Software Design Document

**PDF Grader**

Cole Behrndt

**Dane Buckley**

**Kiel Morris**

**Shadrach Roundy**

**Version 1.0**

**2/22/2022**

**WWU Computer Science**

**Table of Contents**

[**Introduction**](#_3znysh7) **1**

[**Overview**](#_l7zwzhn8p41t) **1**

[**System overview**](#_tyjcwt) **1**

[**Design guidelines**](#_35nkun2) **2**

[Design Concerns](#_e289kwelfz0n) 2

[**System architecture and architecture design**](#_z337ya) **2**

[**Design specification and validation**](#_3j2qqm3) **7**

[Narrative](#_qzoft7gv09ev) 7

[Non-functional requirements](#_lgezi6cwmh5b) 7

[Interface](#_j7xxese14yv4) 7

[**Goals and Milestones**](#_wak1ti3rmr7) **11**

[Milestones](#_vb6qw0hz0zr8) 11

[Finalize Libraries](#_diih2yz0wptf) 11

[PDF Viewing](#_lh6dowiyl6wi) 11

[MVC & Editable Values](#_b9osa41r48mi) 11

[Question Box Drawing](#_rl5gu2xctqet) 11

[Export Graded Tests & Test Overview](#_nos2zeiiyybb) 11

[Tested & Feedback Gathered](#_vwdo5dyg94r) 12

[Stretch Goals & QOL Features](#_t7fnb9rco8pv) 12

**Revision History**

| **Name** | **Date** | **Changes** | **Version** |
| --- | --- | --- | --- |
|  | 2/22/22 | Initial Document | 1.0 |
|  |  |  |  |

# Introduction

This document addresses the requirements of the PDF Grader while establishing goals and communicating design to stakeholders. The following are some of the goals and requirements for creating the PDF Grader.

* Importing PDFs
* Creating metadata for various test settings
* Highlighting questions for grading
* Grading questions in the most convenient order
* Exporting various statistics and graded tests

# Overview

Test grading is a time consuming task that professors have to go through. In a more online age, more and more often tests are being submitted through web based programs leading to digital files being graded. In order to more efficiently grade it is proposed that a piece of software can help a professor grade a PDF file while also speeding the process up by presenting the question in a more efficient order.

## 

Use Case Diagram

# System overview

The program will take in one pdf file with multiple tests in it. This means, for a five page test, every five pages is the fifth/last page of each test. You will input how many pages are in a test, and how many questions there are. The professor will then go through each page and drag a box over each question/answer location for the entire test. Using that box, it goes question by question (all the 1s, then all the 2s, etc) of each test for grading.

The grading is not automatic, the professor will be shown the question that is being graded and have an interface where he can type specific feedback for a question and how many points they’re going to lose to be reused on other students answering the same question. There will also be a variety of “canned” answers, but these will be very general. Out of bounds support is needed, so if an answer is trailing off the specified box, they can go to the entire page of that question and then return to the test grading software.

After an initial release more features may be added like allowing two or more people to run this software at the same time and thus be able to grade with a TA, as well as a barcode recognition to be able to automatically gather information.

Once the tests have been graded, the software will generate a table of scores as a pdf, and a pdf for each student. This pdf for the student will be as many pages as were questions in the test, and list their feedback that they received on each question (1 question per page).

# Design guidelines

The UX of the program should be built to be as unobtrusive and efficient as possible, as it is meant for one task repeated over and over again. Therefore menus/submenus should be avoided, clicks should be minimized and all functionalities should be always accessible through shortcuts and buttons.

## Design Concerns

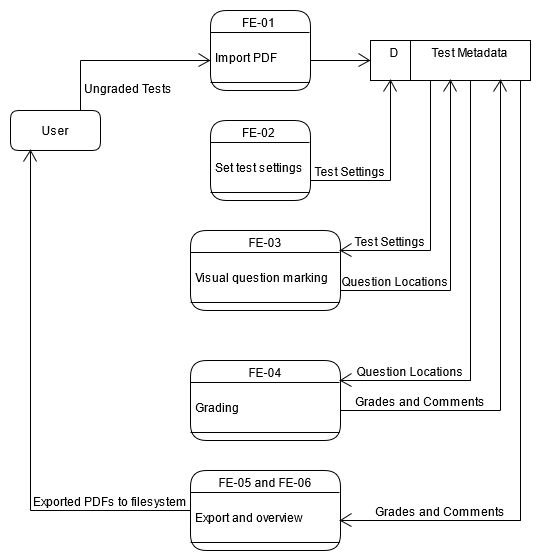
The application has to be functional for both digital and printed/scanned pdfs. This makes saving metadata to the pdf file difficult because a printed test wouldn't be able to retrieve that data. Although barcodes on tests may be able to solve these problems, it would introduce a large design hurdle that could possibly be solved in a more streamlined and straightforward fashion.

In order to minimize clicks while grading, it would be great if we could save general feedback to a question to a repository of answers. Then, when regrading a question a user could pick from previous marks to quickly drop in feedback. As of right now, that functionality is not built into the UI but should be put in later.

