BANGALORE UNIVERSITY

Jnana Bharathi Campus, Bangalore-560056

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATIONS

A Project Report On

"SMART QUESTION PAPER GENERATOR"

Submitted by

KIRAN J (P03NK23S126039)

Under the guidance of

DR. M HANUMANTHAPPA

Senior Professor and Coordinator

Department of Computer Science and Applications

Towards

MINI PROJECT

Prescribed by the BANGALORE UNIVERSITY

MASTER OF COMPUTER APPLICATIONS

For the academic year **2024-2025**

BANGALORE UNIVERSITY

Jnana Bharathi Campus, Bangalore-560056

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATIONS

A Project Report On

"SMART QUESTION PAPER GENERATOR"

CERTIFICATE

This is to certify that **KIRAN J (P03NK23S126039)** has satisfactorily completed the project titled **"Smart question paper generator"** towards Mini project Lab prescribed by the Bangalore University for the 3rd Semester Master of Computer Application course in academic year 2024-2025.

DR. M HANUMANTHAPPA

Guide, Senior Professor and coordinator

Examiners:

1)

2)

DECLARATION

I hereby declare that the project report, titled **"SMART QUESTION PAPER GENERATOR"**, is prepared in partial fulfilment of the requirements for the award of the degree of Master of Computer Applications of Bangalore University for the academic year 2024-2025. Further, the matter embodied in dissertation has not been submitted previously by anybody for the award of any Degree or Diploma to any other University.

KIRAN J (P03NK23S126039)

DATE: 18/06/2025

PLACE: Bangalore

ACKNOWLEDGEMENT

The satisfaction that I feel at the successful completion of the lab work title **"SMART QUESTION PAPER GENERATOR"** would be incomplete if I do not mention the names of the people whose valuable guidance and management has made this project work a success.

It is a great pleasure to express my gratitude and respect to all those who Inspired and helped me in completing this project.

It is pleasure to thank my internal guide **Dr. M HANUMANTHAPPA** and Coordinator of the department of computer science & application, Bangalore University, for his valuable assistance and cooperation who has given me ideas and guidance to make the report a highly useful & knowledge base experience.

It is also pleasure to express my gratitude to all Teaching and Non-teaching staff members of department of Computer Science and Application for their encouragement and providing valuable requirement. With a deep sense of indebtedness, I convey my heartiest thanks to my parents who have taken effort and given me such an opportunity. Their love and blessing has given me strength to acquire knowledge and gain experience in my life. As well as my friends who have helped for this accomplished task.

ABSTARCT

Smart question paper generator using Django is a web-based application designed to streamline the creation of academic question papers. Built using the Django web framework, this system allows admins to securely log in, manage question banks, and automatically generate question papers based on predefined criteria such as subject, marks, and question types. The platform emphasizes ease of use, scalability, and security, enabling institutions to save time and reduce manual errors in examination planning. Core functionalities include admin authentication, question management, and dynamic paper generation. This system enhances educational efficiency through automation and centralized data handling.

The **Smart question paper generator** using Django is a web-based automated system aimed at generating academic examination papers dynamically. Developed using the Django framework (Python), this application facilitates secure admin authentication, structured question bank management. The system architecture follows the Model-View-Template (MVT) pattern and utilizes an MySQL database for backend storage. Educators can input and categorize questions based on subject, topic, marks, and unit wise question Upon selection of criteria, the system uses a randomized algorithm to generate balanced and non-repetitive question papers. The modular design ensures scalability, while the integration of Django's admin interface streamlines content management. This tool aims to eliminate the manual effort and bias in paper setting, offering an efficient and reproducible method for academic assessment design.

CONTENTS

1. INTRODUCTION

Smart question paper generator Using Django the Educational institutions are increasingly using digital tools to improve efficiency. One key area is the preparation of exam question papers. The **Smart question paper generator**, built with the Django web framework, automates and simplifies this task.

