds numerical attributes such as the amount or its price. The user interacts with the system by adding each item and their choice of a blank template which would be returned as a completed template via email. The entities in this system ultimately interact with the Google Cloud functions, running with Node.js in order to fill out the blank template into a completed PDF and sending it with the email through SendGrid.

Entity Relationship Diagram (ERD)



Architectural Design

The 4+1 view model is our special guide for explaining how the PDFGen Commerce Suite (our system name) is put together. It's a helpful structure made up of four main views, each showing a different side of how our system works. Plus, there's an extra view that connects everything by using examples or situations we might come across. This way, it's not just technical talk – it helps us tell a complete story about our system.

1. Logical View:
Description:
The Logical View zeros in on the main concepts in the system, like objects or object classes. It's all about how the software is structured in terms of big-picture arrangements.

Elements:

Objects or object classes for entities like Items, Templates, Users.
Relationships between these objects/classes.
Key abstractions and interfaces.
Focus:
High-level organization of software entities.

(add a diagram for each view. Include the elements in the diagram to create it)

Purpose:
To define the fundamental building blocks and their relationships, providing a structured view of the system.

Relation to Architecture:
This view captures the foundational components and their interactions, forming the basis for the overall system structure.

2. Process View:
Description:
The Process View paints a picture of how the system behaves when it's running – think of it as a dynamic snapshot. It shows the moving parts, like how different processes talk to each other.

Elements:

Node.js Microservices: Each one is like a small worker handling specific jobs.
Cloud Functions: These are like tasks that run for specific processes, especially for PDF creation.
Cloud Run: It hosts the Node.js Microservices.
Docker: It's the worker bee, handling processes related to PDF creation.
Communication channels between processes.
Focus:
Dynamic behavior of the system at runtime.

Purpose:
To illustrate how different processes interact and communicate during system execution.

Relation to Architecture:
This view captures the runtime dynamics, showing how processes collaborate and ensuring efficient system execution.

3. Development View:
Description:
The Development View gives developers a peek into how the software is broken down for building and maintaining. It's like an inside look at how different pieces come together.

Elements:

Source code organization in Visual Studio Code.
Module breakdown: How different modules or microservices are structured.
Dependencies between modules.
Development tools and environments.
Focus:
Software organization for development and maintenance.

Purpose:
To demonstrate the modular structure, dependencies, and development tools, aiding developers in efficient coding.

Relation to Architecture:
This view provides insights into how the system is organized for development, influencing coding practices and maintenance strategies.

4. Physical View:
Description:
The Physical View shows the nuts and bolts – the hardware and how software components are spread out across processors. It's all about where things live in the real world.

Elements:

Google Cloud Platform resources: Cloud Functions, Cloud Run, Cloud Storage, Artifact Registry.
Distribution of software components across physical hardware.
Deployment nodes and servers.
Communication paths between hardware elements.
Focus:
System hardware and deployment.

Purpose:
To address deployment and scalability considerations, showcasing the physical setup of the system.

Relation to Architecture:
This view helps make decisions about where software components live, ensuring scalability and efficient resource allocation.

+1: Related using Use Cases or Scenarios:
Description:
This view ties everything together using real-world examples or situations. It's the bridge that connects technical views to what users will actually experience.

Elements:

Use cases or scenarios showcasing user interactions.
Traceability between use cases and logical, process, development, and physical views.
Illustration of how different views contribute to fulfilling specific user needs.
Focus:
User interactions and system functionality.