# System architecture and architecture design

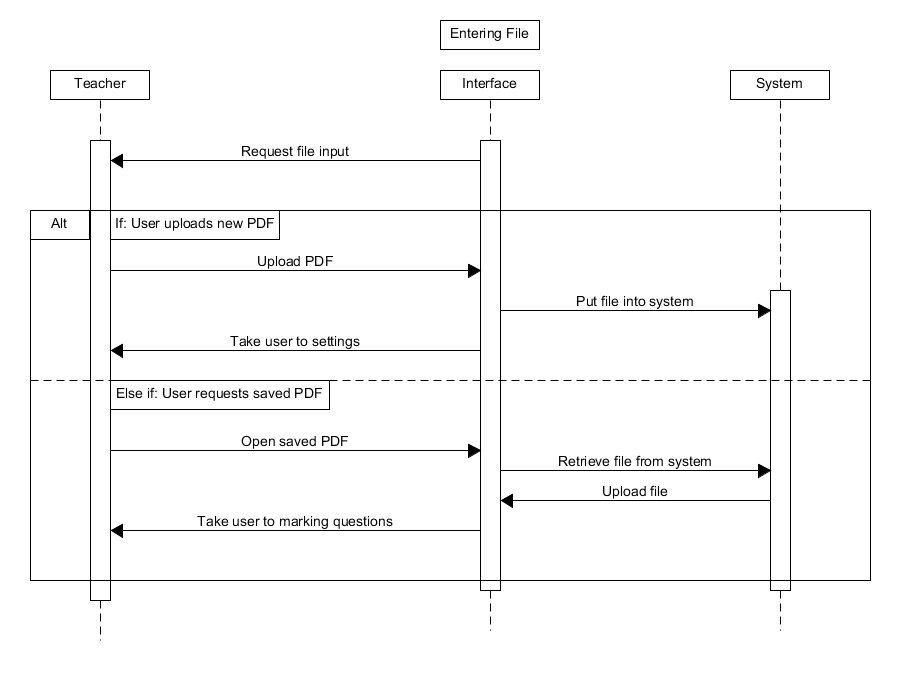
The program will be implemented by incorporating the Model View Controller design pattern. While the architecture is likely to change before and during implementation, this is a good first step to identifying problem points and places we should focus on when designing our application. Below there is a data flow diagram that visualizes what data is being used and

saved. While most goals have been fulfilled with this diagram, it is missing what is happening at a deeper level with the individual PDFs.

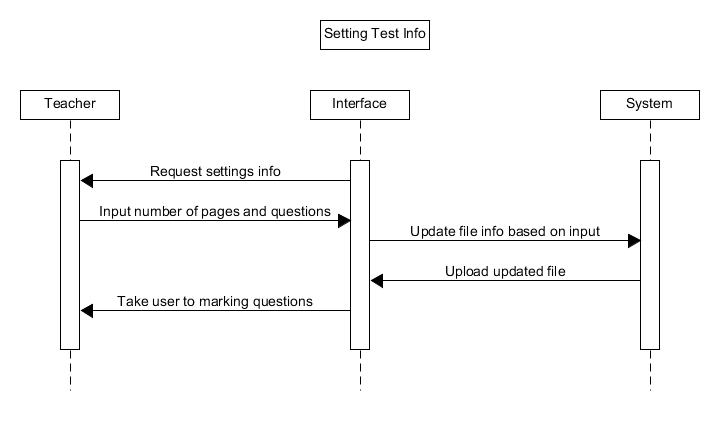


Data Flow Diagram

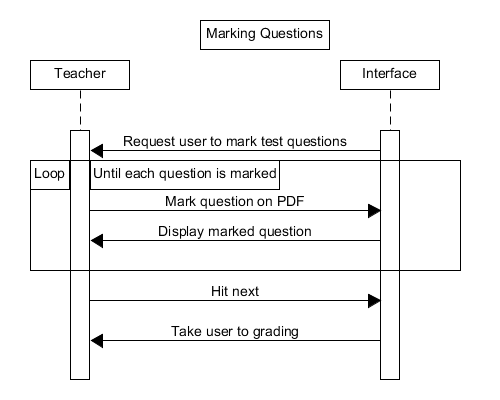
Following are sequence diagrams of all the major program use cases. Additionally, in order to more fully represent the architecture of the application, below we have a UML class diagram with the tentative classes that are planned to be used. This diagram is subject to change as we dive deeper into streamlining the UI and developing the app; as requirements might change and issues might arise.



