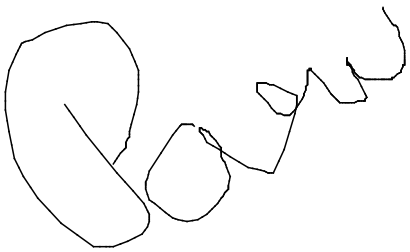
Experiments with moodle - Plugin development/enhancements

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 "**Experiments with moodle - Plugin development/enhancements**" is a bonafide work of **Parvathy S Kumar & Asha Jose** (**Roll No. 111901040 & 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

List of Figures

Chapter 1

Introduction

Moodle is an open-source software that is widely used as a learning management system or LMS. A Learning Management system is software that assists in implementing and assessing any learning process. 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.

The word Moodle was originally an acronym for Modular Object-Oriented Dynamic Learning Environment. Moodle is the learning management software used in IIT Palakkad. It supports multiple features that enhance the process of remote learning thereby making the process of online learning organized and simpler

1.1 Objective

The main objective of our project is to familiarize ourselves with the process of plugin development in moodle and create a plugin for pdf annotation for the quizzes that do not unreasonably increase the size of the annotated pdf. We also further aim at generalizing the plugin to be used for both quizzes and assignments if possible in the future.

There is already a built-in pdf annotation for assignment submission, but we don't desire to integrate this into the quiz submission file for it is not preferred and because we would like to create a plugin instead of integrating it into the quiz module. The annotated pdf is stored as layers, one layer containing the original pdf and the subsequent layers containing the further annotations.

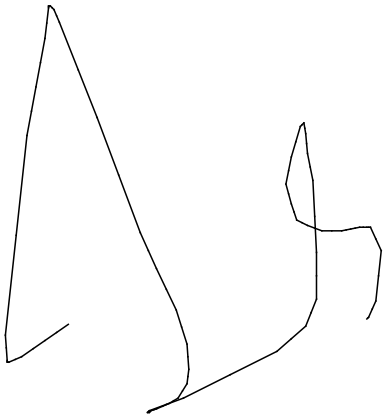
1.2 Scope of the project

The main issue we aim at tackling is maintaining the appropriate size of the file and this involves not converting the different layers in the annotated file to an image before it is finally produced as a PDF. A text document is generally very small compared to an image. 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. Our idea is to store these layers as vector graphics and integrate them directly into the PDF format.

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 roles in the next chapter. In Chapter 3, we have explained the plugin development process in detail along with some sample plugins we developed in order to familiarise ourselves with the process. In Chapter 4, we have briefly discussed the different annotation tools and libraries including Xournal, its libraries, and FPDF/FPDI . In Chapter 5, we have discussed about the new plugin which we developed using FabricJS and FPDF/FPDI. Chapter 6 deals with

the integration of the new PDF Annotation tool with the moodle quiz module. Finally, in Chapter 7, we have briefly discussed the works done till now along with the future works.



Chapter 2

Setting up of Moodle and its Important features

Moodle is an open-source software and its source code can be found at `https://github.com/moodle/moodle`. We are using the latest stable version of moodle - moodle 4.0.4

2.1 Requirements

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

- Apache2.4.41
- MySQL 8.0.30
- PHP 7.4.3

2.2 Moodle Database

The current stable version of moodle has a total of 469 tables. Some of the most relevant tables and some of their relevant fields are discussed below

2.2.1 Course and Enrolments

mdl_course

This table contains most of the information about the courses including the id, fullname & shortname of the course, startdate and enddate of the course, and the description of the course if any. This table is populated with the necessary information regarding a course whenever a new course is created

mdl_enrol

This table consists of the information about the enrolment id, course id to which the enrolment was done, type of enrolment, and the time at which a particular enrolment was done. Whenever an enrolment is done, the table is populated with the necessary values

2.2.2 Users

mdl_user

This contains information about users. It includes the id(primary key of the table), first name, last name, email id, password in an encrypted format, the address, the first login, the last login, and the last access time of the user. All the details taken about the user on the addition of the user into moodle are included in this table

mdl_user_lastaccess

This contains the information about the time of access to a course by a user and hence the important fields of this table are the id(primary key of this table), userid which is the id of the user who accessed a course, and courseid which is the course which the user accessed. Whenever a user visits a course this table is updated with the userid, courseid, and time at which the user accesses this course

mdl_user_enrolments

This table, like the enrol table consists of the enrolment details of users. It consists of the id(the primary key of this table, the enroldid(which is the primary key of the enrol table), and userid (which is the primary key of the user table). This table has the map between the userid and enrolid. That is we get the information of the user corresponding to an enrolment using this table.

2.2.3 Quizzes

mdl_quiz

This table consists of the information about the quiz. This table is populated whenever a quiz is added. It consists of the id(primary key of the table), name of the quiz, course to which it belongs to, time of opening and closing of the quiz, password of the quiz, and the maximum grade of the quiz.

mdl_quiz_grades

This table has information about the grading of each quiz. It has the id(primary key), quizid, userid, and the time at which the grade was modified. This table stores the grades of users for each quiz along with the grading time.