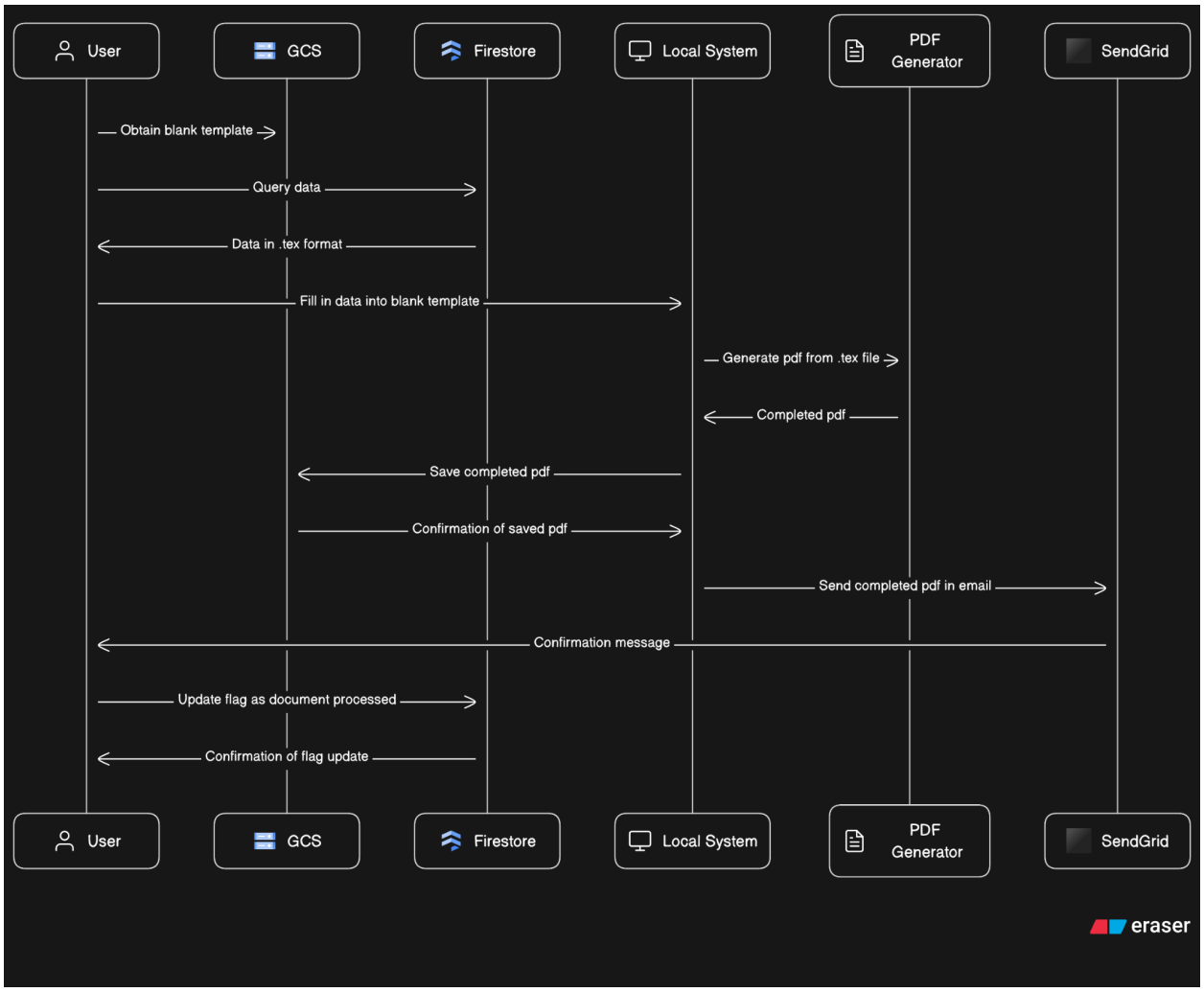
Purpose:

To provide a real-world context for technical views, ensuring alignment with user expectations.

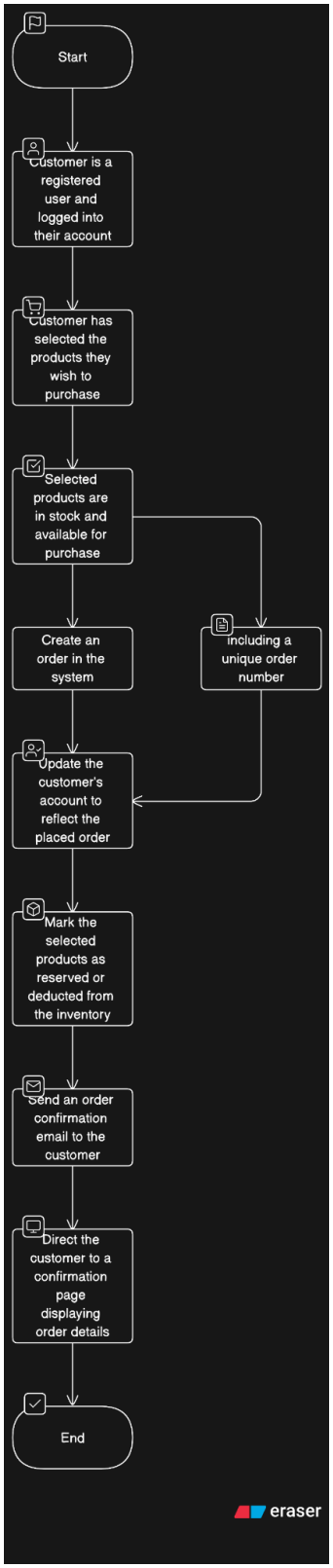
Relation to Architecture:
This view connects technical details with user experiences, ensuring that the architecture caters to real user needs and scenarios.