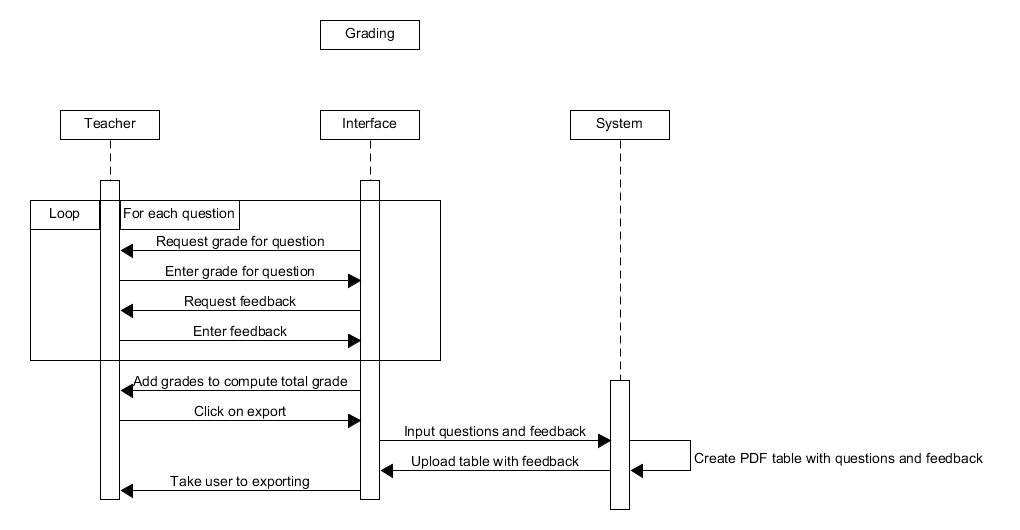
Task: Entering Test File



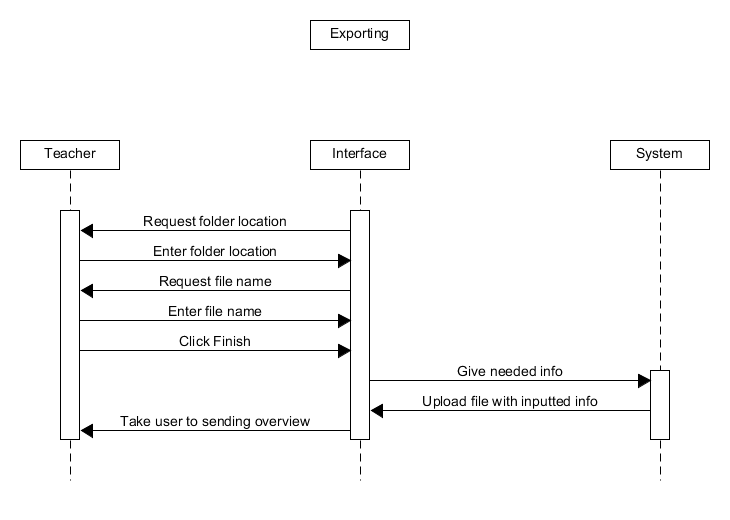
Task: Setting Test Information



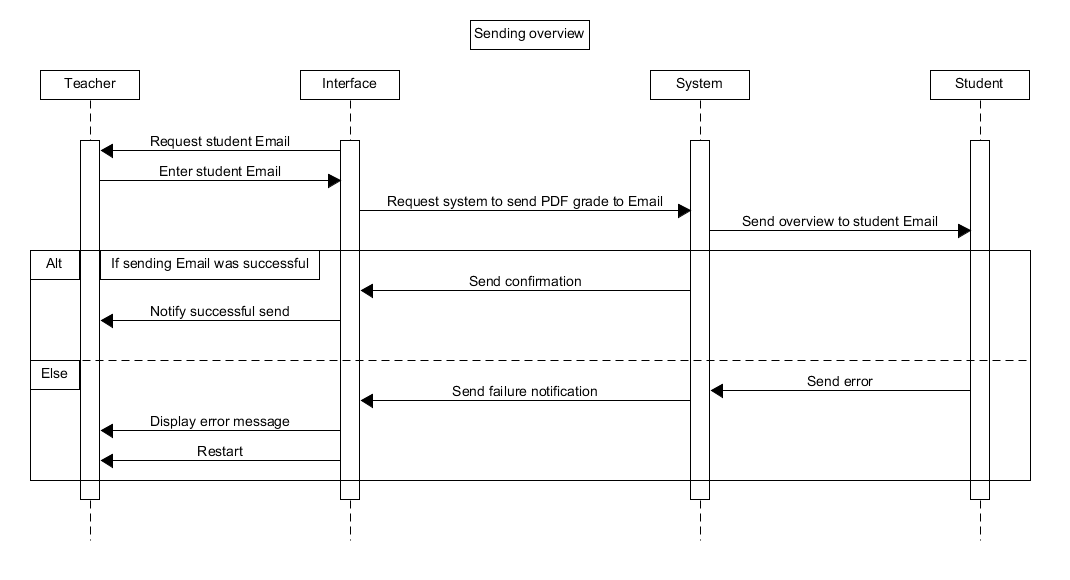
Task: Marking Question Areas



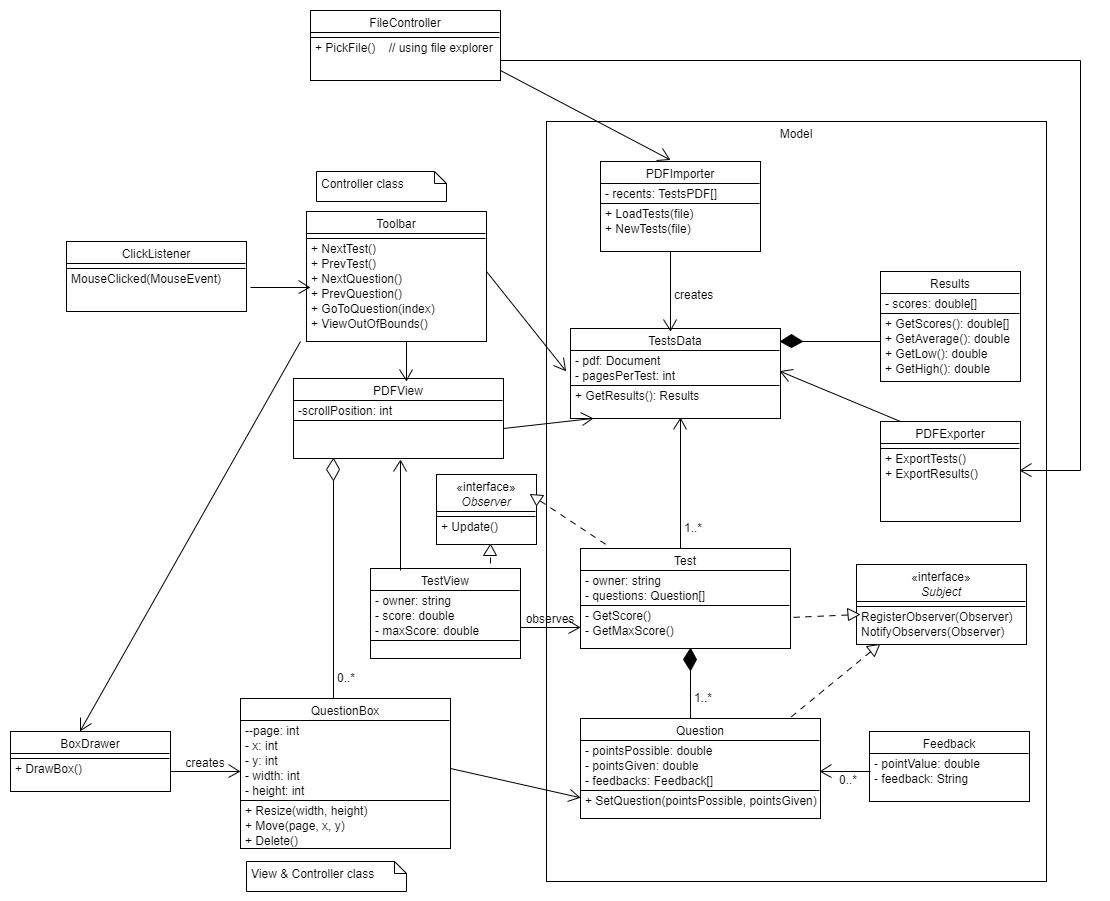
Task: Grading Tests



Task: Exporting Graded Tests



Task: Sending to Students



UML Class Diagram (WIP)

# Design specification and validation

## Narrative

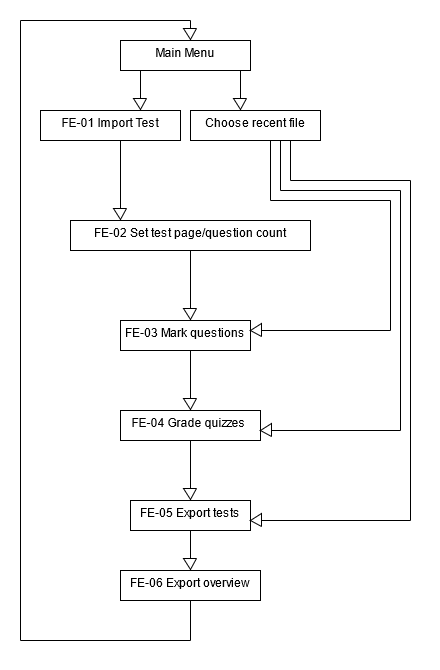
This product is meant to ease the difficulties and tedium of grading PDF assignments for teachers and TA’s, and this product is also meant to provide assignment grades to the students. Teachers and TA’s face the problem of grading assignments under a certain amount of time and that timeframe isn’t very feasible to accomplish when grading a PDF assignment with their current resources. This product fixes this problem by providing an interface that allows the teacher and TA’s to view questions and answers more easily and