

# **PDF Annotator for Moodle Quiz**

*A Project Report Submitted  
in Partial Fulfillment of the Requirements  
for the Degree of*

**Bachelor of Technology**

*by*

**Parvathy S Kumar & Asha Jose**  
(111901040 & 111901011)



INDIAN INSTITUTE  
OF TECHNOLOGY  
**PALAKKAD**

**COMPUTER SCIENCE AND ENGINEERING**  
**INDIAN INSTITUTE OF TECHNOLOGY PALAKKAD**

# CERTIFICATE

*This is to certify that the work contained in the project entitled “**PDF Annotator for Moodle Quiz**” is a bonafide work of **Parvathy S Kumar (Roll No. 111901040)** & **Asha Jose (Roll No. 111901011)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Palakkad under my guidance and that it has not been submitted elsewhere for a degree.*

**Dr Jasine Babu**

Assistant Professor

Department of Computer Science & Engineering

Indian Institute of Technology Palakkad

# Contents

<b>List of Figures</b>	<b>iv</b>
<b>1 Introduction</b>	<b>2</b>
1.1 Objective . . . . .	2
1.2 Scope of the project . . . . .	3
1.3 Organization of the report . . . . .	5
<b>2 Familiarization with Moodle code structure</b>	<b>7</b>
2.1 System Requirements . . . . .	7
2.2 Moodle Database . . . . .	8
2.3 Users and roles . . . . .	8
2.4 Plugins . . . . .	9
2.4.1 Common files required for Moodle plugins . . . . .	9
2.4.2 Sample plugins . . . . .	10
<b>3 Exploration of annotation tools and libraries</b>	<b>15</b>
3.1 Tools used in the preliminary version of the tool . . . . .	15
3.2 Libraries used in Xournal application . . . . .	16
3.2.1 Poppler and Cairo . . . . .	16
3.2.2 Workflow of annotation handling in Xournal . . . . .	16
3.2.3 Non-compatibility with PHP . . . . .	18

3.3	FPDF and FPDFI . . . . .	18
3.3.1	FPDF . . . . .	18
3.3.2	FPDI . . . . .	19
3.3.3	WorkFlow of annotation using FPDFI/FPDF . . . . .	19
<b>4</b>	<b>A new PDF annotation feature in Moodle using FPDF/FPDI</b>	<b>23</b>
4.1	Drawback of the PDF annotator used in assignment . . . . .	23
4.2	Drawback of the preliminary version of PDF annotation tool . . . . .	24
4.3	Workflow of the proposed plugin . . . . .	24
4.4	Step-by-step explanation of the workflow . . . . .	25
4.4.1	User interface and adding annotations over a file . . . . .	25
4.4.2	Serialization of the annotations . . . . .	27
4.4.3	Passing the serialized data to FPDF/FPDI and getting the output PDF . . . . .	28
<b>5</b>	<b>Integrating the PDF annotation tool with the Moodle quiz module and File API</b>	<b>29</b>
5.1	Quiz module . . . . .	29
5.2	File API . . . . .	30
5.2.1	get_file_storage class . . . . .	31
5.3	Getting the file submitted by the user . . . . .	32
5.4	Loading the plugin and annotating . . . . .	33
5.5	Saving the annotated file . . . . .	33
<b>6</b>	<b>Code structure and workflow of PDF annotator</b>	<b>37</b>
6.1	Loading the annotator window . . . . .	37
6.2	Supporting multiple file types . . . . .	39
6.2.1	Imagemagick . . . . .	40
6.2.2	Using Imagemagick in PHP . . . . .	40

6.3	PDF Annotator UI . . . . .	41
6.3.1	pdfannotate.js . . . . .	41
6.4	Annotating the file and uploading to database . . . . .	43
<b>7</b>	<b>Error handling, UI bug fix and testing</b>	<b>45</b>
7.1	Error Handling . . . . .	45
7.1.1	PDF Version above 1.4 . . . . .	45
7.1.2	Lack of permissions to directories . . . . .	46
7.1.3	PHP 8 Compatibility . . . . .	47
7.2	Bug Fixes in the UI . . . . .	48
7.2.1	Translation of freehand drawings . . . . .	48
7.2.2	Active tool fix . . . . .	49
7.2.3	Tool icon highlight fix . . . . .	50
7.3	Testing . . . . .	50
7.4	Installation and version support . . . . .	52
<b>8</b>	<b>Conclusion and Future Work</b>	<b>53</b>
8.1	Conclusion . . . . .	53
8.2	Future Work . . . . .	54
	<b>References</b>	<b>55</b>

# List of Figures

2.1	Roles and Descriptions . . . . .	9
2.2	Plugin to list the available courses for enrolment . . . . .	11
2.3	Plugin to list those online users who share at least a course with the logged-in user . . . . .	11
2.4	Plugin to create the Message . . . . .	12
2.5	Adding of pages for message creation . . . . .	12
2.6	Successful Creation of Message . . . . .	12
2.7	Messages are added to the Database . . . . .	13
3.1	Workflow of rendering PDF . . . . .	17
3.2	Functioning of FPDF/FPDI . . . . .	19
3.3	PDF Annotated using FPDF/FPDI . . . . .	20
3.4	Size of the input PDF . . . . .	21
3.5	Size of the output PDF . . . . .	21
4.1	Workflow of the PDF annotator . . . . .	25
5.1	mdl_file table . . . . .	30
5.2	File record object . . . . .	31
5.3	Usage of <code>get_file</code> function . . . . .	31
5.4	Function to save a local copy of the input PDF . . . . .	33
5.5	Sending annotation data to php . . . . .	33

5.6	Getting annotation data from javascript and creating a file object . . . . .	34
5.7	User Interface for annotation . . . . .	35
5.8	Size of the input and output PDFs . . . . .	35
6.1	Workflow of the annotation process . . . . .	38
6.2	Workflow of annotator.php . . . . .	42
6.3	Uploading and annotating the file . . . . .	44





# Chapter 1

## Introduction

A Learning Management System (LMS) is a software that assists in implementing and assessing any learning process. An LMS provides an organized way for delivering courses to students wherein the delivering of lectures, discussions of subjects, uploading and submitting of assignments, quizzes, and other related activities as well as tracking the student's progress can be done in the same software. This makes the process of teaching and learning much more simpler and effective.

Moodle [1] is an open-source software that is widely used as a learning management system. In IIT Palakkad as well, Moodle is the learning management software used for course content management. The word “Moodle” was originally an acronym for Modular Object-Oriented Dynamic Learning Environment. It supports multiple features that enhance the process of remote learning thereby making the process of online learning organized and simpler.

### 1.1 Objective

Moodle quiz module allows the students to upload files, such as PDF documents, as answers to the essay-type questions. The main objective of our project is to develop a PDF annotation feature for the quiz module. Such a feature is currently not available with Moodle.

Currently, the teacher has to download the PDF uploaded by the student, correct it in an outside Moodle environment and re-upload it in Moodle.

A basic version of the essay-type answer annotation feature was developed by Vishal Rao and Tausif Iqbal [2], as part of their final year project work. However, the implementation had a drawback which is that the annotated file size was often much larger compared to the original submitted file.

In this project, our major goal is to improve the software by fixing this issue. The basic requirements expected of the tool are that PDF annotations should not change the basic properties of the original PDF being annotated and should not considerably increase the size of the annotated PDF, in comparison with the original file. The PDF annotator should have all the necessary tools required by a teacher to grade a quiz, such as a brush for freehand drawing/writing over the PDF, a highlight box for highlighting some areas of text in the document, and a text box to add comments in the answer scripts. The PDF annotator should also be able to annotate additional file types commonly used by uploading answers, such as images and plain text files.

## 1.2 Scope of the project

As mentioned above, currently Moodle does not support the annotation of files uploaded as essay-type answers in quizzes. However, Moodle's assignment module has a PDF annotation tool that allows the teacher to correct the PDF uploaded by students in assignments. We cannot reuse the same PDF annotation feature used in the assignment module since it is using `yui.js`, a library that is no longer maintained.

A preliminary version of the requisite tool was developed by Vishal Rao and Tausif Iqbal [2]. The javascript based front-end of their annotator and associated basic functionalities were adapted from the work of Ravisha Hesh [3] and they used the well-known FabricJS open-source library to annotate PDFs. In their work, while saving the annotated PDF, each page along with the annotations on that page was converted into an image and

the images together were converted into a PDF. This resulted in a significant increase in the size of the annotated PDF and the properties of the PDF lost. This is because in a text document, one character is just 1 byte while even a small image needs more space to be stored since they are recognized by computers as pixels instead of characters. While this image is converted to PDF, the bit depth and the resolution of an image overall increase the size of the PDF which is undesirable. In our work, we provide an improved tool, by fixing this issue, as summarized below.

In advanced PDF annotation tools, the annotations are done as a vector graphics layer above the original PDF, which would conserve the properties of the PDF and would not drastically increase its size. We initially explored some popular open-source PDF annotation libraries which work in this way, such as Poppler [4] and Cairo [5]. Since the integration of these libraries was not complete in PHP, which is the basic platform of Moodle base code, we chose to use the FPDF/FPDI bundled library [6, 7]. These are PHP libraries that help to load a PDF file and annotate it using the different functions present in them.

We adapted the FabricJS based front-end of the annotator developed in [2]. We implemented some additional functions for supporting the front-end, such as adding translation for the freehand drawings, locking some movements, and scaling. These were not required in [2], because of their direct conversion to images, which in turn led to the blow-up in file size. We have added a serialization function that would serialize the annotations done in the PDF into a JSON format, while the user clicks the button to save annotated file. The JSON data is then sent to the back-end Moodle code, where it is parsed and converted to the format supported by FPDF/FPDI using the functions provided by FPDF/FPDI and is used to produce the annotated file, which is then stored in Moodle's database. This is fundamentally different from the work done in the previous year and this is critical for the file size. The file handling in the back-end is also done as required by the annotator. On the Moodle side of file handling, we have maintained a similar code structure as in [2]. We have adapted their PHP code suitably for integration with FPDF/FPDI and have done a

cleaning up of their base code to make it better maintainable.

In our code, we have tried to do proper handling of the errors that we could anticipate. We have tested the code for various use cases and error situations and have fixed all issues encountered.

A repository containing the source code required for our PDF annotator tool and its installation support including a Makefile and supporting documentation is maintained in Github [8] and can be found at: [https://github.com/Parvathy-S-Kumar/Moodle\\_Quiz\\_PDF\\_Annotator](https://github.com/Parvathy-S-Kumar/Moodle_Quiz_PDF_Annotator). This repository contains both our source code and the required files of the FPDF/FDPI support libraries. Our PDF annotator works readily for Moodle versions from 3.11 to 4.2 and is compatible with PHP 7.4 and PHP 8 versions. The documentation also directs the user on how to make the tool work with earlier Moodle versions.