Detailed Design

Sequence Diagram



User Interface Design



Technology and Tools

Introduction:

In this section, we will detail the technology stack and tools that will play a crucial role in the development of our project. These choices are a result of careful consideration and align with our project's specific goals and requirements.

Front End:

The application we constructed is a backend one, and as such it does not include any front end elements.

Back End:

language/runtime/framework: javascript, node.js, express

The application is written in javascript as it is a great language for web development. Node.js will be used as it is a runtime environment that allows for javascript execution outside of a web browser, for instance on a windows machine in VS code. Express will be used as the framework in docker, as it is a light and flexible node.js web Framework that provides a good set of features for web applications.

Server environment: Google cloud platform

We have chosen Google Cloud platform as our hosting provider for its reliability and scalability, ensuring that our back-end services are available to users without interruption. The Google cloud platform provides several essential resources/solutions needed to get the application up and running. These include cloud functions, cloud run, cloud storage, artifact registry, and firestore.

Database Management System:

Our data will be stored in Google Cloud Firestore, which is a scalable NoSQL database that is well-suited to handling structured data.

Development environment: VS code and cloud functions

The majority of the development will take place in VS code with occasional editing within cloud functions. The reason we chose VS code is because it is optimized for building and debugging web and cloud applications.

Tools/software: docker, texlive,pdflatex

In order to achieve certain parts of the application some tools/software are required. To solve the problem of pdf creation, we used docker along with VS code to create a docker image that contains instructions on how to handle tex to pdf. We chose docker because of its abilities to build, share, and run container applications. Within this image is texlive and pdflatex, these are required for the tex to pdf conversion.

Dependencies:

In order for certain aspects of the code to run, send, and or receive information the certain dependencies are needed. On the could function side, @google-cloud/firestore,@google-cloud/functions-framework, @google-cloud/storage, @sendgrid/mail,

child_proces, util, pdflatex, node-modules, debug, os, fs,axios, and firestore. On the docker side, @google-cloud/storage,child_proces, util, until, fs,and express

APIs: sendgrid, google cloud APIs

Sendgrid will be used as our emailing service for the distribution of the generated document. Google APIs that we will use include compute engine, cloud functions, cloud monitoring, cloud logging, cloud run admin, artifacts registry, cloud DNS, cloud firestore, cloud build, cloud auto scaling. Each of these are needed to ensure proper communication and functionality.

Version control:

We will utilize Visual Studio Code and google cloud platform for version control. Visual Studio Code will contain versions of code that will then be pushed to the google cloud platform which has shared access, so all members can view the code and different iterations.

Collaboration

Discord will be used for real-time communication, ensuring smooth teamwork and efficient progress tracking.

In conclusion, our technology stack and tool choices have been thoughtfully made to ensure that they align with the specific needs and goals of our project. Each component plays a crucial role in our development process, and we are committed to maintaining and updating these tools as our project evolves.

Project Assumptions and Constraints

Assumptions:

Assumption 1: Hardware Availability

Description: We assume that the necessary hardware resources will be readily available for our project.
Rationale: This assumption is based on our enrollment in CSUS and in course CSC131.

Assumption 2: User Training

Description: We assume that user training will not be necessary for using our user-friendly software.
Rationale: User feedback and the intuitive design of our software support this assumption.

Assumption 3: Project Scope Stability

Description: We assume that there will be no significant changes in the project scope.
Rationale: This assumption is made after thorough project scope definition and client requirements analysis.

Constraints:

Constraint 1: Budget Limitations

Description: Budget constraints restrict our ability to acquire additional software licenses.
Impact: This constraint may affect our short-term scalability for accommodating more users.

Constraint 2: Regulatory Compliance

Description: Regulatory constraints require strict adherence to industry standards.
Impact: This constraint mandates rigorous testing and documentation to ensure compliance with industry regulations.

Constraint 3: Fixed Project Deadline

Description: The project is under a time constraint with a fixed deadline.
Impact: This fixed deadline necessitates meticulous project scheduling and a well-defined project plan.

Mitigation Strategies: Regular compliance checks and documentation reviews will ensure adherence to industry standards. Project scheduling will include buffer time to accommodate unexpected delays.

Conclusion:
By documenting our assumptions and constraints, we can make informed decisions, manage risks effectively, and ensure the successful delivery of our project.