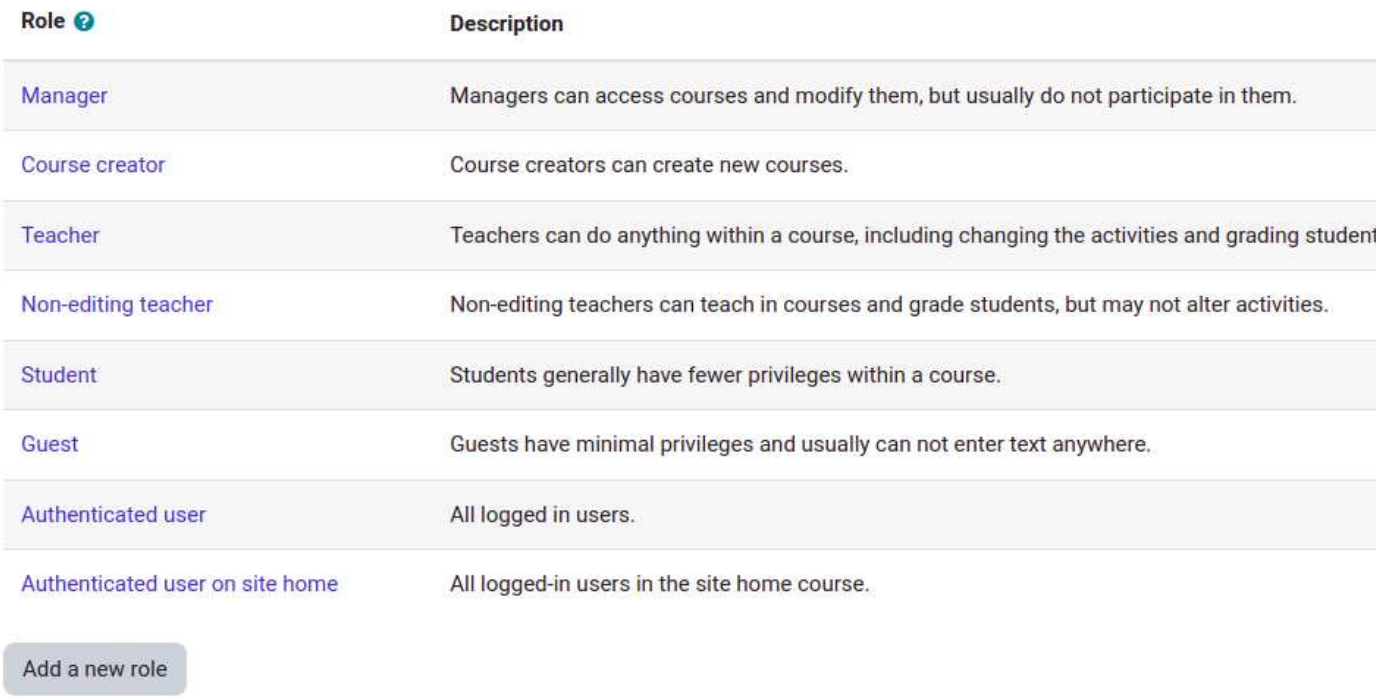
2.3 Users and Roles

A role is a system-wide defined collection of permissions that can be assigned to a specific user in a specific context.
Some of the permissions given to a role include "prevent", "allow" and "prohibit".
A context is a part of moodle which can include a user, a course, the whole site, or a particular block.

2.3.1 Different roles in moodle

The Standard roles in moodle can be identified from the figure in moodle. Site Administrator is the role having all permissions and can do anything on moodle.

Apart from the role of Site Administrator and the other roles listed in the figure, other roles can be manually created and only the users having administrative privileges will be able to do so.



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 student
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

Chapter 3

Plugin Development

Plugins are methods that add different functionalities to the original software without altering them. Moodle offers modularity and extensibility, and this is primarily achieved through the development of plugins, hence in order to add any functionality to moodle, we can use plugins.

3.1 Plugin Types in Moodle

There are vast varieties of plugin types in moodle. Some of the important and useful plugins are discussed below.

- **Activity modules**: These are important types of plugins that provide different activities in moodle. Such plugins have to be written in the */mod* file in moodle. Quiz, Assignments, Forums etc are different kinds of activity modules present in moodle.

- **Assignment submission and Assignment feedback plugins**: These plugins are used to add different forms of assignment submissions and feedback. These are contained in the moodle path */mod/assign/submission* and */mod/assign/feedback* respectively

- **Quiz reports**: It is used to add miscellaneous features to the quiz module and different methods to analyze and display the quiz results. These types of plugins are added to the moodle path */mod/quiz/report*.

- **Block**: These are small and useful pieces of information that can be moved to different positions in moodle. The list of online users and the pending activity list are examples of block plugins. Such plugins have to be added in the moodle path */block*

- **Course formats**: These are different methods for displaying activities and blocks in a course. These types of plugins are present in the moodle path */course/format*

3.2 Common files in 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/plugintype_pluginname.php**: There must be an English translation for each language string defined by a plugin. These are specified in a file named after the plugin in the plugin's lang/en directory. This is a mandatory file for every plugin.

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

3.3 Sample Plugins

3.3.1 Plugin to list the available courses for enrolment

This plugin is used to display the list of available courses that the user is still not enrolled in. This is a block-type plugin and can be displayed in the dashboard of moodle. We are displaying this information on the right side of the dashboard of moodle. This plugin is developed using the help of 3 tables of moodle namely, course, enrol and user_enrolments. After joining these tables using the necessary fields, we will retrieve the name of the course and the userid information. We will get the name of the course from the *course* table and userid from the *user_enrolments table*. The *enrol* table contains the enrol id along with the courseid while the *user_enrolments* contains the enrolid along with the userid. On joining these 2 tables, we are actually getting a list of the userid of users along with the courseid of courses these users are enrolled in. We will get the information of the currently logged-in user using the $USER variable in moodle. Using the ID of $USER, we can get the list of courses that the currently logged-in user is enrolled in. We can get the available courses the currently logged-in user can be enrolled in by taking the difference of this course list from the total courses list. 3.1

3.3.2 Plugin to list the online users who have at least 1 course in common with the logged-in user

Moodle has an existing plugin that will display the list of online users regardless of whether the user is relevant to the logged-in user or not. This plugin aims at 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. This is also a block-type plugin and is displayed on the right side of the dashboard. This plugin uses the *user_lastaccess* and the *user* tables of moodle. Using the method used in the previous plugin, we will get the list of courses the currently logged-in user is enrolled in. The *user_lastaccess* table has the fields userid, courseid and

timeaccess which is actually giving the last accessed time of a user in a particular course. The time is in the UNIX epoch time format. The Unix epoch is the number of seconds that have elapsed since January 1, 1970, at midnight. With this information, we can get the unique userid of users sharing at least one course with the user and accordingly get the full name of the user which is displayed in the plugin. 3.2

