



IntelliQuiz

Project Report

CPSC 2350 - Software Practices

Instructor: Parsa Rajabi

Langara College

Authors:

Harpreet Singh

Miguel Fierro

Patrick Fang

Utsav Monga

[Github](#)

Table of Contents

Table of Contents.....	2
Overview.....	3
Accomplishments.....	3
Overview of SDLC Model:.....	4
Changes to Project Plan:.....	4
Features & User Stories.....	5
API Features.....	6
Testing Description.....	9
CI/CD infrastructure Description.....	10
Data Flow Diagram.....	11
Challenges & Takeaways.....	12
Work Division.....	13

Overview

IntelliQuiz is an AI-powered quiz maker web application, designed for students who wish to streamline study sessions and instructors who want to simplify quiz creation for their classes.

The app is able to take text from the user interface or PDF documents as input. The type and number of questions are set by the user, and can be downloaded as a lockable PDF or attempted from within the application. Quiz questions are presented one at a time during attempts, each question has a “Hint” option in case the user does not understand what is being asked. Finally, in addition to the attempt score, an explanation for wrong answers is provided as well as performance feedback at the end of the quiz pointing toward topics that need more review. This final summary can also be downloaded as a report PDF.

Accomplishments

- Developed and deployed a functional web application using React and Tailwind CSS.
- Followed the Kanban board methodology as our SDLC.
- Integrated OpenAI and AdobePDF APIs to provide six core features:
 - OpenAI:
 - Generates quiz questions based on user-provided text input.
 - Offers hints and explanations for challenging questions.
 - Provides feedback on quiz performance with areas for improvement.
 - AdobePDF:
 - Extracts text from uploaded PDFs for quiz creation.
 - Generates downloadable PDFs with quiz questions and answers, and attempt performance reports.
 - Offers password protection option to quiz PDFs.
- Built a reliable CI/CD pipeline with GitHub actions to:
 - Run Vitest on open pull requests and merges to the main branch
 - Build and deploy the site on merges to the main branch.

Overview of SDLC Model:

The team decided to use the Kanban SDLC framework because of its flexibility and ease of use within the GitHub interface, which helped manage workflow and reduce bottlenecks. The number of tasks in progress are limited to ensure that each task is given proper attention and doesn't overwhelm the team. Additionally, the Kanban board made it easy for the team to see what features were being worked on by who, and what was left to do in the project.

- **What Worked:** Kanban facilitated task management and provided a clear view of progress.
- **What Didn't Work:**
 - It was difficult for the team to adapt to the Kanban model, for the first half of the project we neglected it while we coded and quickly ran into conflicts.
 - We never set a standard for naming issues, naming branches, adding tags, commit messages, project milestones, etc. Which made our Kanban board feel disjointed.

Changes to Project Plan:

- **Main Branch Protection:** Switched from pushing directly to main, to an Issue-Pull Request system with reviewers.
- **PDF API:** The team chose ILovePDF as our initial PDF handling API. However, this API does not offer a data-driven PDF generation tool. It instead relies on public URLs to turn HTML to PDFs, after many failed tries getting this feature to work, the decision was made to use AdobePDF instead, as we could use only the data from our application to generate PDF files. This decision came at the cost of complexity and speed as every feature requires multiple HTTP requests to perform.
- **Using the GPT-4 engine:** At first, the GPT-3.5-turbo engine was used for every OpenAI API call, because of the lower cost and general effectiveness of the answers. However, we realized that the engine sometimes provides unreliable results, for example: Generating less questions than requested, revealing the answer on the hint text, generating the wrong type of questions, not returning the questions with the JSON format, setting a wrong answer as

correct. Tests with the GPT-4 engine were not perfect, but more effective in solving these issues.

- **Testing Framework:** Initially, Jest was considered for testing due to its wide use on the React ecosystem. However, because the app's build tool is Vite, the team switched to Vitest for its ease of use.
- **Deployment platform:** The team intended to use the Render platform to deploy the site, however, since we were already using GitHub Actions to perform automatic testing, and Vite applications can use a workflow file to deploy, we decided to deploy to GitHub Pages instead.