### **1.3 Organization of the report**

This chapter briefly discusses the importance of the learning management system and about Moodle, the LMS used in IIT Palakkad. We have described the objective of our project and the problems we are trying to tackle through our project. We have also discussed the setting up and important features of Moodle like the Moodle database and users and roles in Chapter 2. It also explains the plugin development process along with some sample plugins we developed in order to familiarise ourselves with the process. In Chapter 3, we have briefly discussed the different annotation tools and libraries including Xournal, its libraries, and FPDF/FPDI. In Chapter 4, we have discussed the new plugin which we developed using FabricJS and FPDF/FPDI. Chapter 5 deals with the integration of the new PDF Annotation tool with the Moodle quiz module. In Chapter 6, we have explained in detail the file structure, functions, and workflow of the PDF annotator. Details about error handling, testing, bug fixes, and version control are explained in Chapter 7. Finally, in Chapter 8, we have briefly discussed the works done till now along with the future works.



# Chapter 2

## Familiarization with Moodle code structure

In this chapter, we first describe the basic setup of Moodle that we used for our experiments. Then we describe some of our initial learning experiments to familiarize ourselves with the code structure of Moodle, before starting with the development of the intended feature.

### 2.1 System Requirements

Moodle is an open-source software and its source code is maintained in GitHub [9]. We are currently using the latest LTS version of Moodle - Moodle 4.1 [10]. We have verified that in any version from Moodle 3.11 to Moodle 4.2, our tool works without any modifications.

A web server, a database, and PHP are the basic software requirements for running and setting up Moodle. The components and their versions we are currently using are listed below.

- Apache2.4.41
- MySQL 8.0.30
- PHP 8.0

The complete set of software and hardware system requirements can be found in the release notes of Moodle. 4.1 [10]. For earlier versions of Moodle, the software versions of the dependencies would also be earlier versions and the details of it would be available from the release notes of the corresponding Moodle release.

## 2.2 Moodle Database

The latest LTS version of Moodle has a total of 470 tables. For any developer who is working on Moodle, it is important to understand the basic organization of these tables. Some of the most relevant tables are mentioned below.

- Course and Enrolments
  - mdl\_course
  - mdl\_enrol
- Users
  - mdl\_user
  - mdl\_user\_lastaccess
  - mdl\_user\_enrolments
- Quizzes
  - mdl\_quiz
  - mdl\_quiz\_grades

## 2.3 Users and roles

A role is a system-wide defined collection of permissions that can be assigned to a specific user in a specific context. Fig. 2.1 shows the different default roles available in Moodle. The roles that are most relevant for our project are Teacher and Student.

Role ?	Description
Manager	Managers can access courses and modify them, but usually do not participate in them.
Course creator	Course creators can create new courses.
Teacher	Teachers can do anything within a course, including changing the activities and grading students.
Non-editing teacher	Non-editing teachers can teach in courses and grade students, but may not alter activities.
Student	Students generally have fewer privileges within a course.
Guest	Guests have minimal privileges and usually can not enter text anywhere.
Authenticated user	All logged in users.
Authenticated user on site home	All logged-in users in the site home course.
Add a new role	

**Fig. 2.1** Roles and Descriptions

## 2.4 Plugins

Plugins are pluggable functionalities that could be to the original software without altering them. Modularity and extensibility are fundamental principles followed in Moodle development and this is primarily achieved through the development of plugins. In this section, we describe some of our initial experiments with Moodle plugin development.

### 2.4.1 Common files required for Moodle plugins

There are multiple files that can be present in any Moodle subsystem or plugin. Some of the important files are discussed below

- **version.php**: It contains information about the metadata of the plugin and is used during installation and upgradation. This is a mandatory file for every plugin. It includes information such as the minimum Moodle version required, the dependencies, and the version number.
- **lang/en/pluginname\_pluginname.php**: There must be an English translation for each language string defined by a plugin. This is a mandatory file for every plugin.



- **db/access.php**: This contains the initial information about the access control rules of a plugin

### 2.4.2 Sample plugins

#### **A plugin to list the available courses for enrolment**

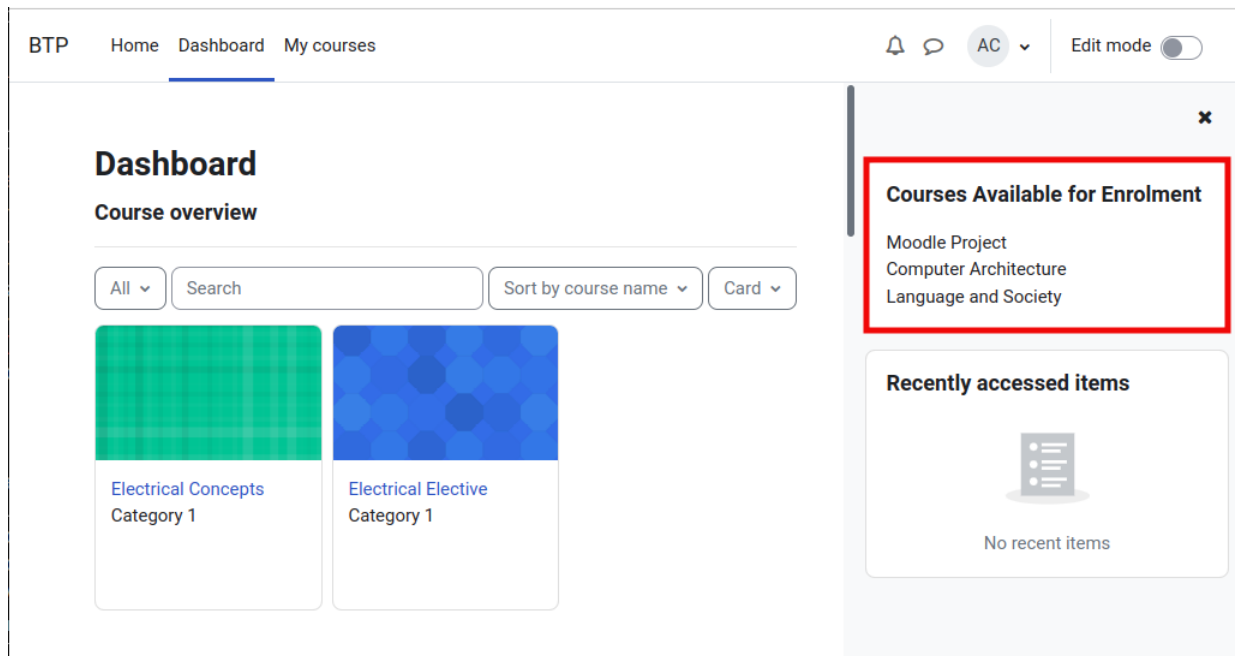
This was an experimental plugin that we created and is used to display the list of available courses that the user is still not enrolled in. See Fig.2.2 to see a screenshot of the usage of this plugin. This plugin is developed using the help of 3 tables of Moodle namely, course, enrol, and user\_enrolments.

#### **A plugin to list the online users who have at least 1 course in common with the logged-in user**

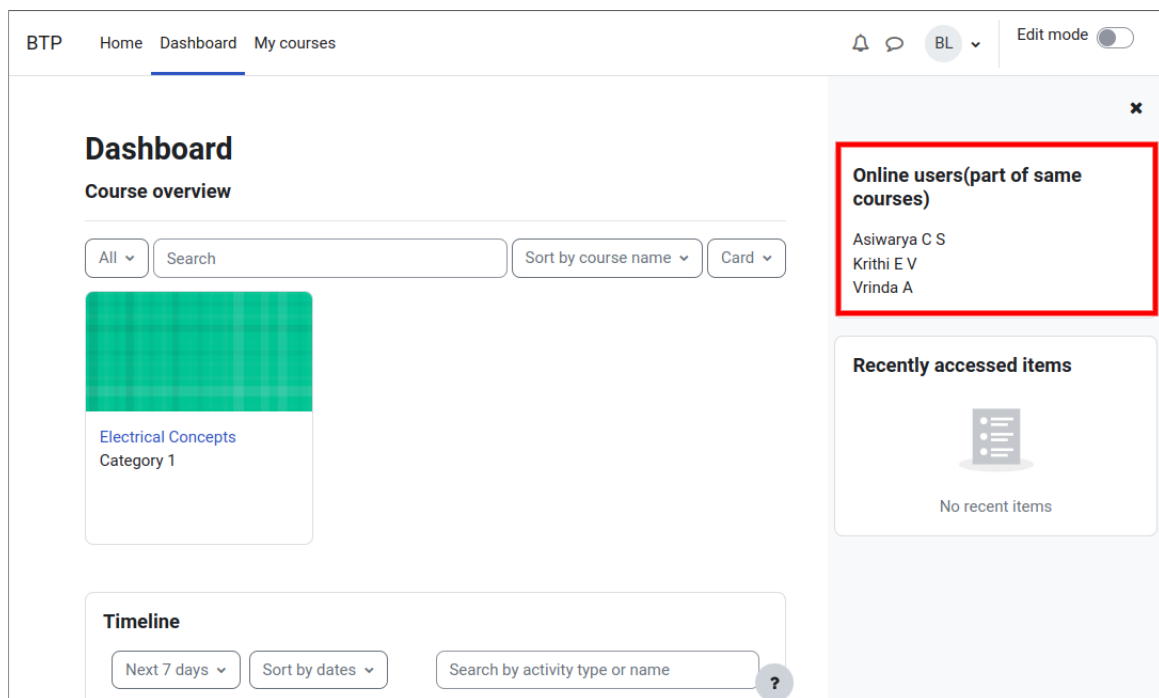
This was another experimental plugin that we developed for displaying only those users who are relevant to the logged-in user, i.e. the users who share at least one course with the logged-in user. See Fig.2.3 This plugin uses the *user\_lastaccess* and the *user* tables of Moodle.