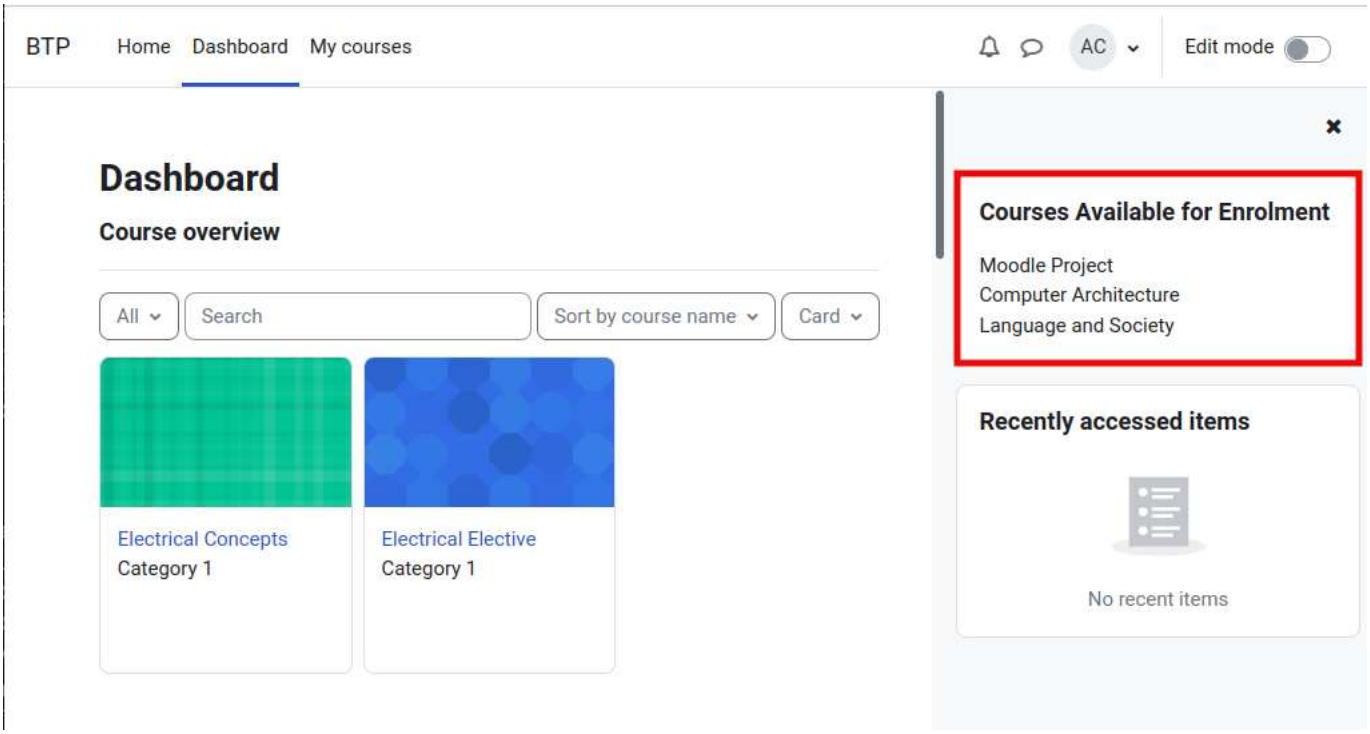


Fig. 3.1 Plugin to list the available courses for enrolment

3.3.3 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 travelling to pages. The files called edit.php and manage.php are created in the message folder in the local directory in moodle. From the edit page, a user can write a message and save it or cancel it and a user can go to the edit page by clicking on the 'create message' option in the dashboard. Both the creation and cancellation options take the user to the manage page where the message displayed is 'you created a message' or 'message cancelled' according to whether it's saved or cancelled

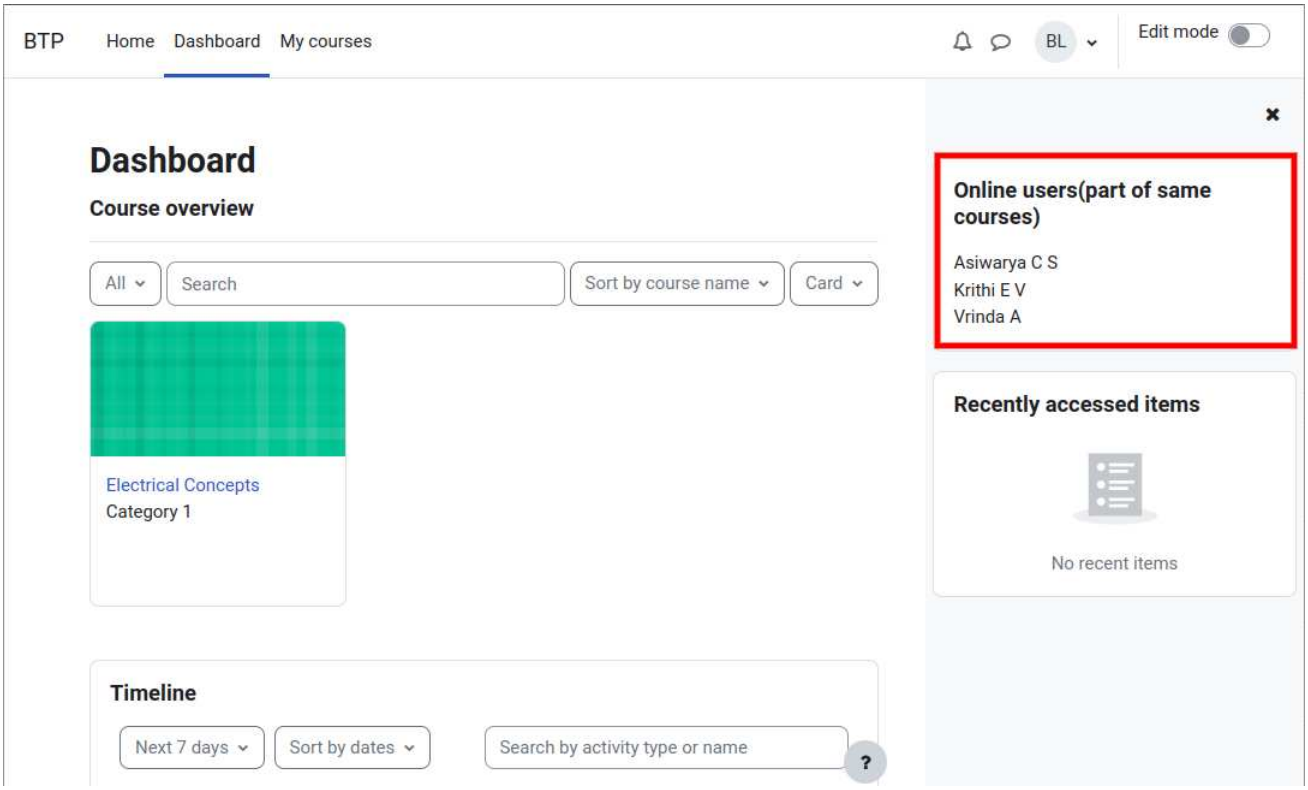


Fig. 3.2 Plugin to list those online users who share at least a course with the logged-in user

respectively. This page will also show all the messages created by the user so far. The 'view messages' option on the dashboard takes the user to the same page. The messages are displayed because they are updated in the database. The folder called classes in the local directory holds an install file that creates a local/message table in the database. The original moodle form is extended in the class folder in a file called edit.php inside the class and these messages are added by the insert/record function. The save and cancel buttons are added by moodle forms' action/buttons.