efficiently. Students face the problem of receiving the grade for an assignment and they don’t fully understand the reason behind their grades. This product fixes this problem by allowing the teacher to give a PDF to the student that gives specific details regarding the received grade.

## Non-functional requirements

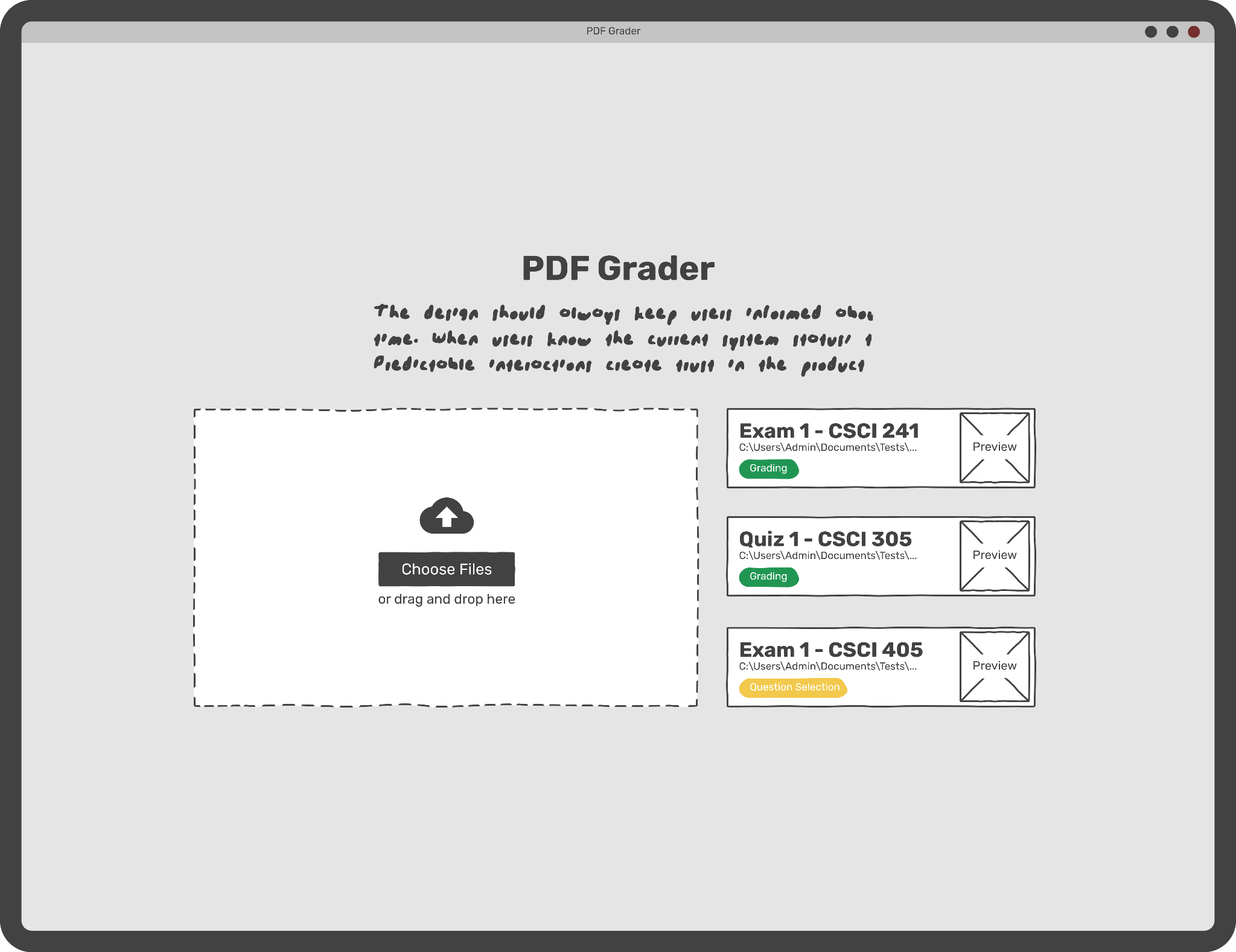
* Compatibility - The software should be able to build for Windows, Mac, and Linux operating Systems
* Security - The software must work entirely offline to protect confidential student data
* Performance - UI navigations between questions/tests should be near-instant for the grader
* Scalability - Should have comparable import and response time for PDFs up to 525+ pages (15 page test for 35 students, for example)
* Memorability - Graders should be able to quickly resume tests they had previously started
* Simplicity - Interface should not be overly cluttered. Only functions related to grading need to be readily accessible