#### **A plugin to create a message and store that message in the database**

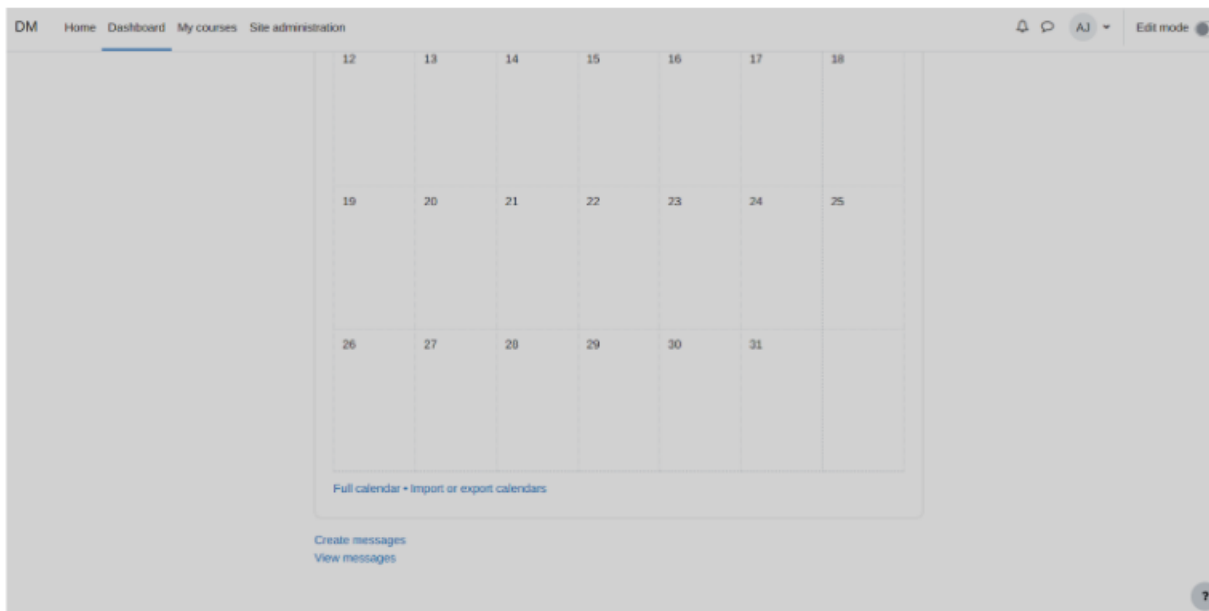
This plugin is used to allow the user to create a message which will be stored in the database. This plugin involves creating pages and traveling to pages. Figures 2.4 to 2.7 shows some screenshots showing the working of this plugin and the database update is done.



**Fig. 2.2** Plugin to list the available courses for enrolment



**Fig. 2.3** Plugin to list those online users who share at least a course with the logged-in user



**Fig. 2.4** Plugin to create the Message

DM Home Dashboard My courses Site administration

---

messagetext

[View messages](#)

**Fig. 2.5** Adding of pages for message creation

you created a message

List of messages:

Hello

Good Morning

[Create messages](#)

[Back to dashboard](#)

**Fig. 2.6** Successful Creation of Message

```
mysql> select * from mdl_local_message;
+----+-----+
| id | messagetext |
+----+-----+
| 1  | Hello       |
| 2  | Good Morning |
+----+-----+
2 rows in set (0.00 sec)
```

**Fig. 2.7** Messages are added to the Database



## Chapter 3

# Exploration of annotation tools and libraries

In this chapter, we describe the explorations that we did for deciding on a suitable annotation tool with proper open-source library support before proceeding with the development work of our annotation tool.

### 3.1 Tools used in the preliminary version of the tool

Initially, we studied the works done as a part of the project "Experiments with Moodle - Plugin development/enhancements" by Vishal Rao and Tausif Iqbal. The libraries used as a part of PDF annotation in the project were `PDF.js`, `FabricJS`, and `jsPDF`. Here, `PDF.js` is converting the PDF to an image as a background and includes annotations over it as fabric objects, an object in the said library, to get the resultant PDF. Because of this, the size of the PDF was increased drastically and since the final PDF is a converted image, we would not be able to select text from the PDF.

## 3.2 Libraries used in Xournal application

Xournal [11] is an open-source application for note-taking. It allows basic sketching and writing with both the keyboard and the mouse. One of the features of Xournal is to annotate a PDF which allows one to open an existing PDF file and make annotations on it. It enables the basic editing options such as drawing, erasing, selecting, undoing, and redoing that are needed for annotating a quiz file submission. This made us consider the annotation libraries used in Xournal to be integrated into Moodle for quiz submission file annotation. Another reason why Xournal was initially preferred for our reference is that, in the latest version of Xournal (0.4.8), the export to PDF version is updated by using the Cairo library and this fixed the issue of the file that is exported after annotation being of large size. The different layers of an annotated PDF being stored as an image create the large-size file issue. Our aim was to solve this issue by storing this as SVG itself and converting it to PDF without storing it as an image, like how Xournal does.

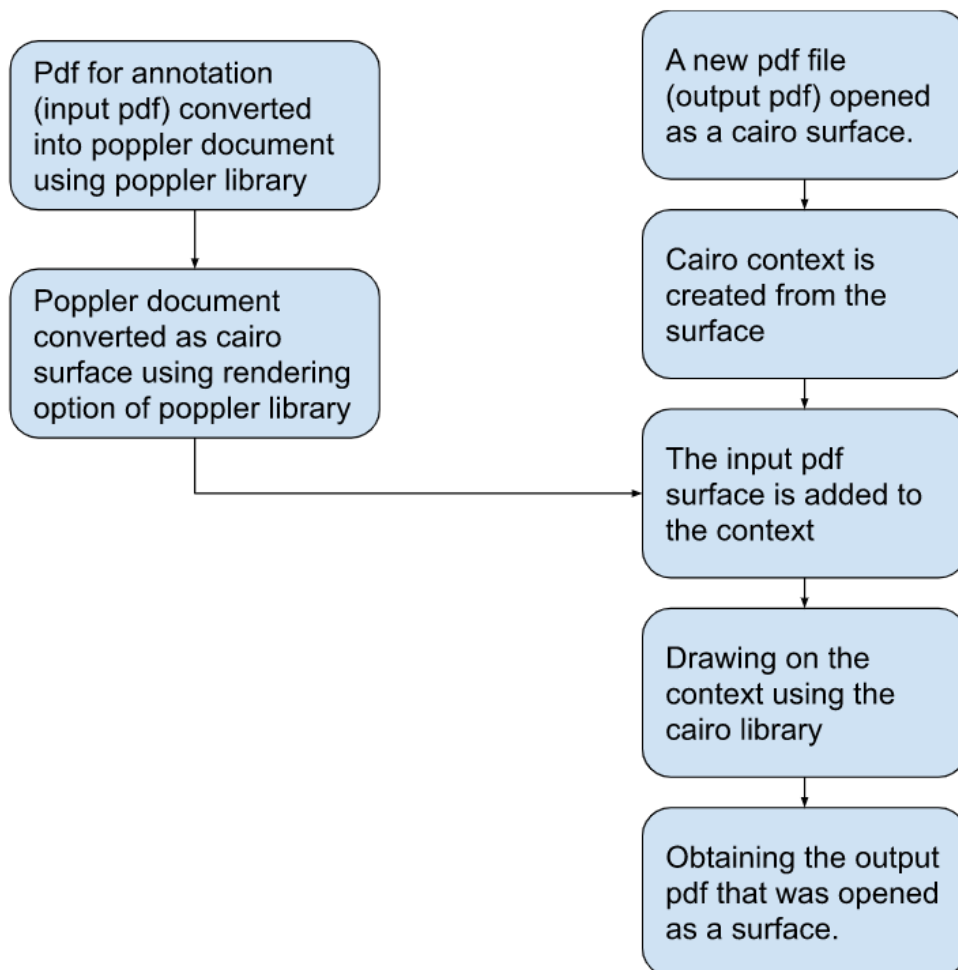
### 3.2.1 Poppler and Cairo

Since version 0.4.5, Xournal uses the Poppler library [4] for annotating a PDF. Poppler uses Cairo and Splash in their back end. Cairo [5] is a 2D graphics library that provides a vector-based implementation and Splash is a javascript rendering service. The usage of the Poppler library in Xournal enables the different layers of annotated PDF to be stored as SVG (scalable vector graphics) and integrate them efficiently allowing the annotated file to be of fairly small size.

### 3.2.2 Workflow of annotation handling in Xournal

Fig. 3.1 shows the workflow of annotation feature handling in Xournal.

The input PDF is first loaded as a Poppler document and then rendered using a function in the Poppler library to a Cairo surface and this surface is imposed on an existing context. Features are then added as new surfaces to this context. Opening a new PDF Cairo surface



**Fig. 3.1** Workflow of rendering PDF



results in a new PDF file being created and this function returns a surface from where we get a Cairo context. Modifying this context results in the PDF created being modified. After adding the input PDF surface, annotations like drawing shapes are added to this context further.

### **3.2.3 Non-compatibility with PHP**

The Poppler and Cairo libraries are originally written in C++ and C respectively and hence our initial attempt to explore the layer addition in this library was through C++. Although both these libraries are supported by various other languages, most of them are not stable. The Cairo library in PHP is not a maintained version. Also, Poppler PHP does not have all the functions that are present in the C++ Poppler library, especially the converting PDF to Cairo surface function is present not in a suitable format.

This made us look for other libraries that are compatible with PHP and would use a similar workflow. Majorly, we were looking for a library that does not convert the original PDF to an image. The FPDF and FPDI library together serves as a good option, hence we decided to proceed with that.

## **3.3 FPDF and FPDI**

FPDF and FPDI together can be used to edit an existing PDF and add annotation layers over it. This method does not increase the size of the resultant PDF hence helping us to solve the main problem we are trying to tackle through our project which is a huge increase in the size of annotated PDF.

### **3.3.1 FPDF**

FPDF stands for Free Portable Document Format [7]. It is a PHP class that aids in the generation of PDF documents and adding annotations over them. It supports images in

different formats. It also allows one to set up colors and links in the PDF. FPDF is released under a permissive license and there is no usage restriction.

### 3.3.2 FPDF

FPDI stands for Free Portable Document Importer [6]. It converts PDF documents to templates that can be used by FPDF. FPDF on itself cannot load an existing PDF. It can only generate a new PDF. FPDI converts an existing PDF into templates such that FPDF can work on it. The main uses of FPDI are defining the PDF from which the pages have to be loaded, importing a page of a document, and using the imported page on a page created with FPDF.

### 3.3.3 WorkFlow of annotation using FPDF/FPDI

The workflow of annotation of a file FPDF/FPDI is as shown in Fig. 3.2.



**Fig. 3.2** Functioning of FPDF/FPDI

To understand the usage of these libraries, we created a toy application for annotating a PDF. Upon compiling and running, the PDF file generated contains all the annotations made (see Fig. 3.3) and the size is not much different from the original size of the PDF (see Fig. 3.4 and Fig. 3.5).

The working of our toy application is as follows. Using the `setSourceFile()` function of FPDI, we loaded an already existing PDF. This function is used to load a PDF into a template which in turn can be used by FPDF functions to add annotations. This function takes the path of the input PDF as input and returns the number of pages in the PDF as output. In Moodle, the same function can be used to load the PDF documents submitted

Asses  
X3  
(assessment?  
name=98)

○ Week 3  
Feedback  
form:  
German-I  
(unit?  
unit=28&lesson=33)

Week 4 ()

DOWNLOAD  
VIDEOS ()Text  
Transcripts ()

Books ()

Ergänzen Sie die Verben in der richtigen Form.  
sein -- sprechen -- lernen -- wohnen -- kommen -- heißen

B: Guten Morgen. Ich \_\_\_\_ 1 \_\_\_\_ Bauer. Und wer \_\_\_\_ 2 \_\_\_\_ Sie?