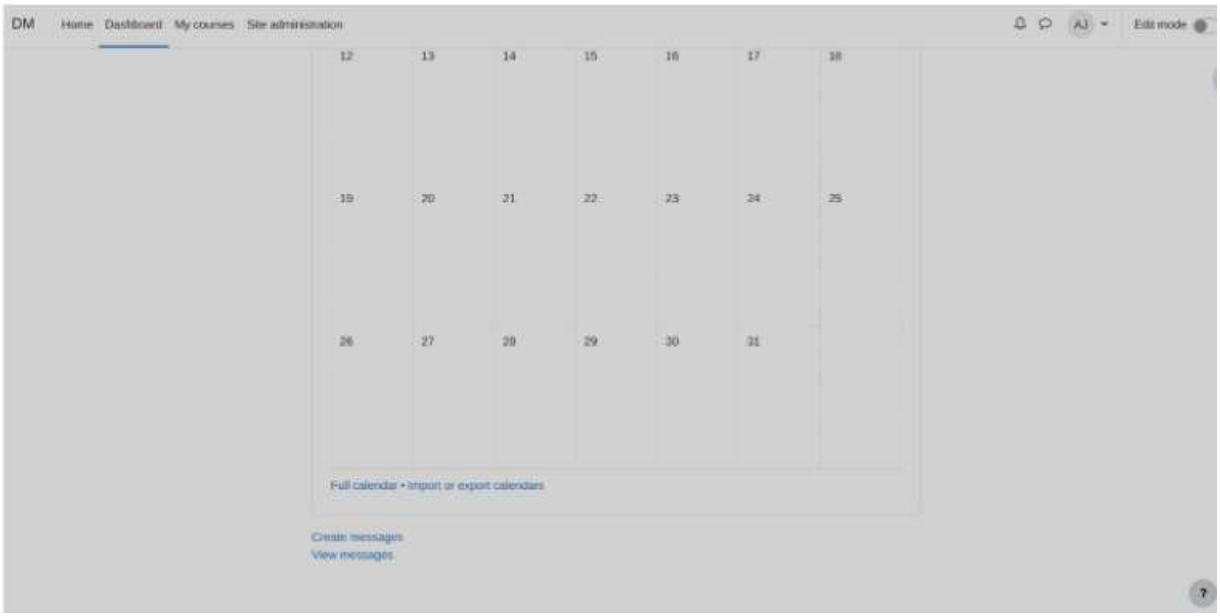


Fig. 3.3 Plugin to create the Message

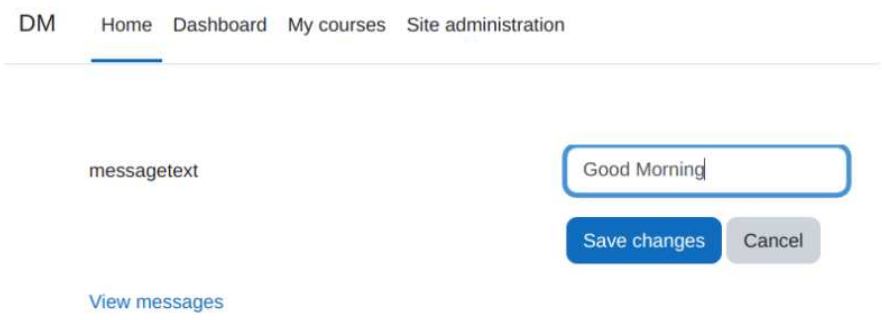


Fig. 3.4 Adding of pages for message creation

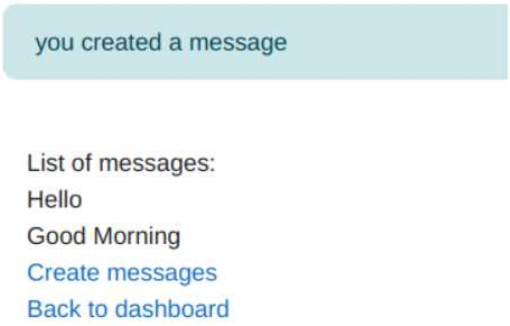


Fig. 3.5 Successful Creation of Message

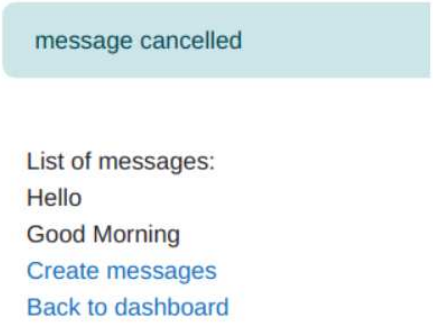


Fig. 3.6 Message is Cancelled

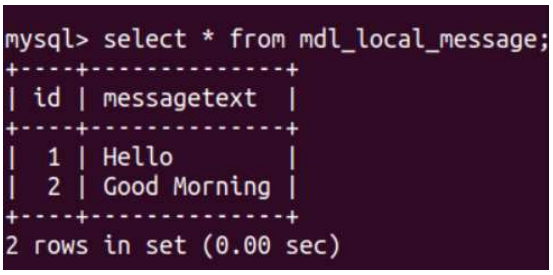


Fig. 3.7 Messages are added to the Database

Chapter 4

Exploration of Annotation Tools and Libraries

4.1 Overview of the work done last year

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. 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. Our project mainly aims at solving these issues.

4.2 Xournal

Xournal 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 like drawing, erasing, selecting, undoing, redoing, etc that are needed

for annotating a quiz file submission, making it ideal to integrate its logic to moodle for quiz submission file annotation. Since version 0.4.5, Xournal uses the Poppler library for annotating a pdf. Poppler uses Cairo and Splash in their back end. Cairo is a 2D graphics library that provides a vector-based implementation and Splash is a javascript rendering service.

4.2.1 Relevance to our project

The usage of the Poppler library 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. Since our aim is to generate a submission file annotation plugin that produces a file not much larger than the submitted file, this is apt. Another reason why Xournal is preferred for reference is that, in the latest version of Xournal (0.4.8), the export to pdf version is updated 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. We aim 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.

4.3 Poppler and Cairo

4.3.1 Workflow

On exploring Xournal, it was found that the input PDF was 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 were then added as new surfaces to this context. Opening a new PDF Cairo surface 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,