Features & User Stories

OpenAI

1. **Generate Quiz Questions:** Users can create quizzes from text or PDFs.
 - *"As a student, I want a tool that tests me on a topic I am reviewing"*
 - *"As a teacher, I want a platform that helps me design tests on a topic I am teaching my class."*
2. **Provide Hints and Explanations:** Users can get hints for challenging questions and Explanations for wrong attempts.
 - *"As a quiz taker, I need the app to nudge me in the right direction if I don't understand a question."*
 - *"As a quiz taker, I want to understand why the answer I gave was wrong."*
3. **Analyze Quiz Performance:** Students will receive feedback on quiz attempts for improvement.
 - *"As a student, I would like to receive constructive feedback to improve my quiz score."*
 - *"As a user, I would like to know what are my weak spots on a topic to review them further."*

AdobePDF

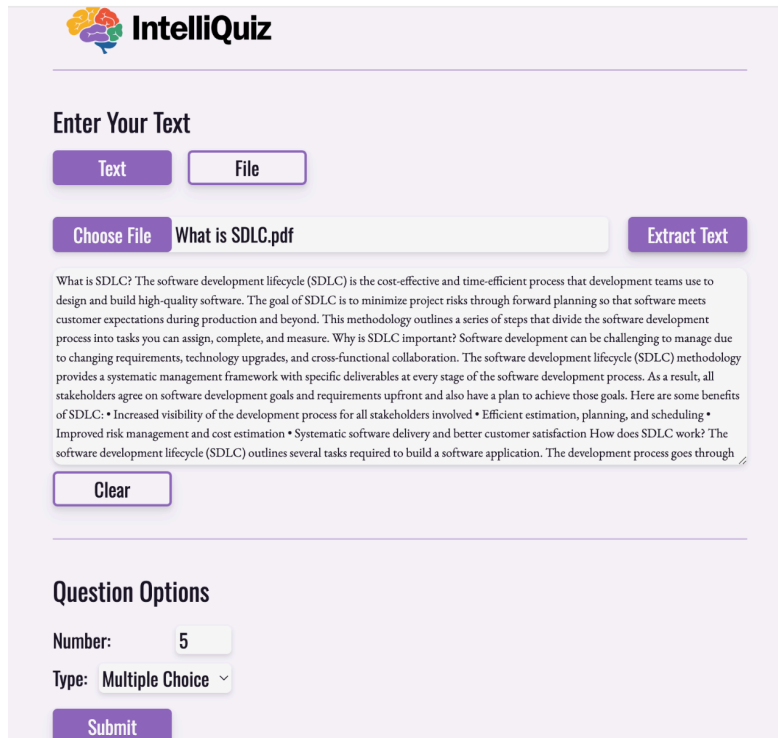
1. **Extract Text from PDFs:** Users can create quizzes from existing PDFs.
 - *"As a student, I would like to quiz myself on the content of the slides from my previous lecture."*
2. **Generate Downloadable PDFs:** Users can download PDFs with quiz questions (before attempt) and summaries/feedback (after attempt).

- *“As a teacher, I would like to keep a copy of the quiz I gave to my students for archiving.”*
 - *“As a teacher, I want to use a paper quiz for my class to minimize cheating”*
 - *“As a user, I want to keep a copy of my performance report to keep track of my progress in the future.”*
3. Password-Protect Quiz PDFs: Teachers can protect quizzes with password-protected PDFs.
- *“As a teacher, I want to protect unreleased quizzes in my file system.”*

API Features

OpenAI


- **Generate Quiz Questions:** The application uses text input from either the user or extracted from a PDF file. We designed a prompt for OpenAI API to analyze the content, and return questions based on the app settings as a JSON array, we then use this array to display it as a quiz in our application.