The **Smart question paper generator** is a web-based application designed to simplify and improve the process of creating exam papers in educational institutions. Traditionally, preparing question papers manually is a time-consuming task that often leads to issues such as repeated questions, inconsistent difficulty levels, and formatting errors. This system, built using Django—a powerful Python framework—offers a centralized platform where all questions are stored and can be easily filtered by subject, topic, marks, and question type. By automating the paper-setting process, it significantly reduces time and errors, ensuring a more organized and fair approach to exam preparation. The application supports role-based admin access, allowing only authorized users to manage questions and generate papers securely. With its clean, user-friendly interface, the tool encourages collaboration among faculty members and enhances the overall quality of assessments. It is flexible, scalable, and well-suited for real-world educational settings.

1.1 Overview

This web-based system allows admin to generate question papers quickly and fairly. It reduces manual effort, avoids repeated or unbalanced questions, and ensures a smooth, error-free process. A centralized database helps with collaboration and consistency across departments.

1.2 Why It's Needed

Manually creating question papers is a time-consuming process that often results in several issues, such as the repetition of questions, imbalanced difficulty levels, and human errors in formatting. This not only affects the quality of the assessments but also increases the workload for administrators and faculty members involved in the exam preparation process.

1.3 Project Goals

The system offers secure access through role-based login, ensuring that only authorized administrators can manage and generate question papers. It features a well-organized question bank that allows users to easily add, edit, and manage questions. Additionally, the application supports automatic question paper generation using filters such as subject, marks, and question type, making the process faster and more efficient.

1.4 Benefits

The system saves significant time and effort by automating the question paper creation process. It helps reduce mistakes and eliminate duplicate questions, ensuring higher accuracy. By allowing controlled filtering and selection, it supports the creation of fair and balanced exam papers. Additionally, the application is easy to use and scalable, making it suitable for institutions of various sizes and adaptable to growing needs.

2. OBJECTIVES

The **Smart question paper generator** is designed with several key features to streamline and enhance the exam preparation process. At its core is a comprehensive question bank that stores all questions in one centralized location, allowing administrators to add, edit, and organize them by subject, topic, marks, and type. Secure login and role-based access are implemented using Django's authentication system, ensuring that only authorized admins can access and manage system features. The application also supports automatic question paper generation based on selected filters like subject and marks, helping to avoid repetition and maintain consistency. A clean and simple user interface provides intuitive navigation for logging in, managing questions, and creating papers. Django's built-in admin tools are leveraged to simplify question management, and the system's structure allows for easy updates and future enhancements, such as PDF export or additional filtering options. Designed to be scalable, the system can handle a large volume of questions and is suitable for real-world use in colleges. Ultimately, it aims to reduce errors and ensure that the generated question papers are fair and balanced in terms of difficulty.

The Smart question paper generator offers a complete solution for efficient and error-free exam paper creation. Along with core features like a secure login, question bank, and auto-generation based on filters, it includes advanced options such as difficulty tagging, paper previews, randomization, and export to PDF. It also supports question history tracking, paper templates, and multi-language capability. With backup, analytics, and email notifications, the system is easy to manage, scalable for institutions, and designed for real-world educational use.

3. PROJECT DESCRIPTION

A web-based application built with Django to help admins create exam question papers quickly and accurately. It uses a centralized question bank and allows paper generation based on subject, marks, chapters, and question type.

3.1 Core Features

The system allows admins to securely log in using Django's authentication system, ensuring only authorized access. Once logged in, admins can add, manage, and organize questions, as well as generate question papers efficiently through the platform.

3.2 Question Bank

Admins have the ability to add, edit, and delete questions, which are categorized by subject, type, marks, and difficulty level for better organization. All question data is stored securely in a MySQL database and managed efficiently using Django's Object-Relational Mapping (ORM) system.

3.3 Paper Generator

The system can generate question papers based on various filters such as subject, question count, question type (for example, 5 short questions and 2 long questions), and marks. It ensures that the selection of questions is randomized and non-repetitive, creating unique and balanced papers each time.