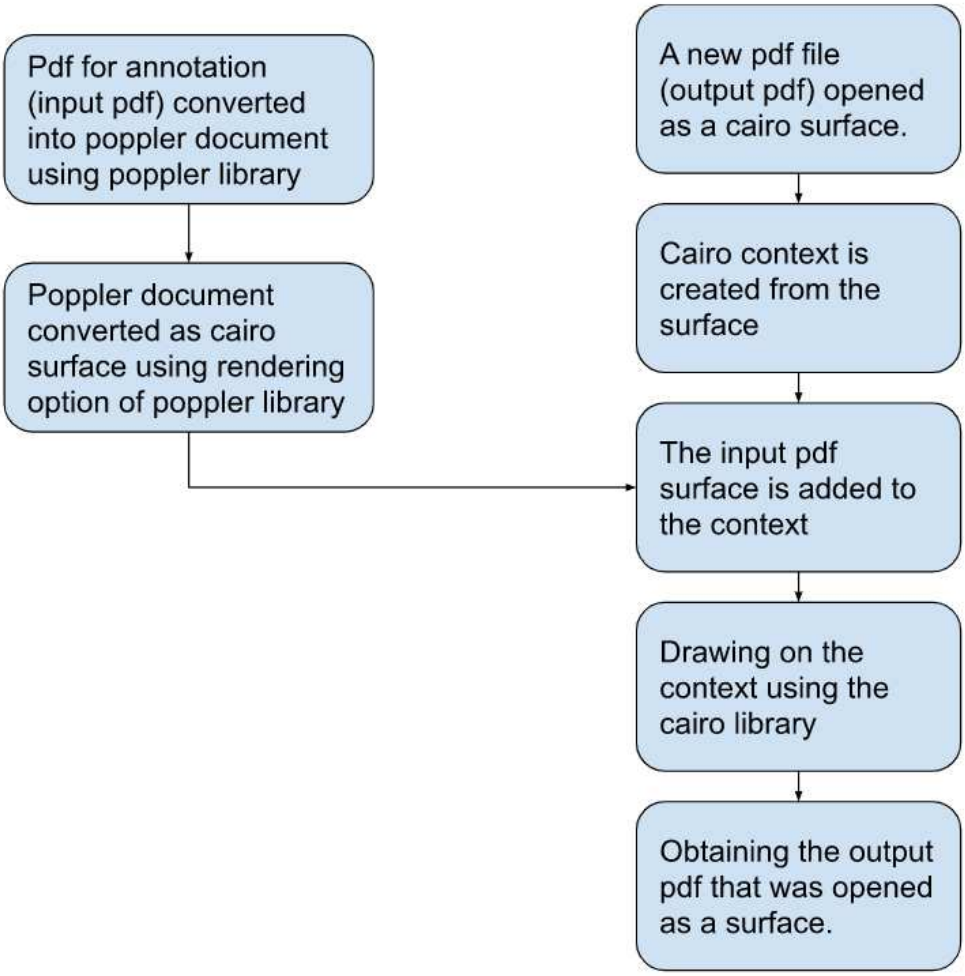


Fig. 4.1 Workflow of rendering PDF

annotations like drawing shapes are added to this context further. Upon compiling and running, the PDF file generated contains all the annotations made and the size is not much different from the original size of the PDF. 4.1 4.2

4.3.2 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 aren't stable. The Cairo library in PHP isn't a maintained version. Also, Poppler PHP doesn't 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 doesn't 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.

4.4 FPDF and FPDI

FPDF and FPDI together can be used to edit an existing PDF and add annotation layers over it. It 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.

4.4.1 FPDF

FPDF stands for Free Portable Document Format 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.

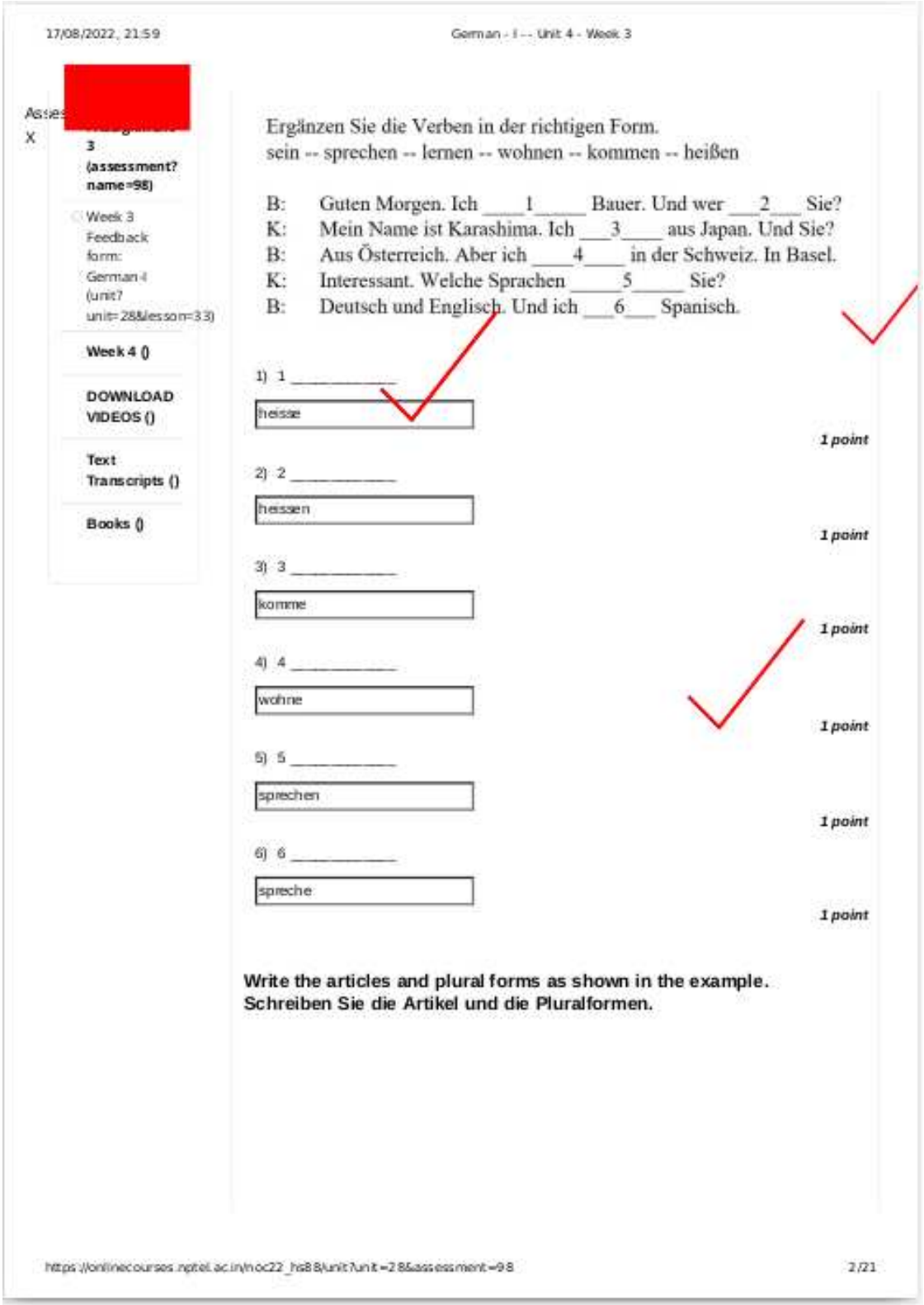


Fig. 4.2 PDF Annotated using Cairo and Poppler

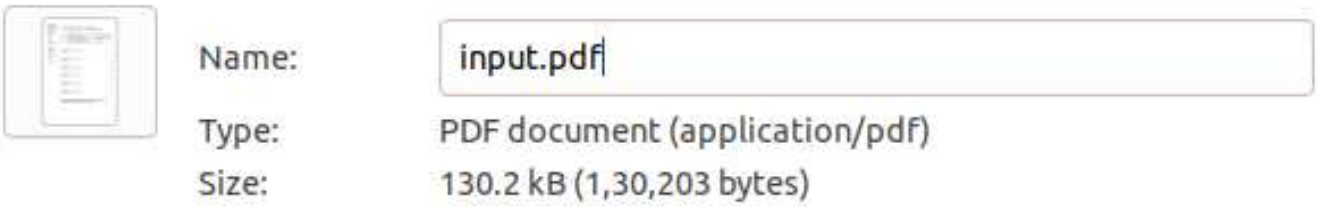


Fig. 4.3 Size of the input PDF

Fig. 4.4 Size of the output PDF

4.4.2 FPDI

FPDI stands for Free Portable Document Importer. 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.