K: Mein Name ist Karashima. Ich \_\_\_\_ 3 \_\_\_\_ aus Japan. Und Sie?

B: Aus Österreich. Aber ich \_\_\_\_ 4 \_\_\_\_ in der Schweiz. In Basel.

K: Interessant. Welche Sprachen \_\_\_\_ 5 \_\_\_\_ Sie?

B: Deutsch und Englisch. Und ich \_\_\_\_ 6 \_\_\_\_ Spanisch.

1) 1 \_\_\_\_\_

heisse

1 point

2) 2 \_\_\_\_\_

heissen

1 point

3) 3 \_\_\_\_\_

komme

1 point

4) 4 \_\_\_\_\_

wohne

1 point

5) 5 \_\_\_\_\_

sprechen

1 point

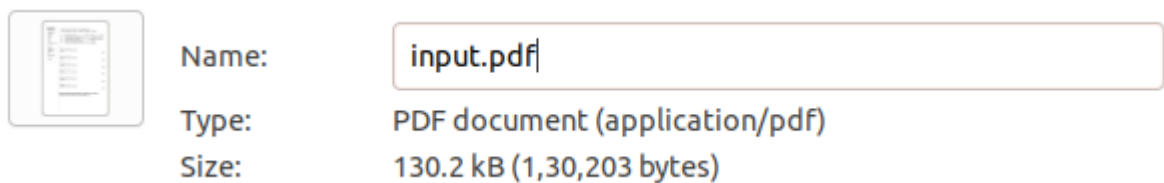
6) 6 \_\_\_\_\_

spreche

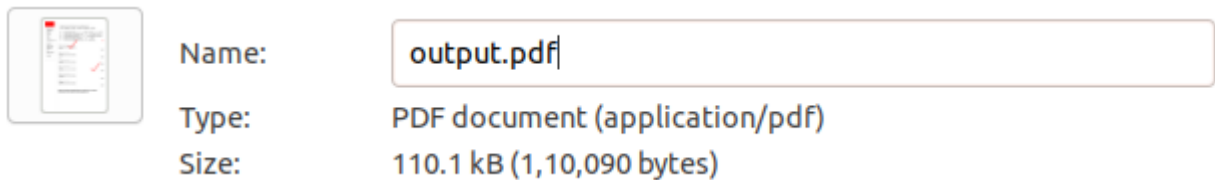
1 point

Write the articles and plural forms as shown in the example.  
Schreiben Sie die Artikel und die Pluralformen.

Fig. 3.3 PDF Annotated using FPDF/FPDI



**Fig. 3.4** Size of the input PDF



**Fig. 3.5** Size of the output PDF

by students for annotation. Once we load the PDF, we can edit page-by-page, by using the `importPage()` function of FPDF to load the page and by annotating using different tools present in FPDF. The `importPage()` function is used to import a page of a PDF by taking the page number as input which is then transformed into a form `XObject`. Form `XObject` is a technique for representing objects such as pages and images within a PDF document. The return value is an id to the imported page.

There are multiple annotation tools present in FPDF. Some of them are the Line tool, Rect tool(Rectangle), Ellipse tool, and Circle tool. Drawing and filling colors can be changed using the functions in FPDF. Using the `write()` function we can add text to the existing document. The text font, its size, and its color can be altered using different functions present in FPDF. It also supports adding images over PDF, a functionality that we are not using at present.

Once the document annotation is done, we can output the PDF and can download it or save it locally using the `output()` function of FPDF. The output function of FPDF supports 3 formats:- displaying the PDF in a browser, locally saving the PDF, or force downloading the PDF. In Moodle, the same function can be used to create the annotated PDF that can be stored in the database.



## Chapter 4

# A new PDF annotation feature in Moodle using FPDF/FPDI

From the previous chapter, we came to the conclusion that the PHP libraries FPDF/FPDI can be used to save a PDF along with the annotations without affecting the size and quality of the PDF. We looked into the other PDF annotators used in Moodle to get an idea of their working and finally made a new PDF annotator which uses a combination of FabricJS and FPDF/FPDI.

### 4.1 Drawback of the PDF annotator used in assignment

There is an existing PDF annotation feature in Moodle available for the assignment module. It uses a combination of Javascript and PHP libraries to annotate the PDF. The libraries used here are `YUI.js`, `TCPDF`, and `FPDI`. Moodle does not recommend reusing this plugin for the quiz module since `YUI.js` is no longer maintained [12]. Also, we cannot use the same plugin for the quiz module since assignment and quiz are two different modules with different features. In the Assignment module, when a student uploads a PDF, the teacher can check it using this plugin. In the quiz module, the PDF annotation part is only required for one type of quiz called essay type question wherein students has an option either to

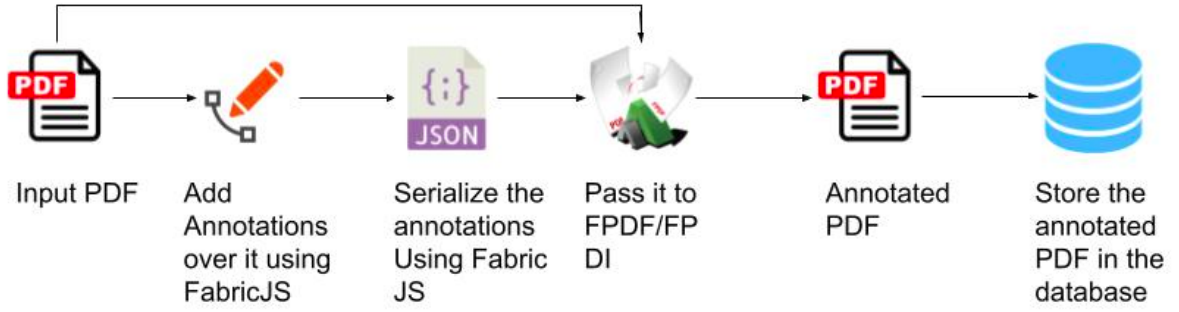
type on the space provided or upload a file as an answer.

## **4.2 Drawback of the preliminary version of PDF annotation tool**

As discussed in the last chapter, in the implementation of Tausif Iqbal and Vishal Rao [2], the javascript plugin for the user interface used FabricJS, jsPDF, and PDF.js. The main drawback of this plugin is the size of the annotated PDF and the fact that the annotated PDF will lose the properties of a PDF since it is converted to an image during the process.

## **4.3 Workflow of the proposed plugin**

In the proposed PDF annotator plugin, the drawing over the PDF and displaying it while annotations are added are done using FabricJS [13], and it is only converted to FPDF objects after saving by clicking on the save button. After a teacher adds annotations over a file submitted by the student, the annotations over the file are serialized using the help of a serialization function that is written as an extended part of the FabricJS module. There are parser functions that parse these serialized data and convert it into data as required by FPDF and FPDF. Now we have the necessary inputs required by FPDF and FPDF for annotation - A PDF and annotation data. By doing this, we are trying to reproduce the file that is being displayed in the UI with the help of FabricJS. The annotated file which is given as the output by FPDF and FPDF will have sizes in ranges similar to that of the input PDF and all the properties of a PDF will be preserved. Now the output PDF will be stored in the Moodle database using the File APIs provided by Moodle. The workflow of the plugin can be seen in Fig: 4.1



**Fig. 4.1** Workflow of the PDF annotator

## 4.4 Step-by-step explanation of the workflow

### 4.4.1 User interface and adding annotations over a file

For the front end of the User Interface part of the plugin, we are using the same UI developed by Vishal Rao and Tausif Iqbal [2] since it has all the necessary features needed for annotating a PDF. The main library required for the UI is FabricJS [13].

#### FabricJS

FabricJS is an open-source Javascript library that can be used to draw various objects and shapes over the HTML Canvas. It takes care of Canvas state and rendering and allows the users to work directly on the objects. There are 7 basic shapes provided by FabricJS and they are `fabric.Rect`, `fabric.Triangle`, `fabric.Polygon`, `fabric.Line`, `fabric.Polyline`, `fabric.Circle` and `fabric.Ellipse`. It also allows us to modify these objects. Many features of these objects can be modified such as the stroke thickness, stroke color, angle of rotation, color, transparency, and relative position of the object on the canvas. FabricJS also allows one to add images to the canvas using `fabric.image` and modify it as required by the user.



## Annotation options available in our annotator

Our user interface supports the following type of annotations to be added to the document.

- **freehand drawing:** This option allows for drawing and writing over the PDF. This could be possibly used for some circling of mistakes or providing feedback in the form of short sentences written by hand. The color of the brush can be selected from a list of colors. The drawing can be translated into different parts of the PDF. Scaling and rotation of a freehand drawing are disabled.
- **drawing highlight box :** The highlight box can be used to highlight any part of the PDF document, possibly to highlight a mistake made by the student. The highlight box is in fact a transparent rectangle. It can be scaled and translated. The fill color of the highlight box can be changed. Rotating a highlight box is disabled.
- **adding text over the file:** There is a text box using which we can add texts over the PDF, possible to provide a small paragraph of typed-in text as feedback to the student. The font size and color can be changed using the options present in the UI. The text box can be translated but cannot be rotated or scaled.
- **changing the stroke and fill color:** The stroke and fill color can be selected from a list of colors present in the UI. These are supported for all the options described above.
- **select:** The user can select an annotation object already added, using this option. Scaling/translation can be done on objects after selecting, except when those options are disabled.
- **delete:** Using this option, the user can select an annotation object already added and click delete to delete that object.
- **saving the annotated file:** On clicking the save button all annotations in the PDF are saved and stored in Moodle's database.

#### 4.4.2 Serialization of the annotations

We need to serialize the annotations made using FabricJS so that we can convert these objects to compatible formats supported by FPDF and FPDI. In serialization, all the data related to an object that is required to completely reproduce the object is added to a JSON file. This means the final annotation data of objects while saving is only present in the JSON file. Once every object in the file is serialized, we can read the resultant JSON file and get the corresponding objects in FPDI and FPDF. FabricJS has in-built functions for serialization. However, these methods need to be extended so as to handle the translation of annotated added using freehand drawings. This is because, the serialization function is called only while the save button is being clicked and if an object is being translated, the JSON file created using the original serialize function of FabricJS would not contain the previous data of the same object.

Some examples of the properties used in the serialization of various annotation objects are given below.

- **highlight box** - Highlight box is a rectangle so, the serialized data include the x-coordinate and y-coordinate of the top left corner, along with the width, height, stroke-thickness, stroke-color, and fill-color. Using these data we can reproduce any rectangle for drawing a highlight box in the file.
- **text** - For text, the serialized data include the text string, the font, the color, the size of the text, and the coordinates of the top-left corner of the bounding box of the text.
- **freehand drawings** - Freehand drawings are considered as a set of very small lines. So the serialized data of a free hand drawing consists of an array of points that are stored while a user drags the mouse. Lines are drawn between every two consecutive points to get the curve.