The screenshot shows the IntelliQuiz application interface. At the top is the IntelliQuiz logo. Below it is a section titled "Enter Your Text" with two buttons: "Text" and "File". Underneath these is a "Choose File" button, a text input field containing "What is SDLC.pdf", and an "Extract Text" button. Below the input field is a preview of the text from the PDF, which discusses the Software Development Lifecycle (SDLC). At the bottom of this section is a "Clear" button. Below the text preview is a section titled "Question Options" with a "Number:" label and a text input field containing "5", and a "Type:" label with a dropdown menu set to "Multiple Choice". At the bottom of this section is a "Submit" button.

- **Provide Hints and Explanations:** If a user finds a question challenging, they can click a "Hint" button, provided under each question in our app. The OpenAI API provides a clue

without revealing the answer. Additionally, when the quiz is submitted, wrong answers are provided with a detailed explanation.


IntelliQuiz

5. What tasks are involved in the maintenance phase of SDLC?

Designing new features

Bug fixes and customer support

Initial coding

Environment configuration

Click here for a hint!

Back

Submit


IntelliQuiz

Designing new features

Bug fixes and customer support

Initial coding

Environment configuration

Addressing software issues post-deployment.

Back

Submit

- **Analyze Quiz Performance:** After completing a quiz, we send the user's answers to the OpenAI API to provide personalized feedback highlighting strengths and weaknesses.

Combining automation and manual testing

Creating schedules

Allocating resources

Coding the product

The purpose of the testing phase in SDLC is to combine automation and manual testing to check the software for bugs and ensure it meets customer requirements.

5. Why is the deployment phase important in SDLC?

Coding the product

Resolving customer issues

Moving latest build copy to production environment

Analyzing requirements

The deployment phase is crucial in SDLC as it involves moving the latest software build to the production environment for customers to use.

Feedback

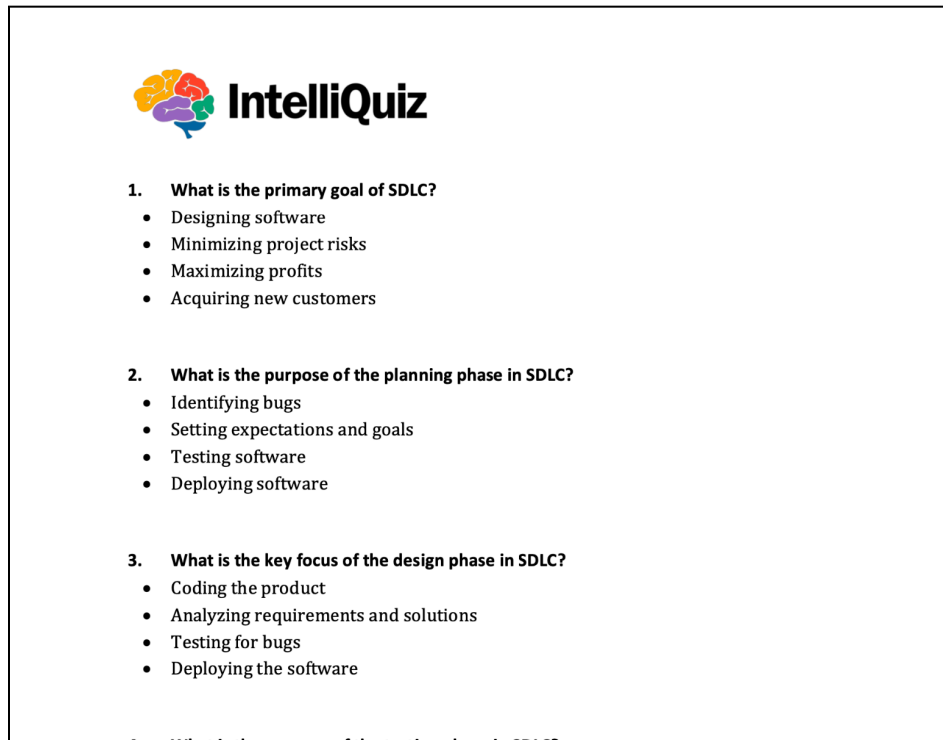
Great job on getting two questions right! It shows that you have a good understanding of those concepts. However, there were three questions that you got wrong, indicating that there may be some areas that need further review. Make sure to go over those questions and understand the correct answers. Keep up the good work on the questions you answered correctly, and continue to practice and study the ones you struggled with. With more effort and practice, you will be able to improve your performance. Keep it up!