3.4 Frontend Interface

The application uses HTML and CSS combined with Django templates to create a clean and user-friendly interface. It includes forms that allow admins to add questions and set criteria for generating question papers. Additionally, there is an option planned for future upgrades to enable downloading the generated papers directly from the system.

3.5 Workflow

Admins can securely log in to the system and add questions along with details such as marks and main topics. They can then generate question papers using various filters to customize the content. Once generated, the papers can be reviewed, and there is an option to download the final output for convenience.

3.6 Example Use Case:

A faculty logs in, adds categorized questions for “Data Structures,” and generates a paper with easy short and medium long questions. The system fetches and arranges questions for print or download.

4. SYSTEM ANALYSIS

A web application built with Django to automate exam question paper creation. It offers a centralized question bank and allows generation of papers based on subject, marks, chapter, and question type. It replaces manual, error-prone processes with an efficient, role-based system.

4.1 Problems in Existing System

The traditional process of creating question papers is time-consuming and manual, often lacking a centralized storage system for questions. This can lead to repeated or imbalanced questions, formatting inconsistencies, and a high risk of human error, all of which affect the quality and fairness of the exams.

4.2 Proposed Solution

The system features secure login for admins to ensure authorized access, along with a comprehensive question bank that categorizes questions by subject, type, and marks. It supports random and balanced paper generation based on various filters, helping to create fair and varied exams. An admin dashboard allows efficient management of both administrators and questions, while a simple web interface with dynamic content rendering provides a smooth and user-friendly experience.

4.3 System Requirement - Functional:

The system provides secure admin authentication to ensure that only authorized users can access its features. Admins have the ability to add, edit, and delete questions, which can be filtered by subject, type, and

marks for easy organization. It also supports generating question papers based on these filters, while implementing access controls to manage admin permissions effectively.

4.4 Non-Functional:

The system is designed with security in mind, incorporating features such as CSRF protection, secure session handling, and password hashing to safeguard user data. It offers a user-friendly interface built with Django templates, making it easy to navigate and use. The application is scalable to accommodate more users and subjects as needed, and it is optimized for speed and efficiency through the use of Django's ORM for database operations.

4.5 Feasibility Summary

Technically, the system is built using open-source technologies like Django and Python, making it deployable on standard servers without requiring specialized hardware. Operationally, it is easy to use and helps reduce the workload of faculty by automating the question paper creation process. Economically, it is cost-effective since it does not involve any licensing fees, making it an affordable solution for educational institutions.

4.6 Benefits

The system offers fully automated paper generation by utilizing a centralized, reusable question database. It ensures that each paper is random, unique, and balanced in terms of question selection. Role-based secure access controls protect the system, allowing only authorized users to manage and generate papers. Additionally, it supports exporting the

generated papers in current formats like PDF and Word for easy distribution and printing.

4.7 Limitations:

The system currently features a basic user interface that can be improved for better usability, and it may require a database upgrade to handle larger deployments effectively.

5. SYSTEM DESIGN

Architectural Design – **Smart question paper generator (Django)**

5.1 MVT Architecture

The system is built on Django's Model-View-Template (MVT) architecture, where the Model defines the database structure, including entities like Questions, Subjects, and Admins. The View handles the application logic, processes user input, and manages response rendering. The Template consists of HTML files embedded with Django tags to display dynamic content to users.

5.2 Core Modules

The system features secure admin authentication using Django's built-in auth system with session handling to manage user sessions. It includes a question bank where admins can add, edit, and delete questions, specifying details such as subject, type, and marks. The paper generator accepts filters like subject and marks to fetch random questions for creating balanced papers. An admin panel powered by Django's admin interface allows easy management of admins and question content. The user interface is built with HTML templates that provide forms, dashboards, and paper displays for a smooth and intuitive user experience.