#### **4.4.3 Passing the serialized data to FPDF/FPDI and getting the output PDF**

The JSON file consisting of the serialized data has to be parsed first and each Fabric object has to be converted to an FPDF object. After parsing the entire JSON file, we will get all the objects as required by FPDF/FPDI. This along with the original input file, when passed to FPDF/FPDI, will render an output PDF that has a size in a range similar to the input PDF with all the properties of the PDF preserved.

##### **Parsing the JSON data**

The JSON file contains a JSON object which holds all the fabric objects and the details about them one after the other. This file is opened and read by a PHP file line by line as a string. This is done because a JSON object cannot be recognized by PHP language and data should be retrieved by parsing each line. A PHP function is used to segregate each object by finding certain keywords in a line and it also identifies the nature of the fabric object. Depending on the object type, separate functions are called for parsing the content of the file containing that object. Details like object type, coordinates, color, and other relevant values are stored in a PHP array and this array is later used for creating FPDF objects with similar properties as the FabricJS objects.

## Chapter 5

# Integrating the PDF annotation tool with the Moodle quiz module and File API

The PDF annotation feature has to be added for the essay-type questions of the quiz module. We have to integrate the PDF annotator proposed in the previous chapter with the quiz module of Moodle. The main features needed for it are:

- A teacher should be able to see the files submitted by the students in the annotator
- After saving, the new annotated file has to be added to the Moodle database

### 5.1 Quiz module

The quiz module allows the teacher to add a quiz activity to a course where she can add different types of questions like multiple-choice, fill-in-the-blanks, and essay types. Our PDF annotator is needed for the essay-type questions in the quiz module. An essay-type question supports multiple file formats like document files (PDF, RTF, TXT), Image files(JPG, PNG, SVG), audio files, video files, and multiple other types of files. Our

annotator is mainly aimed at the document and image files uploaded as answers to essay-type questions.

## 5.2 File API

For managing all the files in Moodle, Moodle provides File API [14]. There are a set of functions provided by File APIs through which we can access, serve, delete, and create files in the Moodle database. We use some of the functions of Moodle's File API in order to get the files submitted by the user displayed in the user interface of the annotator and to save the annotated PDF in the Moodle database.

In Moodle, files are stored according to the SHA-1 hash of their content. All files are stored in a table called mdl\_file. The mdl file can be seen in Fig: 5.1.

```
mysql> SHOW columns from mdl_files;
```

Field	Type	Null	Key	Default	Extra
id	bigint	NO	PRI	NULL	auto_increment
contenthash	varchar(40)	NO	MUL		
pathnamehash	varchar(40)	NO	UNI		
contextid	bigint	NO	MUL	NULL	
component	varchar(100)	NO	MUL		
filearea	varchar(50)	NO			
itemid	bigint	NO		NULL	
filepath	varchar(255)	NO			
filename	varchar(255)	NO			
userid	bigint	YES	MUL	NULL	
filesize	bigint	NO		NULL	
mimetype	varchar(100)	YES		NULL	
status	bigint	NO		0	
source	longtext	YES		NULL	
author	varchar(255)	YES		NULL	
license	varchar(255)	YES	MUL	NULL	
timecreated	bigint	NO		NULL	
timemodified	bigint	NO		NULL	
sortorder	bigint	NO		0	
referencefileid	bigint	YES	MUL	NULL	

20 rows in set (0.16 sec)

**Fig. 5.1** mdl\_file table

Files in Moodle are conceptually stored in File Areas that are used to restrict modules from accessing files belonging to only that particular module. These are uniquely identified by a context id, full component name, file area type, and unique item id. In order to display and serve a user with a file, first we should get the URL of the file. This URL generally has a file-serving script like `pluginfile.php`. The general form of the URL is

```
$url = $CFG->wwwroot/pluginfile.php/$contextid/$component/$filearea/  
arbitrary/extra/information.ext
```

An example of a URL is

```
$url=http://localhost/moodle/pluginfile.php/120/question/  
response_attachments/4/1/2/111901040_CN_LAB10.pdf
```

For most of the functions of the File API, we need to create a File Record Object, and in order to get the file we need to use the `get_file` function.

```
$fileinfo = array(  
    'component' => 'mod_mymodule',    // usually = table name  
    'filearea' => 'myarea',          // usually = table name  
    'itemid' => 0,                    // usually = ID of row in table  
    'contextid' => $context->id,      // ID of context  
    'filepath' => '/',                // any path beginning and ending in /  
    'filename' => 'myfile.txt');      // any filename
```

**Fig. 5.2** File record object

```
$file = $fs->get_file($fileinfo['contextid'], $fileinfo['component'], $fileinfo['filearea'],  
    $fileinfo['itemid'], $fileinfo['filepath'], $fileinfo['filename']);
```

**Fig. 5.3** Usage of `get_file` function

### 5.2.1 `get_file_storage` class

All the functions we need for the annotator are stored in the `get_file_storage` class of the Moodle File API. These are

## Reading a file

The function will retrieve the record we need from the table. After getting the file URL, we cannot use PHP functions like `get_file_contents()` on the Moodle files.

```
if ($file) { $contents = $file->get_content();  
} //If a file exists then get the contents of the file.
```

## Deleting a file

The function will remove the file from the table `mdl_files`. It will check to see if any other files require the content; if not, it will delete the content file. Then `admin/cron.php` delete will delete the content files from the trash directory.

```
if ($file) {  
    $file->delete();  
} //If a file exists, delete the file
```

## Creating a file

The function will determine the SHA1 hash of the file's contents and check to see if a file with this SHA1 hash already exists on the disc, either in the file directory or in the file trash. If it does not, it will save the file there. Using the low-level address, it will add a record for this file to the files table.

## 5.3 Getting the file submitted by the user

Using the FileAPI provided by Moodle, we can get the URL of the file submitted by the user. This URL is needed to display the file in the UI in FabricJS. This part of the PDF Annotator has already been done by Vishal Rao and Tausif Iqbal and hence we are reusing that part of the Annotator.

Using the `get_last_qt_files()` and `get_response_file_url()` we will get the URL of the required files. Additionally, here we need to save a local copy of the files submitted by the user since for annotation, FPDF and FPDI will not accept a URL and need a reference of the location of the input PDF in a local directory. The functions `file_get_contents()` and `file_put_contents()` would not work for Moodle files. It is not permitted to access the file directly from the disc. Instead, we need to make a duplicate or local copy of the file in a temporary location and use that as the input of FPDF/FPDI. For this, we need to use the `copy_content_to()` function.

```
$path = getcwd();  
$original_file->copy_content_to($path . "/dummy.pdf");
```

**Fig. 5.4** Function to save a local copy of the input PDF

## 5.4 Loading the plugin and annotating

In the existing `comment.php` file of the quiz module, there is a reference to `annotator.php`. In the `annotator.php`, we are loading the user interface of the plugin using `index.html`, `clickhandlers.js` and `pdfannotate.js`. The functions called by the interface from `index.html` are contained in `clickhandlers.js` and `pdfannotate.js`.

## 5.5 Saving the annotated file

After the file annotation is done, when the user clicks the Save button, we need to send the annotation data to `upload.php` from the `pdfannotate.js` file. For this purpose, we can use `XMLHttpRequest`.

```
xmlhttp.send("id=" + value + "&contextid=" + contextid + "&attemptid="+attemptid +  
"&filename=" + filename + "&furl=" + furl);
```

**Fig. 5.5** Sending annotation data to php



```
//Getting all the data from mypdfannotate.js
$value = $_POST['id'];
$contextid = $_POST['contextid'];
$attemptid = $_POST['attemptid'];
$filename = $_POST['filename'];
$component = 'question';
$filearea = 'response_attachments';
$filepath = '/';
$itemid = $attemptid;
```

**Fig. 5.6** Getting annotation data from javascript and creating a file object

We have added the required folders of FPDF and FPDI inside the quiz module. Now we have the input PDF(saved from `annotator.php`) and the annotated data. Now we will output the resultant PDF in the local folder. Then we need to save this output PDF into the Moodle database. We first check if the file exists before saving it to the Moodle database. If the same file already exists, we replace it; otherwise, we create a new one. `create_file_from_pathname()` can be used to create a new file.

The user interface of our PDF annotator can be seen in Fig: 5.7 and the size of input and output PDFs can be seen in Fig. 5.8.