## Interface

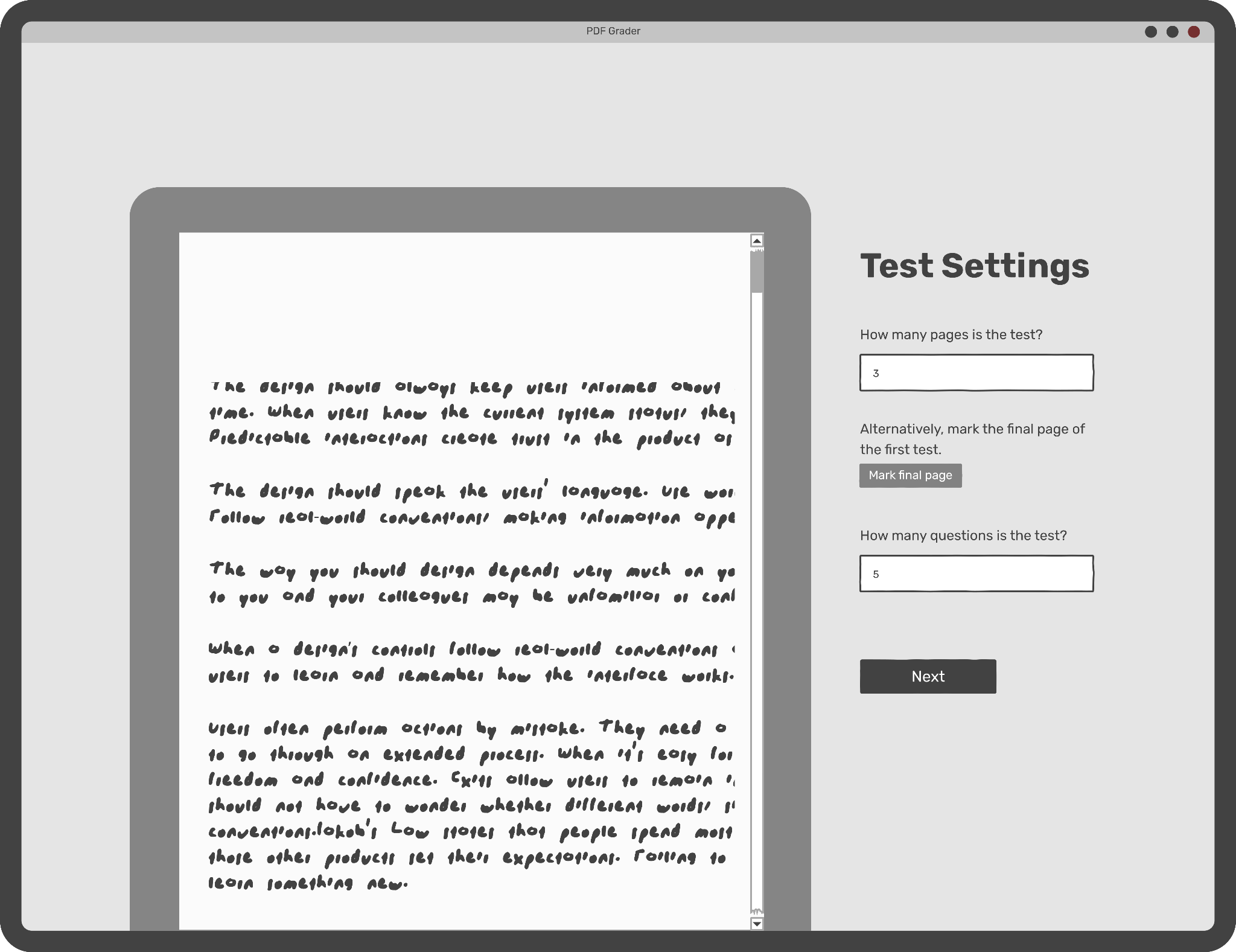
The interface is a very important part of this application because of how tightly integrated it has to be in order to make grading faster. If you have any extra clicks or sections that do not need to be there, it will slow down the process significantly due to having to do the action hundreds of times. The PDF grader has a straightforward set of steps that have to be achieved for all the tests to be graded; these can be seen in the Interface Structure Diagram. A mockup of the interface has also been created for the application. This part of the application has to be flexible and will be modified in the future after more testing in order to minimize time spent grading tests.