Home Page

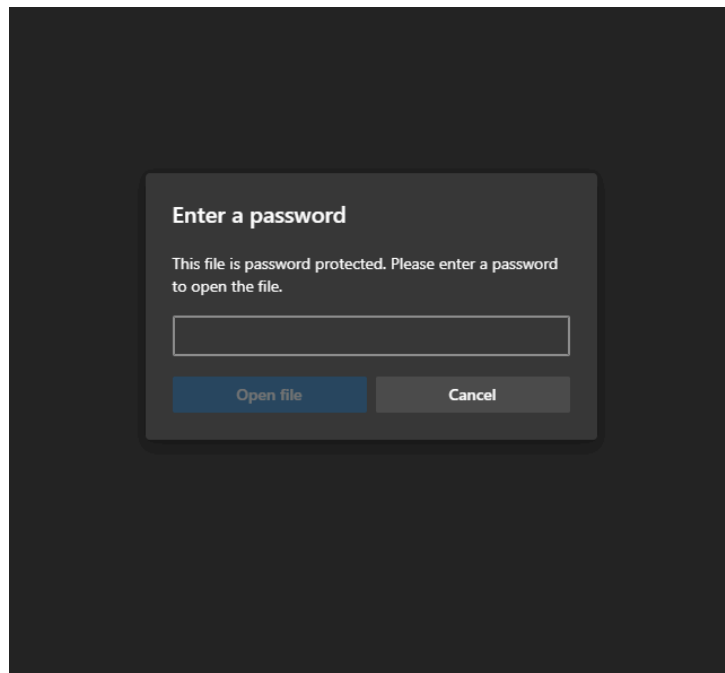
Download Report

AdobePDF

- **Extract Text from PDFs:** Users can upload a PDF file. The AdobePDF API extracts the text content and returns it to the app, to be used as the text input for OpenAI.
- **Generate Downloadable PDFs:** Users can generate PDFs containing the quiz questions and answers returned from OpenAI. After completing the quiz in the app, a PDF report can be generated with the attempt summary and feedback provided by the OpenAI API.



- **Password-Protect Downloadable PDFs:** A user can choose to password-protect the quiz PDFs containing the quiz questions. This feature was mainly made for instructors who wish to protect unreleased quiz files on their computer and avoid document sharing.



Testing Description

Testing was performed using the Vitest testing framework because it is compatible with the Jest syntax and integrates well with the Vite development server. Both the unit and integration test suites can be run locally using **npm run test**.

Unit Testing

Our unit testing approach aimed to verify the reliability and functionality of our API calls. We simulated Axios calls using mocking and helper functions to test the features in isolation. The tests covered a range of scenarios: normal usage, error handling, and edge cases.

Particular attention was given to error scenarios and asynchronous operations. OpenAI responses can vary in quality and in rare cases, the returned data is in a format that is impossible for the app to parse, in these situations we test for the API call function to exit gracefully and retry the call. In the case of the PDF API, the responses are reliable but each feature requires a chain of

asynchronous HTTP requests, our unit tests ensure that each feature goes through each step of the process with correct values and returns a valid response.

Integration Testing

Our integration testing strategy assessed the interactions between UI components and both of the chosen APIs. We thoroughly tested component interactions, including user-triggered actions like button clicks and text inputs to ensure they properly triggered API calls and handled responses. More specifically, we examined UI behavior such as button enabling/disabling based on input as well as component rendering variations depending on state variables.