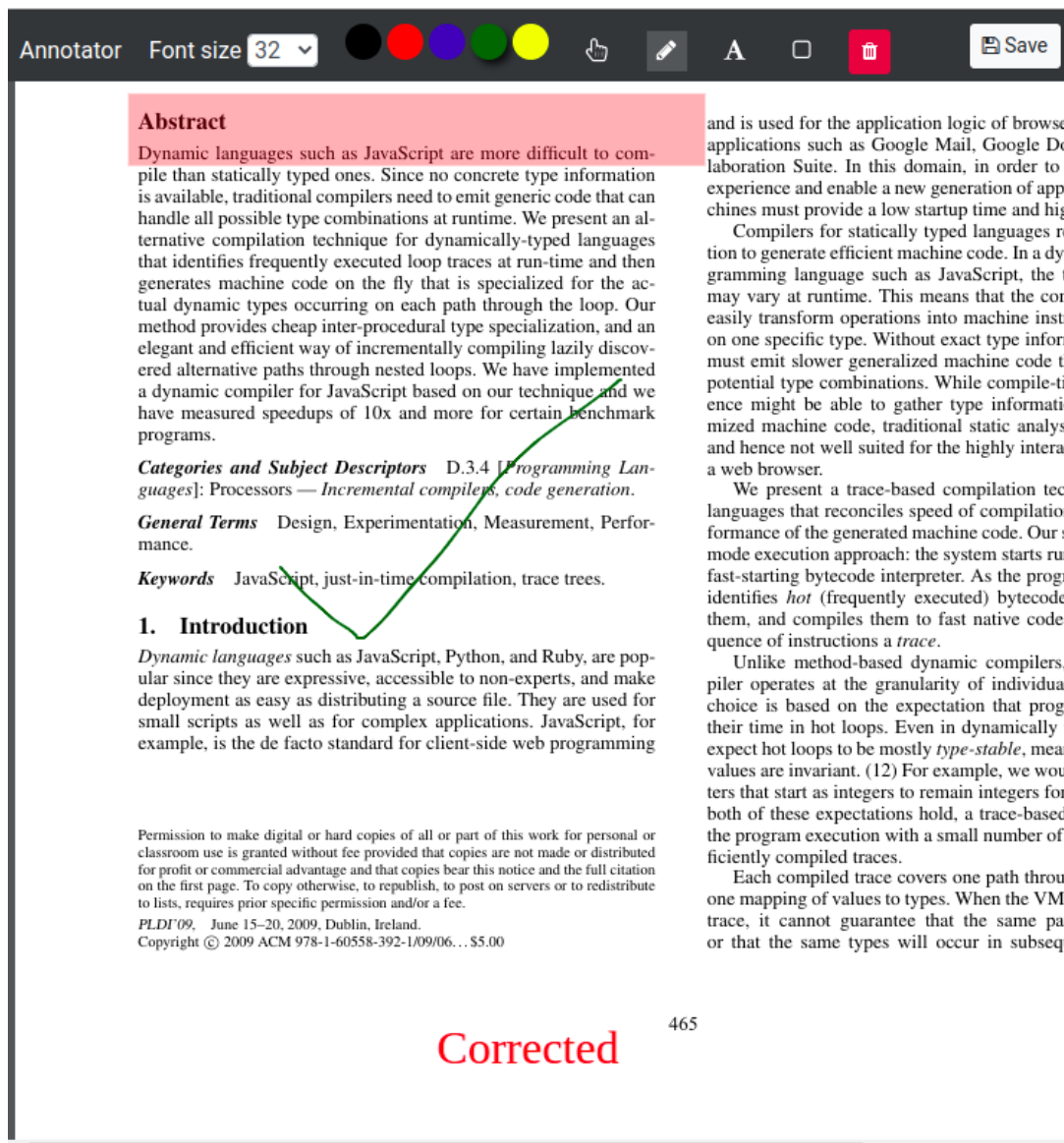


Fig. 5.7 User Interface for annotation



gal-trace.pdf Properties				gal-trace_annotated.pdf Properties			
Basic	Permissions	Open With	Document	Basic	Permissions	Open With	Document
	Name:	gal-trace.pdf			Name:	gal-trace_annotated.pdf	
	Type:	PDF document (application/pdf)			Type:	PDF document (application/pdf)	
	Size:	1.0 MB (10,47,267 bytes)			Size:	1.1 MB (10,54,204 bytes)	

Fig. 5.8 Size of the input and output PDFs



# Chapter 6

## Code structure and workflow of PDF annotator

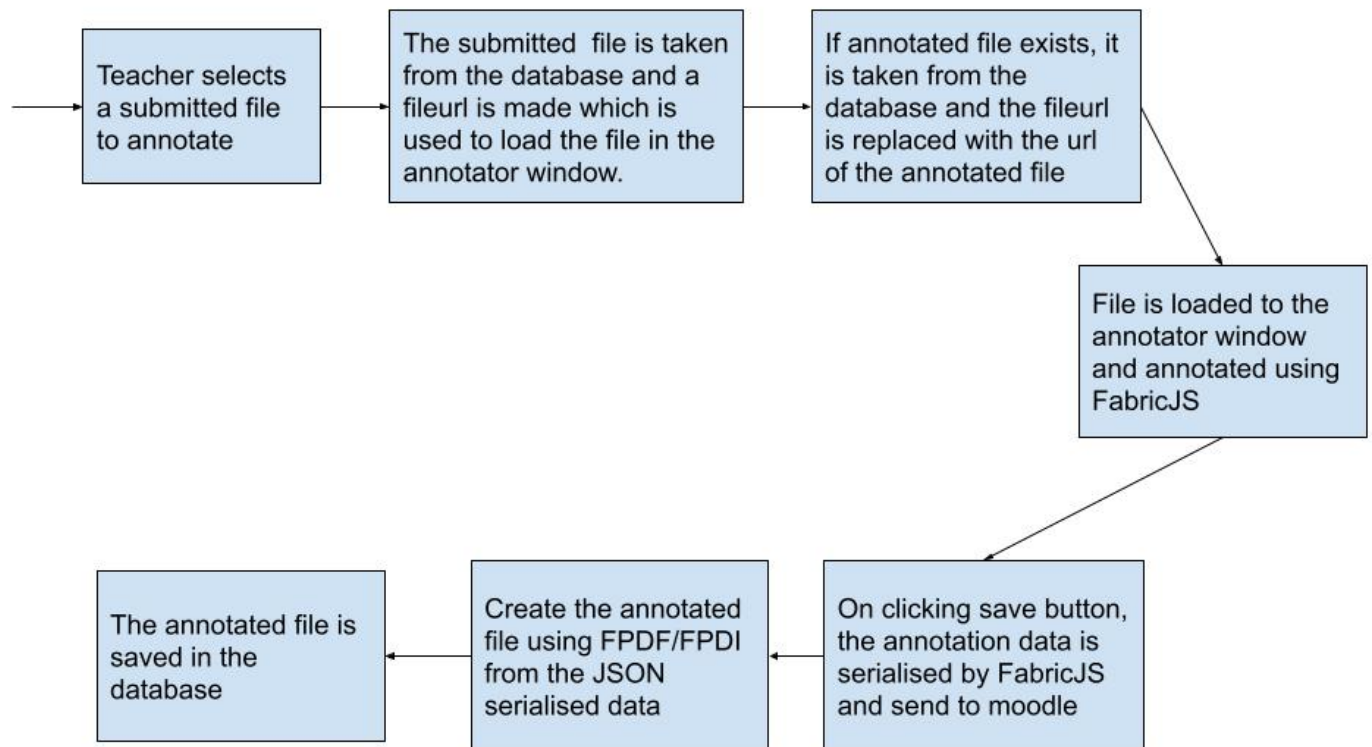
The file structure and the flow of control are described briefly in the previous chapter. This chapter discusses in detail, the purpose of each of the files and the important functions in them. A brief workflow of the annotation process can be seen in Fig. 6.1

### 6.1 Loading the annotator window

When the annotate button is clicked, the file `annotator.php` is loaded.

```
$PAGE->set_url('/mod/quiz/annotator.php', array('attempt' => $attemptid,  
'slot' => $slot));
```

In case of multiple attachments, it identifies the correct file as selected by the teacher. It loads the file to be annotated from Moodle file storage using the File APIs. If the selected file was previously annotated, the last annotated file is loaded onto the annotator window. It also has the necessary functions to check if a file type is supported and to convert compatible file types to PDF formats. It also creates a copy of the file to be annotated since it is required by FPDF/FPDI for the creation of the final output file. Moodle recommends using the



**Fig. 6.1** Workflow of the annotation process

temp folder present in the moodledata folder for the handling of temporary files. The moodledata folder has the necessary permission set for creating and accessing data stored within it. We have created a folder **EssayPDF** inside this temp directory of moodledata for all the file handling related to the PDF Annotator. The workflow of `annotator.php` can be seen in Fig. 6.2

```
//The $tempPath is the path to the subdirectory  
//EssayPDF created in Moodle's temp directory  
$tempPath = $CFG->tempdir . "/EssayPDF";
```

Finally, it loads the annotator window, by invoking `index.html` file. It also populates some of the JavaScript variables that are to be used in `clickhandlers.js` and `pdfannotate.js`

```
//If there are no exceptions we can proceed to the annotator window  
if($scanProceed == true)  
{  
    // include the html file; It has all the features of an annotator  
    include "/index.html";  
}  
?>
```

## 6.2 Supporting multiple file types

Our PDF annotator requires the input file in PDF Format. In Moodle essay-type questions, multiple file types are supported. In order to be able to annotate different types of files, the files have to be first converted to a PDF. Then the converted file has to be stored in the Moodle database and the URL of the converted file has to be passed on to the annotator window instead of the URL of the original file. This part of the PDF annotator had been done by Vishal Rao and Tausif Iqbal last year.

### 6.2.1 Imagemagick

Imagemagick is a free open-source tool that allows the editing and manipulation of digital images. It also supports the conversion of multiple file formats like jpg, jpeg, png, and txt. The following steps have to be done to include Imagemagick in the PDF annotator.

- Install Imagemagick in Ubuntu using the following command

```
sudo apt install imagemagick
```

To support the conversion of images using imagemagick, a line has to be changed in the policy.xml file. This file can be found here:- “/etc/ImageMagick(Version number)/policy.xml”. Change the following line

```
<policy domain="coder" rights="none" pattern="PDF" />
```

to

```
<policy domain="coder" rights="read|write" pattern="PDF" />
```

- For testing the working of Imagemagick, try converting an image file and a text file into PDF.

```
convert file_name.file_type converted_file_name.pdf
```

### 6.2.2 Using Imagemagick in PHP

Using the command we can execute the commands of Imagemagick via the shell and get its output. We will create a copy of the original file in the **EssayPDF** folder present in the Moodle temp directory. After executing the Imagemagick command and getting the converted file in the **EssayPDF** directory, we would delete the original file since it is no longer required.

```
if($mime === "image")  
  
    $command = "convert " . $fileToConvert . " " . $dummyFile;
```

```

else if($mime=="text")
{
    $command = "convert TEXT:" . $fileToConvert ." " . $dummyFile;
}
else
{
    $scanProceed=false;
    throw new Exception("Unsupported File Type");
}