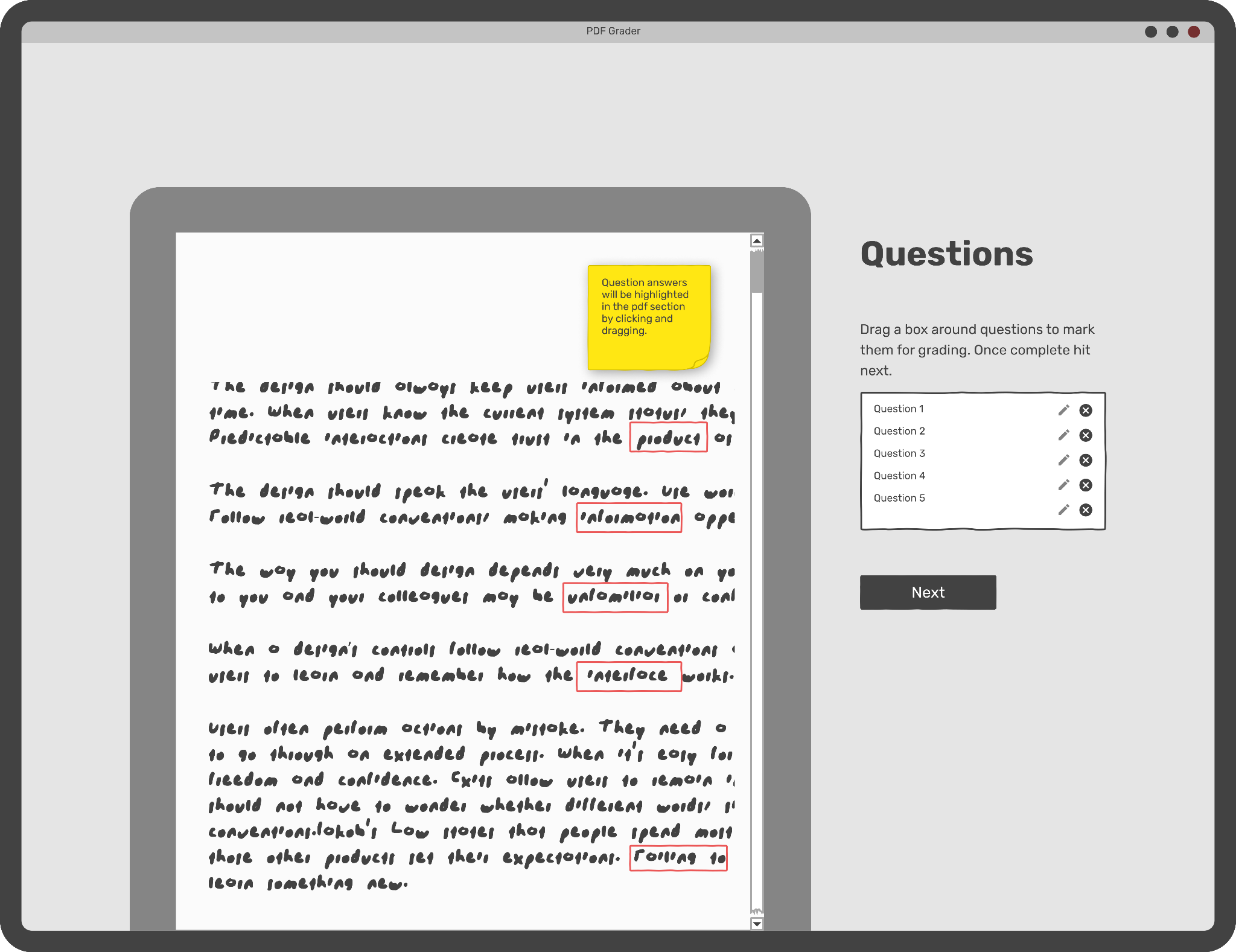
Interface Structure Diagram



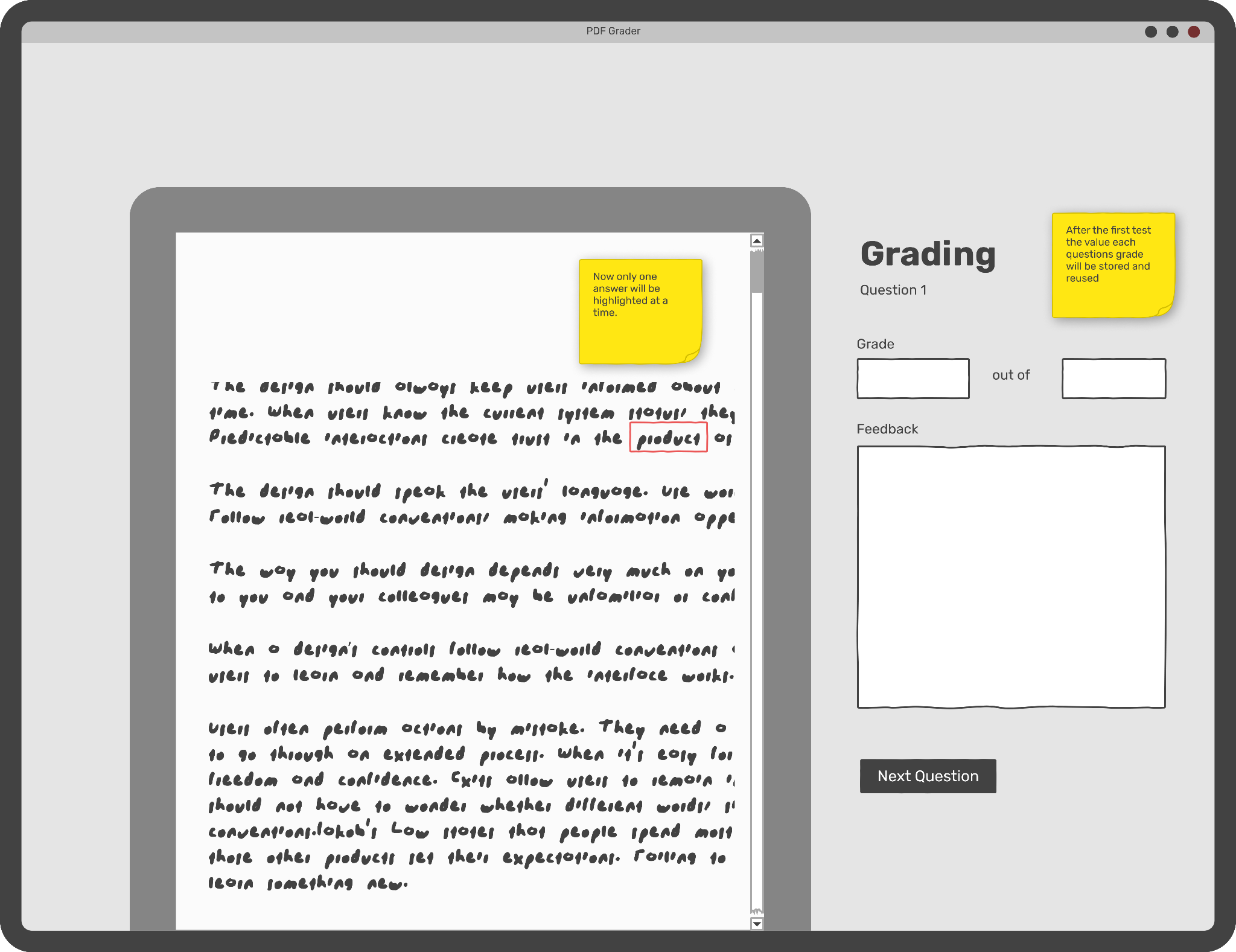
The introduction page lets the user import a pdf or choose from a list of already started tests.