5.3 Database Design (MySQL via Django ORM)

The system includes several key database tables: the Admin table stores admin credentials and roles to manage access, the Subject table holds subject names and their corresponding IDs, and the Question table contains question text along with associated details such as subject, type,

and marks. Additionally, there is an optional Generated Paper table that logs information about created papers, including timestamps and the questions used, to maintain a record of past exams.

5.4 Data Flow Diagram (DFD)

- Level 0 (Context): Admin interacts to login, manage questions, generate papers
- Level 1 (Processes): Login → Question Management → Paper Generation → Admin Actions

5.5 User Interface Design

The system includes a login page that authenticates admins securely, providing access to the question paper dashboard where admins can add and view questions as well as generate papers. There is also an admin dashboard for managing both admin users and question content. The paper generator allows selection of subject, marks, and question count to customize the exam paper. Finally, the result page displays the generated paper in a printable format for easy use.

5.6 Scalability & Extensibility

The system features a modular design that allows for easy upgrades, such as adding PDF export, analytics, or integration with learning management systems (LMS). It supports large question banks and multiple subjects, making it suitable for diverse educational needs. Additionally, it is easily scalable and can be adapted to use databases like PostgreSQL or MySQL for larger deployments.

6. PROJECT REQUIREMENTS

Components and Technology Used

The project is developed using Django, a high-level Python web framework that facilitates rapid development and clean, maintainable code. MySQL is used as the backend database to store and manage structured data efficiently. The frontend is built using HTML and CSS for layout and design, JavaScript for adding interactivity, and Django Templates for rendering dynamic content. User authentication and authorization are handled using Django's built-in authentication system, ensuring secure login and user session management. Additionally, the Django Admin Panel is utilized as the admin interface, providing a powerful and user-friendly way to manage application data and users without requiring additional coding.

7. FEATURES

User Login & Roles

Access to the system requires users to log in, ensuring that only authorized individuals can use its features.

Admin Role:

The system allows for comprehensive management of admins and all questions, enabling efficient organization and control. It also supports the generation of question papers based on the stored question bank and specified criteria.

Admin Features

The system provides robust management of the question bank, allowing admins to add, edit, and delete questions. Each question can be categorized by subject, type (such as short or long), and marks, ensuring organized and detailed question management.

Generate Question Papers

The system allows users to choose the subject, number of questions, type, and marks when generating question papers. It ensures a random and balanced selection of questions without any repeats, creating fair and varied exams.

System Features

The system features a simple interface built with HTML and Django templates, providing easy navigation for admins. It also utilizes Django's built-in admin panel to enable efficient control and management of data.

Security

Passwords are stored securely using hashing techniques, and the system includes protection against CSRF and other common web threats. Access to the application is limited strictly to authenticated admins to ensure security.

Data & Database

All data, including admin information and questions, is stored securely in a database. The system uses SQL by default and is easily upgradeable to more advanced database systems as needed.

Ready for Future

The system has a clean and organized codebase, making it easy to maintain and extend with new features such as PDF export.

8. IMPLEMENTATION

Project Setup

The system is built using Python and Django. Django was installed using the command pip install django. The project and app were created with the commands django-admin start project question generator and python manage.py startapp core, respectively.

Database

The system uses MySQL as the default Django database. It includes models for Questions, Subjects, and Users, leveraging Django's built-in authentication system. To set up the database schema, migrations are created and applied using the commands python manage.py makemigrations and python manage.py migrate.

User Authentication

The system utilizes Django's built-in login and logout functionality to manage user authentication. Only registered users can access the system's features, with specific roles assigned, such as Admin, to control permissions and access levels.

Question Bank Module

Faculty members can add questions through user-friendly forms that include fields for Subject, Type, Chapters, and Marks. The data entered is saved securely using Django models, ensuring efficient storage and management.

Paper Generation Module

Users can select criteria such as Subject, number of questions, Chapter, Year, Marks, and Type to customize their question paper. The system then randomly selects questions that match these filters, and the generated paper is displayed for review and printing.