```

## 6.3 PDF Annotator UI

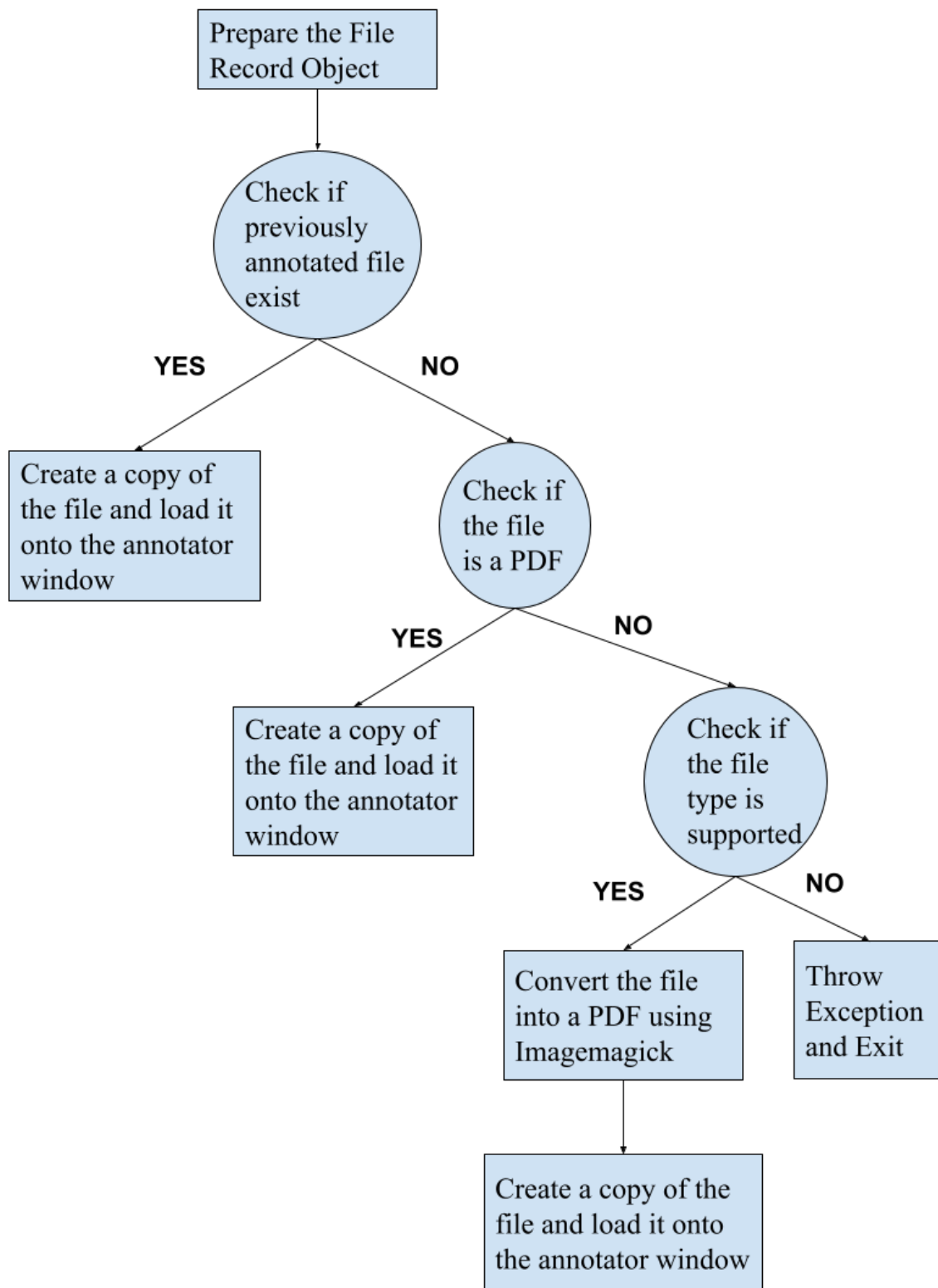
The files `index.html`, `clickhandlers.js`, and `pdfannotate.js` constitute the UI and the functions called by the UI for the PDF annotator. `index.html` has the necessary icons and buttons present in it. Each button click is a function call to `clickhandlers.js`. `clickhandlers.js` receives the `fileurl` from `annotator.php`. It is responsible for showing the file to be annotated in the UI. It also acts as an interface between `index.html` and `pdfannotate.js`. On every button click, when the `index.html` calls `clickhandlers.js`, it calls the corresponding functions in `pdfannotate.js`.

### 6.3.1 pdfannotate.js

All the FabricJS functions are invoked from `pdfannotate.js`. It converts each page of the file to be annotated as a Fabric Object and lets the user draw and edit over it using different FabricJS functions. There are functions for freehand drawing, drawing a rectangle, and inserting text.

When a user has finished annotation, he can click the Save button. On clicking the save button, all the drawings and texts over the PDF are serialized using the FabricJS serializer.





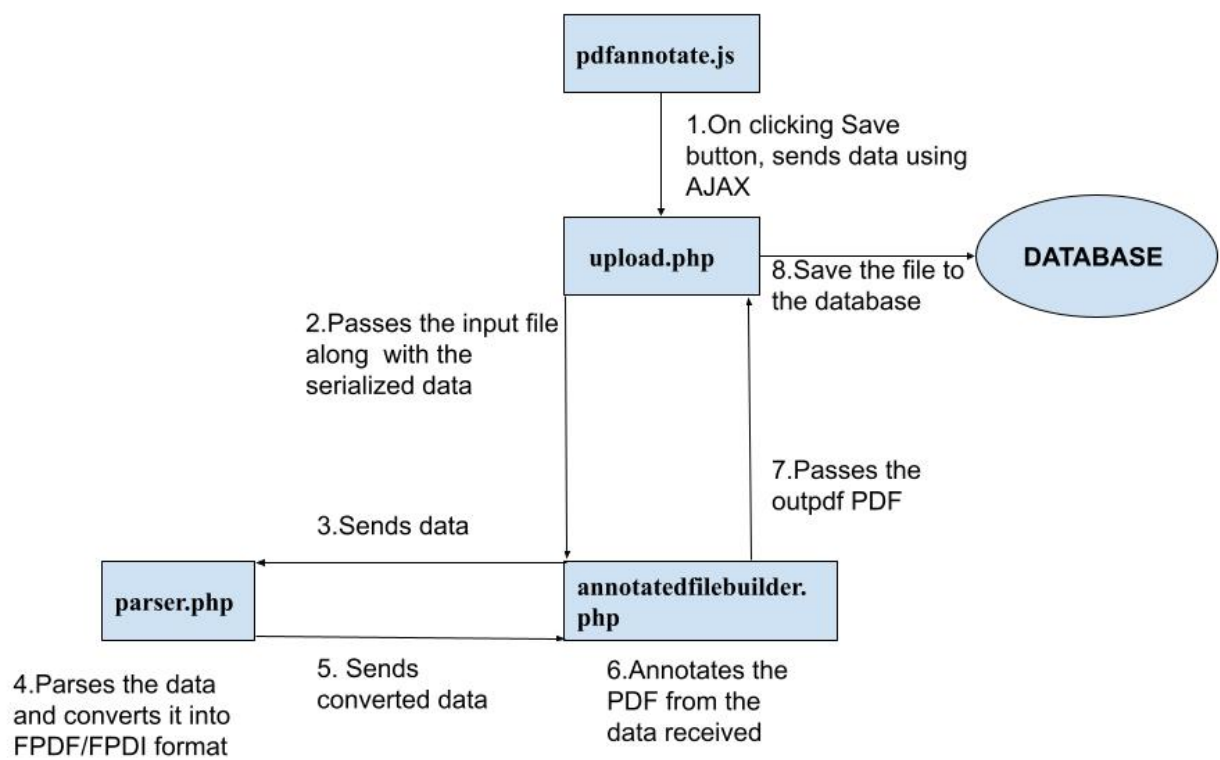
**Fig. 6.2** Workflow of `annotator.php`

In the `serializePdf` function, we have created a list corresponding to each page and have pushed the objects present on the page to the list corresponding to that page. After serialization, we will get the serialized data in JSON format. We send this data along with other necessary variables like `fileurl`, `filename`, `contextid` (which we will get from `annotator.php`) which are required to create the output file to `upload.php` in the form of an AJAX request.

```
xmlhttp.send("id=" + value + "&contextid=" + contextid +  
    "&attemptid="+attemptid + "&filename=" + filename + "&furl=" + furl);
```

## 6.4 Annotating the file and uploading to database

After getting the serialized data (that represents the annotations done in the UI) along with the required parameters for getting the file location from the database, the `annotatedfilebuilder.php` is called where the serialized data is passed and an FPDF object is created. Since the serialized data is of a format compatible with FabricJS, the function in `parser.php` is called, so that the data is converted to the format supported by FPDF/FPDI, and these data are stored in arrays. These arrays are used in `annotatedfilebuilder.php` for editing the FPDF object corresponding to the annotations in the UI. The FPDF object created is an AlphaPDF [15] class object. The AlphaPDF is an extended class on FPDF that is used for setting transparency. This class used is defined in the `alphapdf.php` taken from their official website. After the AlphaPDF object is fully edited, this object is returned to `upload.php` where the object is converted to a PDF and this PDF is stored in the EssayPDF folder created in the temp folder of the moodledata directory. The file record object is created using this output file after accessing the file storage of Moodle, and the file is stored in the database.



**Fig. 6.3** Uploading and annotating the file

# Chapter 7

## Error handling, UI bug fix and testing

This chapter focuses on the handling of multiple errors we encountered during testing and the support of the PDF annotator for multiple versions of Moodle.

### 7.1 Error Handling

#### 7.1.1 PDF Version above 1.4

The free parser provided by FPDF only supports PDF versions up to 1.4. Even if we were able to annotate the file in the UI for PDFs having versions above 1.4, the annotations were not present in the output PDF. So we decided to check the version of the PDF before creating the output file. If the version is above 1.4, we will convert it to version 1.4.

#### Ghostscript

Ghostscript is an interpreter for the PostScript language. It can be used to convert PDF files to PostScript and vice-versa. It also has the ability to convert one PDF version to another. In `upload.php`, we are initially checking the version of the PDF file and converting

it to 1.4, if the version is above 1.4. In case Ghostscript is not installed in the user's system, the user will not be able to save the file with the annotations when the PDF file version is above 1.4.

```
$line_first = fgets($filepdf);
preg_match_all('!\d+!', $line_first, $matches);
// save that number in a variable
$pdfversion = implode('.', $matches[0]);
if($pdfversion > "1.4")
{
    $srcfile_new=$path."/newdummy.pdf";
    $srcfile=$file;
    //Using GhostScript convert the PDF version to 1.4
    $shellOutput = shell_exec('gs -sDEVICE=pdfwrite -dCompatibilityLevel=1.4
    -dNOPAUSE -dBATCh -sOutputFile="'. $srcfile_new.'" "'. $srcfile.'"');
    if(is_null($shellOutput))
    {
        throw new Exception("PDF conversion using GhostScript failed");
    }
}
```

### 7.1.2 Lack of permissions to directories

We were using multiple temporary files for various purposes like a copy of the original file, the output file before storing it in the database, and files for conversions. We initially stored these files in the present working directory i.e. inside *mod/quiz* in Moodle. Generally, the Moodle files are stored in directories like *var* and *opt* that require certain permissions. Since initially we were running in our local systems and since we had root privileges there were no issues. But the user running Moodle may not always have the root privileges and in

that case, our annotator will fail since the temporary files could not be created due to the lack of proper permissions.

### **Moodle temp directory**

Moodle recommends using the temp directory present in Moodle's data root, to store and manage temporary files. During the installation of Moodle, this directory is given the required permissions to handle temporary files. We have created an additional directory inside the temp directory of the moodledata folder to handle the temporary files created related to PDF annotation.

During installation, if a user does not give the required permission to this directory or the user gives the wrong path to the directory when prompted, an exception is thrown which says the permission is denied.

### **7.1.3 PHP 8 Compatibility**

Some of the methods and practices that worked in a lesser version of PHP, resulted in an error while using PHP 8.

#### **count() method**

The `count()` method used in `annotatedfilebuilder.php` is used for getting the count of objects present in an array. The new feature of PHP required explicit conversion of the variable to an array before applying the count method.

```
if(count((array)$currPage) ==0)
    continue;
$objnum=count((array)$currPage[0] ["objects"]);
```

## **end() method**

The `end()` method is used in line with the `explode()` method at various places in `annotator.php`. The new feature requires the parameter of the `end()` method to be passed as a reference which requires an actual variable rather than the result of `explode()` being directly passed to the `end()`. Splitting it into two lines solved this issue.

```
$format = explode(".", $filename);  
$format = end($format);
```

## **preg\_match()**

The usage of the `preg_match()` method in `parser.php` was one of the main sources of error. The parameter passed to `preg_match()` for matching was not compatible with the new version of this function. Removing it and checking it manually using if statements solved this error.

```
//This is the previous version that resulted in an error in PHP 8  
if (preg_match('/rgb/', $str))  
{  
    $str = substr($str, 5,-1);  
    $val = explode(",", $str);  
}
```

## **7.2 Bug Fixes in the UI**

### **7.2.1 Translation of freehand drawings**

We noticed that even though we could translate a freehand drawing in the UI, the same translation was not reflected in the serialized data we get. For freehand drawings, the data is serialized by considering the drawings as very small lines. So the serialized data will be

a set of points. But even on translation, the coordinates of these points were not changing. Therefore, the translation of freehand drawings was not reflected in the final annotated PDF.