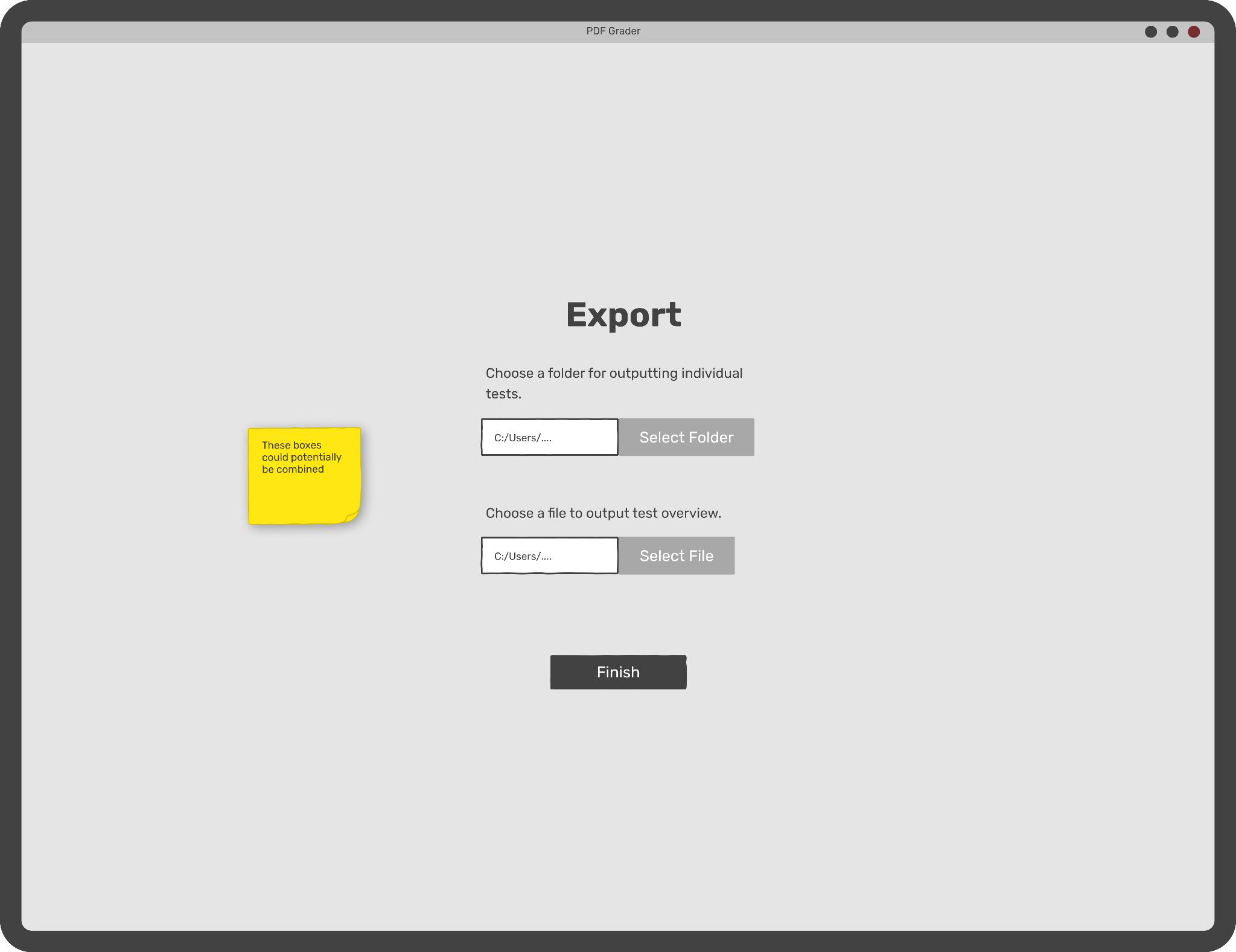
The test settings page sets up some simple settings.