The entire application relies on a global state set correctly by the data returned from OpenAI and our context provider. Therefore, integration tests are conducted within this context provider to validate that all functions are done within the correct environment.

CI/CD infrastructure Description

Our CI/CD pipeline follows two main stages: testing and deploying. These two stages are handled by two workflow files **test.yml**, and **deploy.yml**.

Testing

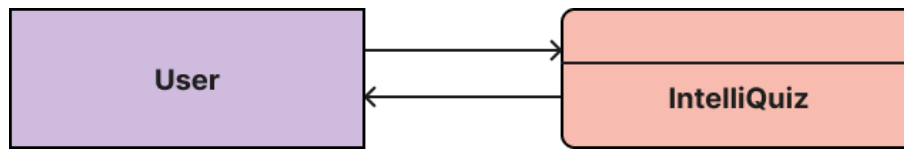
When a Pull Request is opened for the main branch of the repository, we use GitHub Actions to run all the unit and integration tests found in the repository on the code. If the tests do not pass, merging is blocked. The PR then needs to be closed, and reopened when new changes are made until all test cases pass. This prevents feature-breaking bugs reaching the main branch and the deployed site.

Deploying

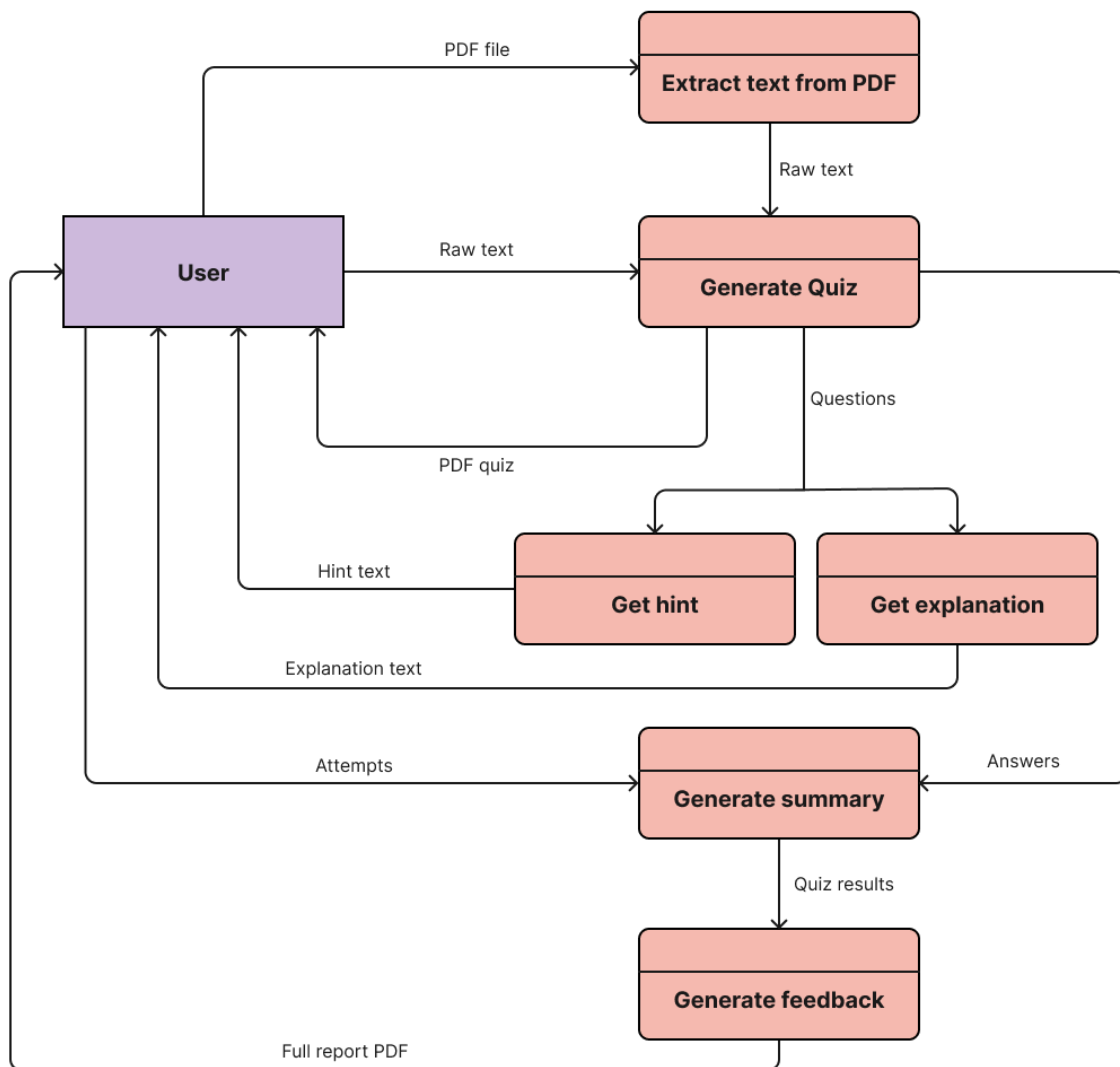
Our repository has configured Actions to build and deploy to GitHub Pages. After a change is made to the main branch, another workflow makes a build of the application using the environment variables containing our API keys, the build folder is delivered to GitHub Pages for deployment on the default URL.