### Transformation matrix

A transformation matrix is a matrix that, through matrix multiplication, transforms one vector into another. The transformation matrix modifies the cartesian system and maps the vector's coordinates to the new coordinates. The 3 functions with which we can get the transformation matrix of an object is

- `obj.calcTransformMatrix()`
- `obj._calcTranslateMatrix()`
- `obj.calcOwnMatrix()`

We are using `obj.calcTransformMatrix()` function since we found it more accurate. We have to pass the original points and the transformation Matrix as input to the transform function in FabricJS to get the new points. We have to subtract the offset from these points to get the translated points.

```
var matrix=pathObj.calcTransformMatrix();  
var newPoints= fabric.util.transformPoint(points, matrix);
```

#### 7.2.2 Active tool fix

Originally while the rectangle or text is the selected tool, after an object is created, the selected tool is retained. This means a user can draw the object that is selected again and again without having to click on the tool icon again. But this resulted in a bug which is while the area of the new object creation falls inside that of the previously created object, the previous object is selected as well as a new object is created at the same time. To fix



this bug, the active tool is set to the select tool every time a rectangle or text object is created. So the user needs to click on the tool icon again to create an object again.

### 7.2.3 Tool icon highlight fix

When the tool icon is automatically set to select without the user explicitly clicking on select, the tool icon highlighted is the object creator before that. This is fixed in the `fabricClickHandler()` function in `pdfannotate.js` where after a rectangle or text object is created, the select tool is explicitly set as highlighted.

```
//Change the currently selected tool in the UI to select  
//if the active tool is a highlight box or text  
    if (inst.active_tool == 2 || inst.active_tool == 4) {  
        var element = document.querySelector("#select");  
        $(".tool-button.active").removeClass("active");  
        $(element).addClass("active");  
    }
```

## 7.3 Testing

The following are some of the main cases we tested for the PDF annotator.

- **Dummy file not created.**

The dummy file is the file before annotations, if this file is not created due to any reason like permission denied, the user is not allowed to annotate by throwing an exception.

- **Output file not created.**

If the dummy file is successfully created, and the output file is not created, the user is alerted with the 'not able to save file' message while trying to save.

- **PDF version greater than 1.4.**

This was tested and the Ghostscript handles this. In case the Ghostscript is not installed or the Ghostscript cannot successfully handle the case, an exception is thrown.

- **Both the dummy file and output file were created but failed to update into the database**

Moodle throws an exception in this case while the user is trying to save a file.

- **Unsupported file format** Moodle does not allow users to upload format that is not supported by Moodle's File APIs. Other formats like docx which is supported by Moodle but not supported by our developed PDF annotation tool throws exception error.

- **File size testing** The size of the file were tested for various file size and only a few KB increase was observed. The upper limit for file size can be set a)for a particular quiz by the faculty of the course b)in course property settings c)by the site administrator as the site upload limit d)PHP configuration settings in the `php.ini` file. It was tested for all these conditions with the file size matching the upper limit.

- **Working of Tools** All the tools provided by the UI such as freehand, highlight box, and text were tested whether they are working fine.

- **Testing selection and translation transformation features** The selection feature and translation after selection is tested for all tool objects. Transformation is locked for freehand and is tested for all other objects. It was also tested whether the currently highlighted object shown in the UI is correctly shown.

- **Deletion of object** All the objects created in the current annotation window can be selected and deleted and these workings were tested.

- **Comparing file after annotation with the file shown in the UI** It was tested for all objects whether the location, color, transparency, and size were matching for

the object shown in the UI by FabricJS and for the objects present in the file after annotation by FPDF/FPDI.

## 7.4 Installation and version support

Our PDF annotator supports Moodle versions 3.11 - 4.2 for PHP versions 7.4 and 8.

The following packages are necessary for the PDF annotator to work.

- Ghostscript
- Imagemagick

We have created a Makefile for the ease of installation of the annotator. There are 2 folders inside the repository. The folder **common** contains files that are version independent and there are folders for versions 4.0, 4.1, and 4.2. Moodle version 3.11 can use the files inside 4.0. Makefile has 2 targets

- **quiz\_annotator**: This is used for installing the PDF Annotator.

```
sudo make quiz_annotator MOODLE_VERSION=x.x MOODLE_DATA_DIR=y
```

The default Moodle version is **4.1** and the default data directory is **/var/moodle-data**

- **restore**: Used to delete files used for PDF annotation and restore the old files. This is done with the help of a backup folder which is created at the time of installation.

```
sudo make restore
```

# Chapter 8

## Conclusion and Future Work

### 8.1 Conclusion

We worked on creating a PDF annotation feature for Moodle’s quiz module for the essay-type question. We prioritized working on a feature that does not convert the PDF into an image thereby retaining the properties of the PDF including the selection of the text and importantly the final size of the annotated PDF. We developed on the work done by Tausif Iqbal and Vishal Rao [2]. Their work used the Fabricjs library where the PDF was converted into an image and annotations were added to it. Upon saving, this image was converted into a PDF and uploaded to the database. We changed this method so as to fix the issue of file size blow-up. We used their front-end code for the UI and extracted the annotation values for producing the final output PDF. The annotation values were obtained by creating a `serializepdf()` function written as part of the javascript file. This function takes the stored fabric object and converts necessary details into JSON format to be used by the back end. We explored various PDF annotation libraries and the existing annotation feature of the assignment module for the back end. The idea of using the latter was discarded since the `yui.js` used in the same was no longer maintained. Out of the libraries explored that involved annotating features without the PDF being converted into an image, we found FPDF/FPDI to be the most suitable and well-maintained for our use

case. The earlier said annotation JSON data obtained after serializing is processed and converted into a data array suitable for creating FPDF/FPDI objects. This data array is then used to create the FPDF/FPDI object that is used to finally output the annotated PDF. This PDF is stored in the database. We made slight changes in the UI to make it seamless with the back-end such as locking some transformation movement which was not a concern for the work done by last year's since it was based on converting the pages to images. We also changed the function call invoked during the save button being clicked which is necessary for the back-end changes explained so far.

We tested our tool for Moodle versions 3.11 to 4.2 for PHP versions 7.4 and 8.0. We fixed the bugs that arose in PHP version 8.0 due to changes in features in PHP functions. We also fixed the bugs identified in the UI used before. The deliverables of the project, including the source code, Makefile for easy installation, and supporting documentation are made available in GitHub[8].

## 8.2 Future Work

Our PDF annotator consists of a combination of PHP and Javascript files. Migrating as much code to PHP from Javascript for improved safety is one of the main future works. While doing this, it may be still required to have some Javascript code because the user experience will depend on the response time of the interface while annotating the document.

Making the PDF annotator as a plug-in is another useful future work since it ensures modularity and the site administrator can enable or disable the plugin according to the use case. It would be ideal if the same plugin can be used in the assignment module and any other module that needs a PDF annotation feature.

We are planning to contact the Moodle code maintainers for integration of the PDF annotator into Moodle. It is expected that they may suggest better standardization of the code so as to comply with Moodle's coding conventions and follow a more systematic testing procedure. This can be planned as a starting point for potential future work.

# References

- [1] Moodle Pty Ltd. Moodle Open Source Learning Platform. <https://moodle.org/>, 2023. [Online; accessed 13-May-2023].
- [2] Tausif Iqbal and Vishal Rao. GitHub Page: Quiz Annotator Plugin that Enables Teacher to Annotate Essay Type Question in moodle. [https://github.com/TausifIqbal/moodle\\_quiz\\_annotator](https://github.com/TausifIqbal/moodle_quiz_annotator), 2021. [Online; accessed 13-May-2023].
- [3] Ravisha Hesh. GitHub Page: Wrapper for PDF JS to Add Annotations . <https://github.com/RavishaHesh/PDFJsAnnotations/>, 2023. [Online; accessed 13-May-2023].
- [4] Kristian Høgsberg. Poppler PDF Rendering Library. <https://poppler.freedesktop.org/>, 2023. [Online; accessed 13-May-2023].
- [5] Behdad Esfahbod and Carl Worth. Cairo Graphics Library. <https://www.cairographics.org/>, 2023. [Online; accessed 13-May-2023].
- [6] Setasign GmbH & Co. KG. FPDF Free PDF Document Importer. <https://www.setasign.com/products/fpdf/about/>, 2023. [Online; accessed 13-May-2023].
- [7] Olivier Plathey. FPDF library. <http://www.fpdf.org/>, 2020. [Online; accessed 13-May-2023].

- [8] Parvathy S Kumar and Asha Jose. GitHub Page: PDF Annotator for the Quiz Module of moodle. [https://github.com/Parvathy-S-Kumar/Moodle\\_Quiz\\_PDF\\_Annotator](https://github.com/Parvathy-S-Kumar/Moodle_Quiz_PDF_Annotator), 2023. [Online; accessed 13-May-2023].
- [9] Moodle Pty Ltd. GitHub Page: Moodle - the world's open source learning platform. <https://github.com/moodle/moodle>, 2023. [Online; accessed 13-May-2023].
- [10] Moodle Pty Ltd. Moodle 4.1, Moodle Developer Resources. <https://moodle.org/>, 2023. [Online; accessed 13-May-2023].
- [11] Denis Auroux. Xournal. <https://xournal.sourceforge.net/>, 2023. [Online; accessed 13-May-2023].
- [12] Moodle Pty Ltd. Moodle-Improve Annotation - MoodleDocs. [https://docs.moodle.org/dev/Improve\\_Annotation](https://docs.moodle.org/dev/Improve_Annotation), 2022. [Online; accessed 13-May-2023].
- [13] Juriy Zaytsev, Stefan Kienzle, and Andrea Bogazzi. FabricJS Documentation. <http://fabricjs.com/>, 2023. [Online; accessed 13-May-2023].
- [14] Moodle Pty Ltd. Moodle 4.1 User Documentation, File API - MoodleDocs. <https://moodledev.io/docs/apis/subsystems/files>, 2023. [Online; accessed 13-May-2023].
- [15] Martin Hall-May. AlphaPDF - Transparency Support for FPDF. <http://www.fpdf.org/en/script/script74.php>, 2023. [Online; accessed 13-May-2023].