4.4.3 WorkFlow

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 by students for annotation. Once we load the PDF, we can edit page-by-page, by using the **importPage()** function of FPDI 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 font, its size, and its color can be altered using different functions present in FPDF. It also supports adding images over PDF.

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 load the resultant PDF and load it in the database. 4.6 4.7

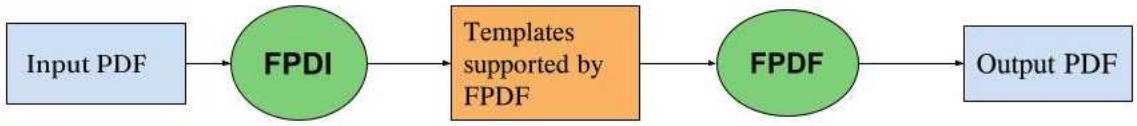
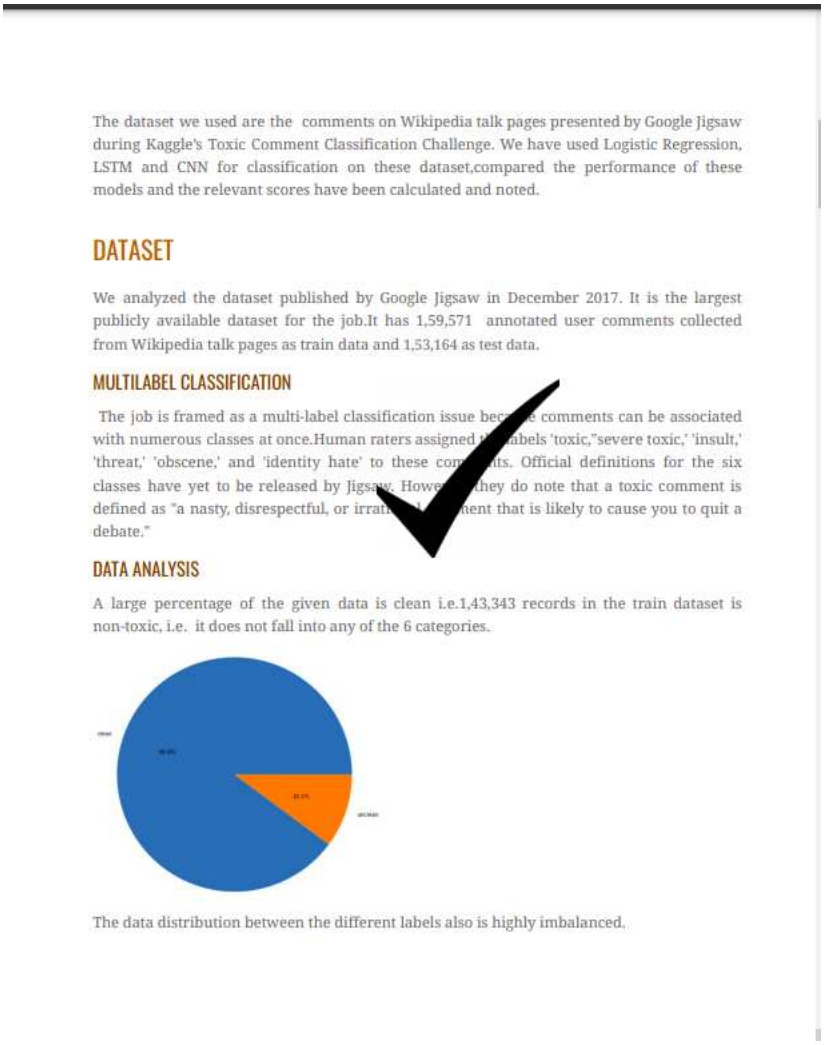


Fig. 4.5 Functioning of FPDF/FPDI

4.5 Integration to Moodle

Our initial approach is to add the PDF annotator to the Quiz module. The files related to Quiz are located in the 'mod/quiz/' directory in the moodle folder. For adding the above, it is required to retrieve the PDF from the database, annotate it using the said library, export it back to the database, and make provisions for displaying it. For annotating the PDF, it is planned to display a button for annotation, and clicking on it would pop up a window that allows a user to annotate the opened PDF. We plan to reuse the user interface



The dataset we used are the comments on Wikipedia talk pages presented by Google Jigsaw during Kaggle's Toxic Comment Classification Challenge. We have used Logistic Regression, LSTM and CNN for classification on these dataset,compared the performance of these models and the relevant scores have been calculated and noted.

DATASET

We analyzed the dataset published by Google Jigsaw in December 2017. It is the largest publicly available dataset for the job.It has 1,59,571 annotated user comments collected from Wikipedia talk pages as train data and 1,53,164 as test data.

MULTILABEL CLASSIFICATION

The job is framed as a multi-label classification issue bec e comments can be associated with numerous classes at once.Human raters assigned abels 'toxic,"severe toxic,' 'insult,' 'threat,' 'obscene,' and 'identity hate' to these com ts. Official definitions for the six classes have yet to be released by Jigsaw. Howe they do note that a toxic comment is defined as "a nasty, disrespectful, or irrati ment that is likely to cause you to quit a debate."

DATA ANALYSIS

A large percentage of the given data is clean i.e.1,43,343 records in the train dataset is non-toxic, i.e. it does not fall into any of the 6 categories.

The data distribution between the different labels also is highly imbalanced.

Fig. 4.6 Sample PDF with a tick image inserted over it

Fig. 4.7 Added lines and inserted text in a PDF

of the said window and the files needed to retrieve and integrate the PDF to be annotated from the project done by Vishal Rao and Tausif Iqbal, our last years'. As our first step, we plan to include our file as annotator.php and try some basic operations like saving the PDF to check the working.

Chapter 5

A new PDF Annotation feature using FPDF/FPDI

The PDF Annotator should have a User Interface where a user can add annotations over the PDF and on clicking a Save button the user should be able to save the annotations. 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.

5.1 Existing PDF Annotation tools and their drawbacks

5.1.1 Javascript plugin

As discussed in section 4.1, the javascript plugin uses 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.

5.1.2 Assignment Plugin

This 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. 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.

5.2 Workflow of the proposed Plugin

In the proposed PDF annotator plugin, the drawing over the PDF and displaying it while annotating is done using FabricJS 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 FPDI. Now we have the necessary inputs required by FPDI 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 FPDI 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.