Django Admin Panel

The system allows management of Questions, Subjects, and Users through the built-in Django admin interface, accessible at the /admin / URL for easy and centralized control.

Frontend & Templates

The system uses Django Template Language combined with HTML to render dynamic content, providing a simple user interface. Optional styling can be added using Bootstrap or CSS to enhance the visual appearance.

Running the App

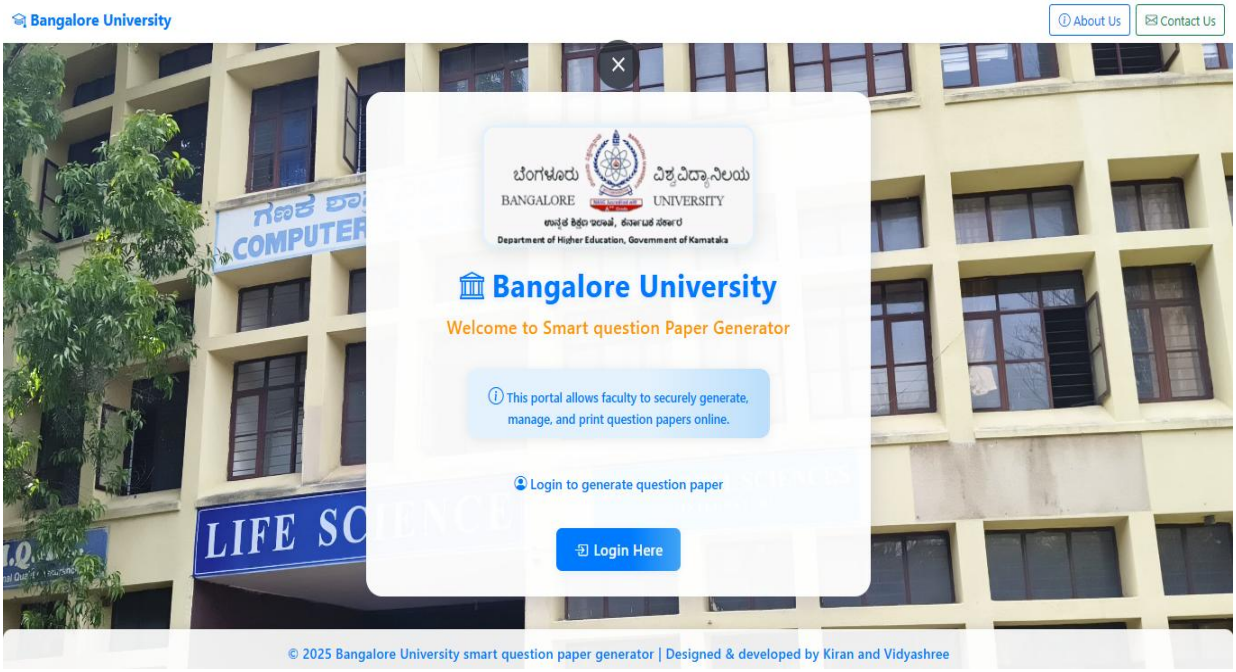
To start the development server, run the command python manage.py runserver. Once running, the application can be accessed locally at http://127.0.0.1:8000/.