The question highlighting page allows the user to highlight questions with a click and drag.



The grading page will be used for quick grading, saving certain values for later the user will see this page hundreds of times, so it needs to be streamlined.



Finally, the user will export their stats and tests when grading is completed.

# Goals and Milestones

While the PDF grader is easily divisible into chunks, some steps need to be completed before others and some are less important to complete for a minimum viable product. A list of various goals and milestones can be found below along with a priority matrix, where the top two quadrants of the graph represent our minimum viable product.

## Milestones

### Finalize Libraries

Before development begins, any dependencies needed for functionalities such as PDF manipulation and GUI-related tools need to be researched and finalized. Integrating them into the project first will provide a solid base for grading specific code to begin.

### PDF Viewing

The first core functionality of the software that needs completion is the requirement of importing, viewing, and navigating a pdf. Any buttons or gestures related to navigation should be included at this point, even if their integration/placement in the GUI is not finalized. This should be solidified early so that the other functionalities can be built on top of it independently.

### MVC & Editable Values

At this point, in addition to the basic PDF front-end view, the MVC design pattern should be mostly implemented. Our model of test data should be functional, with information pertaining to program flow such as page/question count compatible with navigation. Simple controller events to modify data such as question points and feedback should also be included. The controller may not be in line with our final UI design in terms of buttons, text fields and placement, but it should be able to edit and hold grade data correctly in appropriate model data structures. Thus, at least a single student test should be able to have grade data entered.

### Question Box Drawing

Included in this version is the ability to draw and edit the boxes that will designate a question space. This includes drawing the shape of a question onto the pdf document, resizing it by its corners, moving the box within the confines of the pages, and deleting a box to redraw it. These should all be functional through dynamic buttons and click events on the custom box object.

### Export Graded Tests & Test Overview

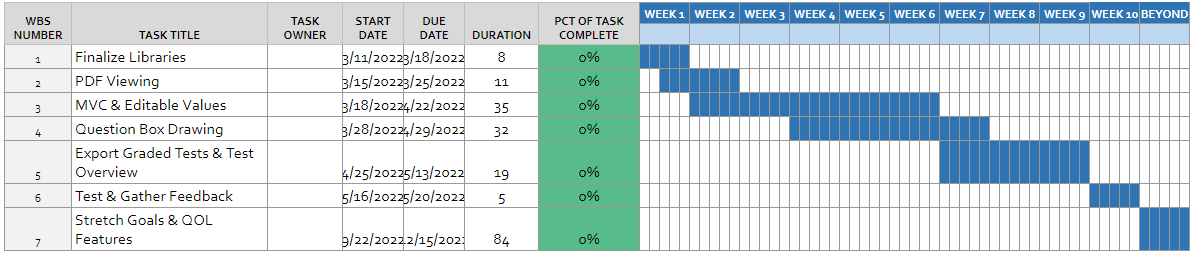
While grading should be functional within a program instance by milestone 3, this marks the completion of functionality to output individual students’ test results by retrieving question box data and test data and “baking” them into a new pdf document. Also included is the functionality to aggregate all test data and output an overview page containing scores as well as other calculated data such as average or median in a well organized table. With these features the application will meet criteria for a Minimum Viable Product.

### Tested & Feedback Gathered

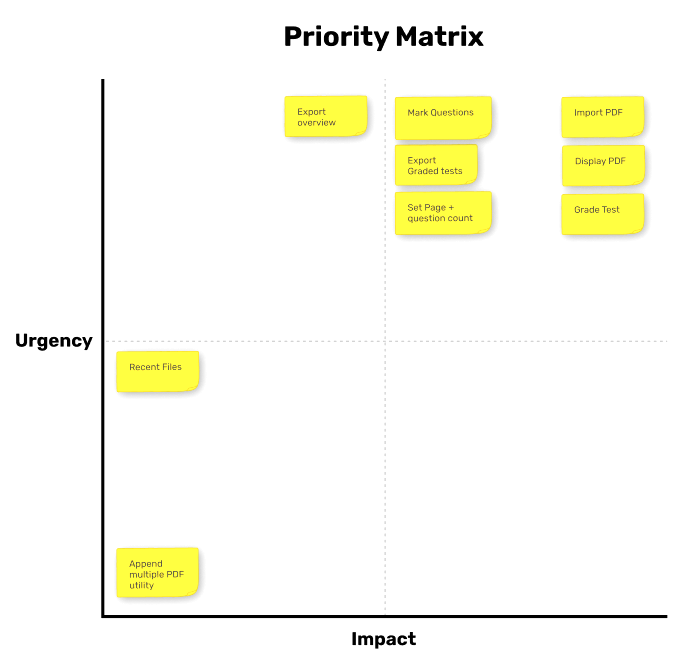
Once we have the MVP, the next task will be to test its functionality in practice by actually grading tests with our client, Aran Clauson. This process will illuminate the pitfalls and annoyances with our design (particularly UI) to the developers personally and with feedback from real stakeholders. With the gathered information we will optimize the application’s UI and workflow for ease of ease and efficiency.

### Stretch Goals & QOL Features

This stage represents the finished application once our development time has concluded. The final additions will be changes built upon feedback, or Quality of Life features missing from previous versions, such as “Recent Files” or “Append multiple PDF utility”. If time allows, extra features may be added. One possibility would be a barcode/glyph that holds test data (including pages, question box placement and point value, etc.) that can be read and loaded into the application to skip manual setup. However this stage is subject to modification as development progresses.



**Gantt Chart:** Visualizes the timeline specified above



A priority matrix to define how important tasks are for a minimum viable product.