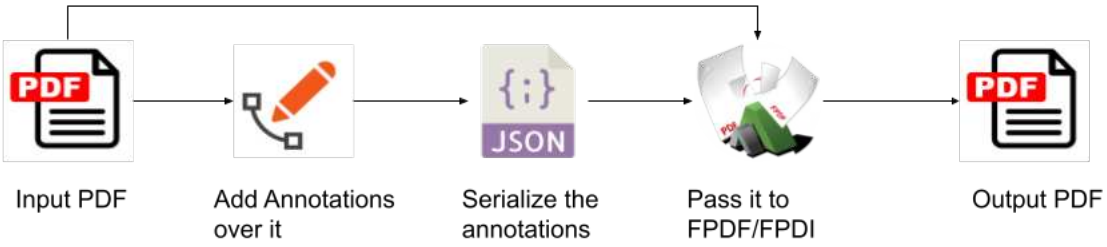


Fig. 5.1 Workflow of the PDF Annotator

5.3 Step-by-Step Explanation of the Workflow

5.3.1 User-Interface and Adding Annotations over a file

For the User Interface part of the plugin, we are using the same UI developed by Vishal Rao and Tausif Iqbal since it has all the necessary features needed for annotating a PDF. Only necessary changes that mainly include the function that is being called while clicking on the save button are changed. The main library required for the UI is FabricJS. The User Interface has tools for

- free hand drawing
- drawing a rectangle
- adding an image onto the file
- adding text over the file

- changing the stroke and fill color
- saving the annotated file

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 like 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.

5.3.2 Serialization of the annotations

We will 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 data of objects while saving is only present in the JSON file. So, if an object is being translated, the JSON file would not contain the previous data of the same object as the serialization function is called only while the save button is being clicked. 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 that needs to be extended. Some examples of serialization of objects are given below

- **Rectangle** - For a rectangle, 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 drawn in the file.

- **Image** - For an image, the serialized data include the location of that image in a local system or in the web, the coordinates of the top-left corner along with the width and height.

- **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.

- **Lines** - For lines, the serialized data include the x-coordinate and y-coordinated of the starting point and ending point along with the stroke-color and thickness.

- **Free-hand 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.

5.3.3 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 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 6

Integrating the PDF Annotation tool with the moodle quiz module

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

1. A teacher should be able to see the files submitted by the students in the annotator
2. After saving, the new annotated file has to be added to the moodle database

6.1 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 files of essay-type questions.

6.2 File API

For managing all the files in moodle, moodle provides File API. 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.

Files in moodle are conceptually stored in File areas that are used to restrict modules from

```
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. 6.1 mdl_file table

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 plugin.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

```php
$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. 6.2 File record object

order to get the file we need to use the get_file function

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

Fig. 6.3 Get File function

6.2.1 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

It 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

It 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

It 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.

6.3 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. `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, 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. 6.4 Function to save a local copy of the input PDF

6.4 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 `myindex.html`, `myscript.js` and `mypdfannotate.js`. The functions called by the interface from `myindex.html` are contained in `myscript.js` and `mypdfannotate.js`.

6.5 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 `mypdfannotate.js` file. For this purpose, we can use XMLHttpRequest. We have added the required folders of FPDF and FPDI inside the