Folder Structure

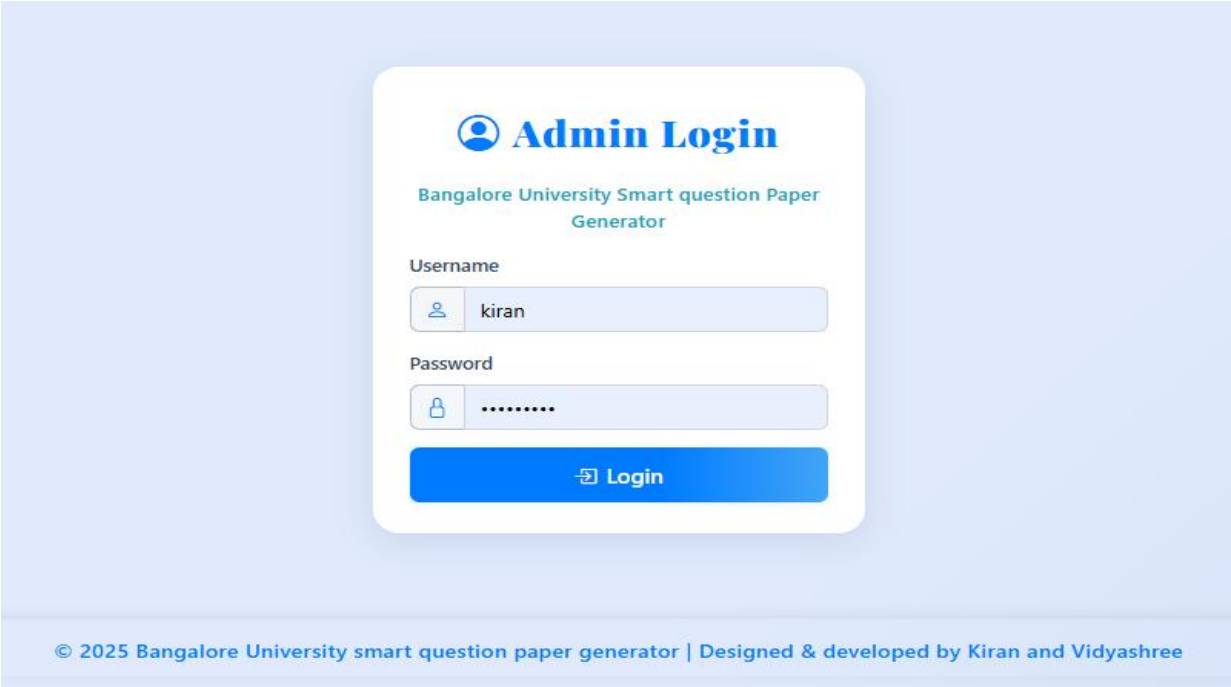
The project structure includes the question_generator/ directory, which contains the project settings, and the core/ app directory that holds the models, views, and URLs. HTML files are stored in the templates/ folder, and the database is maintained in the db.sqlite3 file.

9. SNAPSHOTS

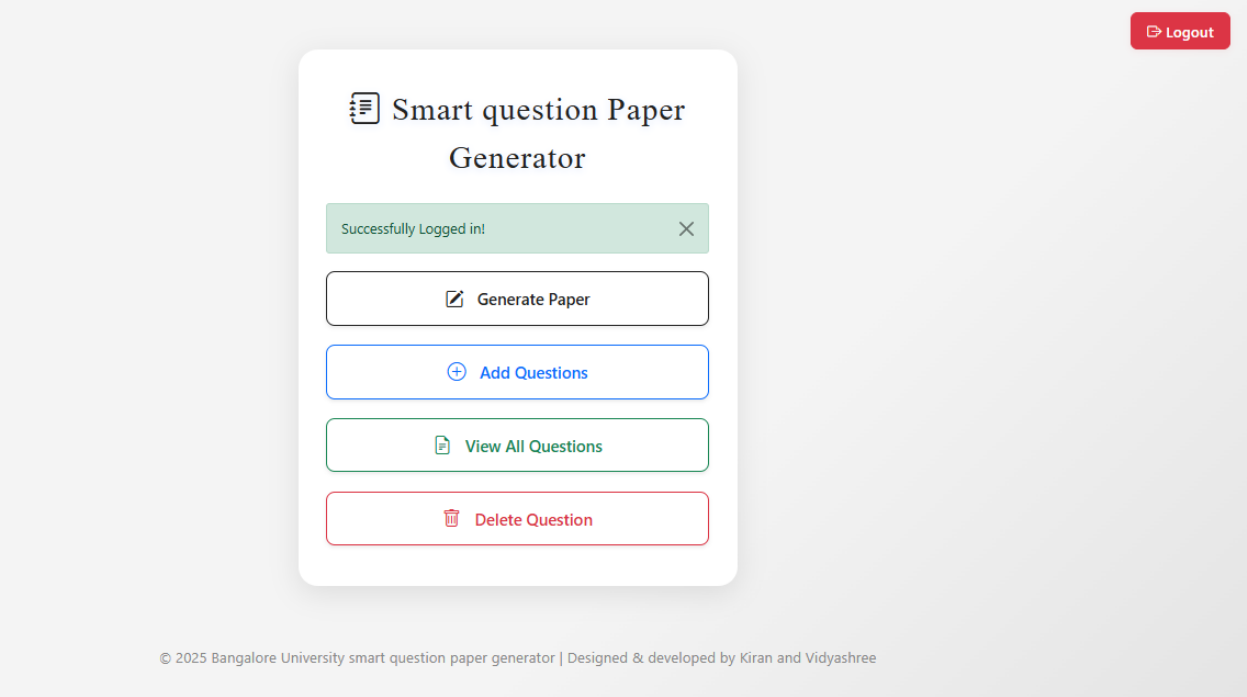
Home page (About and Contact us)