Data Flow Diagram

Level 0



Level 1



Challenges & Takeaways

Challenges	Takeaways
Initially, the team relied solely on communication to avoid conflicts, leading to unreviewed commits polluting the codebase. A major bug was buried under these commits. Unfortunately, these updates also contained valuable work, so we couldn't simply undo them. This meant manually fixing the bug, which significantly delayed our progress.	<p>After the manual reset of the repository, the team unanimously decided to enforce branch protection rules to require a pull request with at least 1 review from a teammate to push changes to the main branch. This rule was made stronger as we added automated testing on open pull requests, to further help avoid such situations.</p> <p>We learned the importance of code reviews and branch protection, as well as the use of proper communication.</p>
<p>The initial PDF API, ILovePDF, could not generate documents from raw data. We were not aware of this before the implementation of the text extraction feature. This started a three-week long crisis where we had to find an API capable of all three features, replace our existing text-extraction feature and implement the remaining two: generating documents and password protection.</p> <p>The jump in complexity from ILovePDF to Adobe PDF Services is monumental, requiring multiple API calls, MS Word document templates, and Amazon S3 cloud object storage to perform just one of our features.</p>	<p>All of this could have been avoided by reading the ILovePDF documentation thoroughly and realising that it alone could not provide all three features.</p> <p>The team learned that documentation needs to be read under the context of what we are trying to achieve and the tools that we have at our disposal.</p> <p>Additionally, the complexity of the API taught us how to integrate multiple services and tasks under various API calls to achieve a single outcome.</p>
The entire team was unfamiliar with unit and integration testing. We left the testing cases almost until the end of the development process.	Fortunately, this did not cause any major roadblocks, but it did help us realize how important the testing phase is, and how it could have helped us avoid many previous headaches.
Not all of the team was familiar with the tech stack we chose for the project, this caused initial uneven contributions and a challenging experience for those who had to learn the stack.	When we noticed this situation, the team resorted to pair programming for code contribution and quickly noticed an improvement in code quality and work parity.

Work Division

Task	Harpreet	Miguel	Patrick	Utsav
Final Report		X		X
Web Page components	X			
Web Page design	X	X	X	X
OpenAI: Quiz Generation		X		X
OpenAI: Hint & Explanations			X	X
OpenAI: Feedback		X	X	X
AdobePDF: Text extraction		X		
AdobePDF: PDF generation		X		
AdobePDF: Password protection		X		
Unit Testing	X			
Integration Testing	X			
CI/CD Pipeline: Test, Build, Deploy		X	X	
Presentation	X	X	X	X
Demo video				X