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

Fig. 6.5 Sending annotation data to php

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

```
//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. 6.6 Getting annotation data from javascript and creating a file object

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.

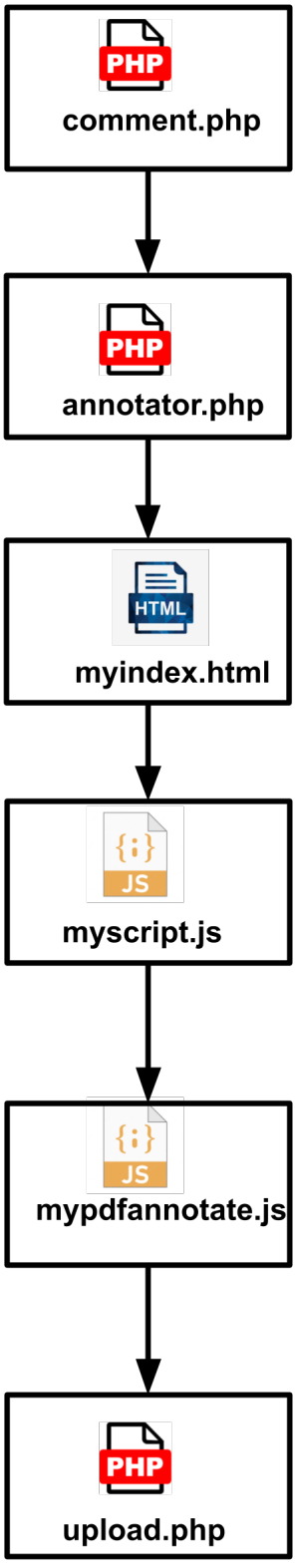


Fig. 6.7 Flow chart of files used for annotation

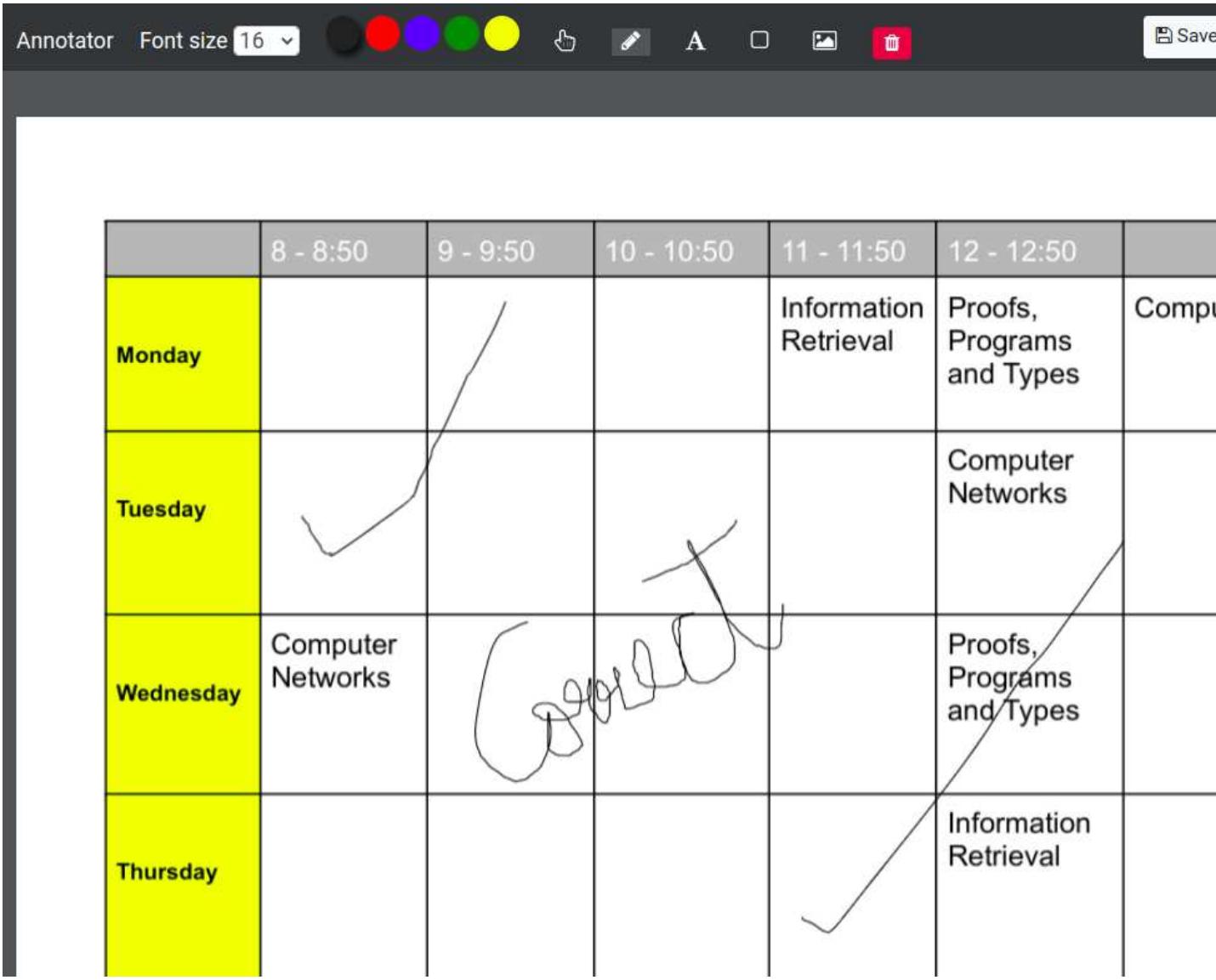
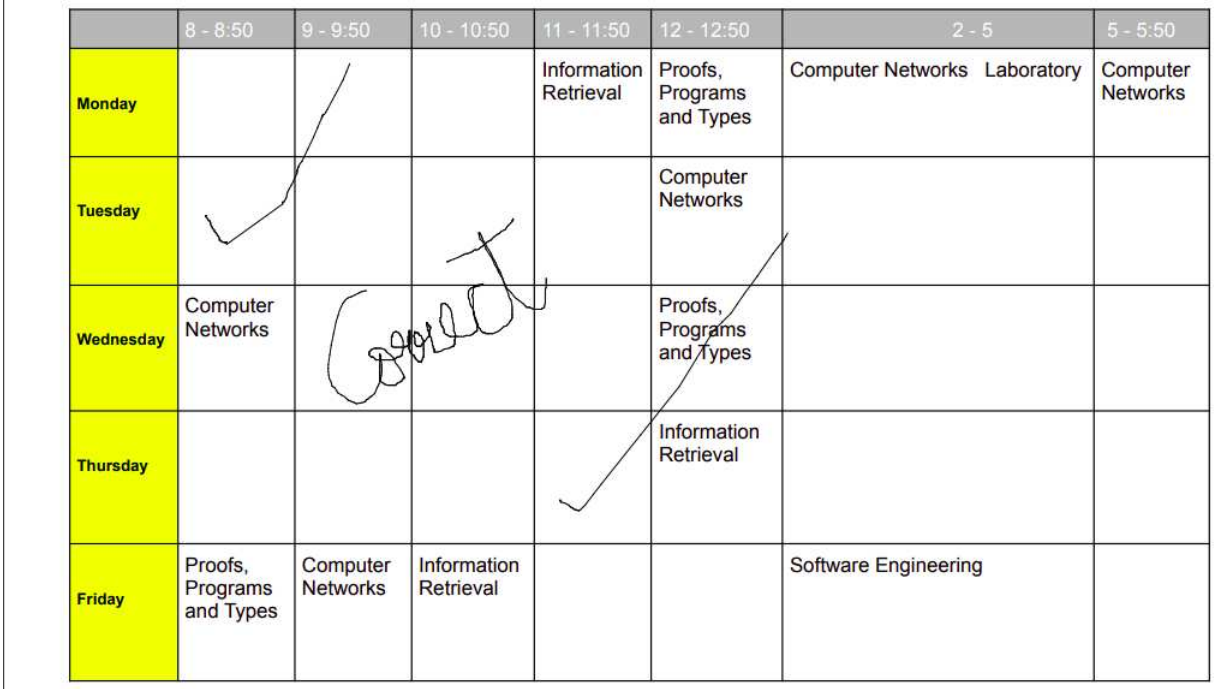


Fig. 6.8 User Interface for annotation



	8 - 8:50	9 - 9:50	10 - 10:50	11 - 11:50	12 - 12:50	2 - 5	5 - 5:50
Monday				Information Retrieval	Proofs, Programs and Types	Computer Networks Laboratory	Computer Networks
Tuesday					Computer Networks		
Wednesday	Computer Networks				Proofs, Programs and Types		
Thursday					Information Retrieval		
Friday	Proofs, Programs and Types	Computer Networks	Information Retrieval			Software Engineering	

Fig. 6.9 Final PDF after annotation

Fig. 6.10 Size of the input and output PDFs

Chapter 7

Conclusion and Future Work

7.1 Conclusion

After installing and setting up moodle in our local system, we learned and understood the different features of moodle including creating courses and users, adding assignments, quizzes, and activities inside courses, different roles present in moodle, adding new roles, and granting and revoking permissions from different roles. We studied the database structure of moodle and learned about some of the important tables present in moodle. Further, we learned how to develop a plugin in moodle and created two simple useful plugins. We looked for various PDF annotation libraries that can annotate PDF documents and save them without considerably increasing the size of the resultant PDF. We came across Poppler and Cairo libraries and explored the functionality of the libraries and the methods by which annotation works in these libraries. Since the integration of these libraries is not complete in PHP, we looked for alternate options and came across FPDF/FPDI. We looked through the functions present in these libraries and found these libraries apt for our project. We also tried annotating PDF documents using FPDF/FPDI.

We finalized FPDF/FPDI to be the annotation library. For the user interface of the plugin, we are reusing the UI made by Vishal Rao and Tausif Iqbal. We integrated the PDF annotator with moodle. Now the annotated PDF has a size in a range similar to that of

the input PDF, with the properties of PDF being conserved and hence achieving one of our prime objectives.

7.2 Future Work

- Adding a highlighter to the tools in the annotator
- Make PDF annotation as a plugin that can be used for all types of PDF annotations like assignments and quizzes
- Try to expand the annotation module for different types of files.

References

[1] File APIs of moodle : https://docs.moodle.org/dev/File_API

[2] File API internals of moodle: https://docs.moodle.org/dev/File_API_internals

[3] Moodle quiz database : https://docs.moodle.org/dev/Quiz_database_structure

[4] FPDF official website : http://www.fpdf.org/

[5] FPDI official website : https://www.setasign.com/products/fpdi/about/

[6] XMLHttpRequest : https://stackoverflow.com/questions/43098620/sending-data-from-javascript-to-php-via-xmlhttprequest

[7] Sending POST request : https://stackoverflow.com/questions/9713058/send-post-data-using-xmlhttprequest

[8] FabricJS functions : http://fabricjs.com/fabric-intro-part-3

[9] FabricJS canvas to PDF : https://stackoverflow.com/questions/24081191/fabric-js-canvas-into-pdftcpdf

[10] Transparency in FPDF : http://www.fpdf.org/en/script/script74.php

Appendix A

Code

Github Link