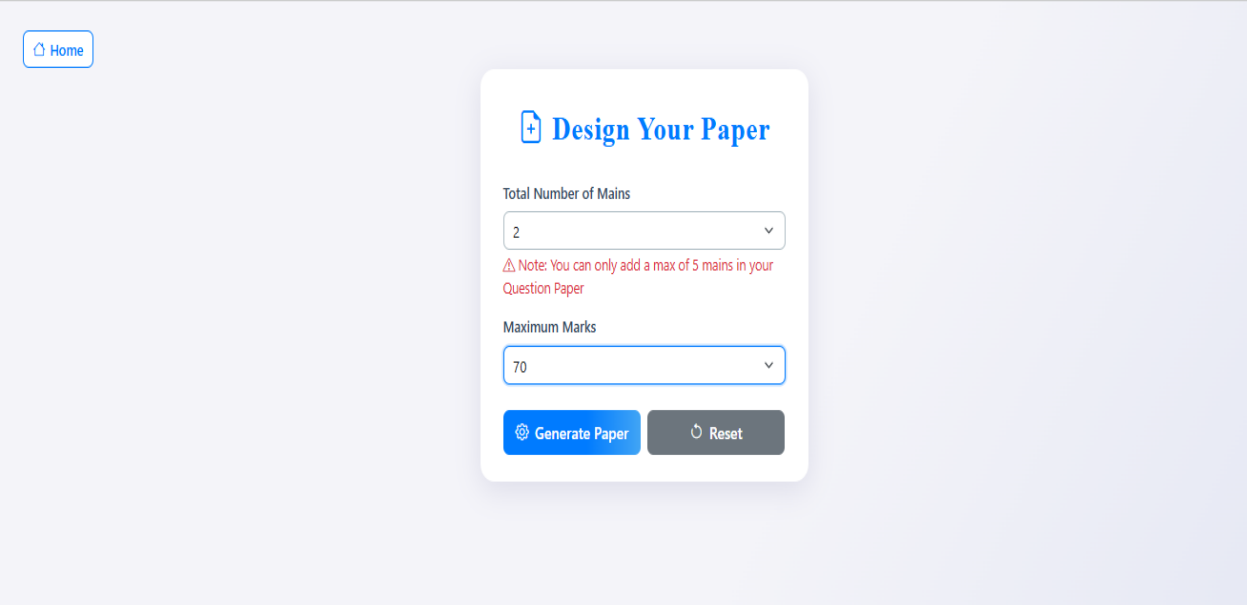
Admin login page



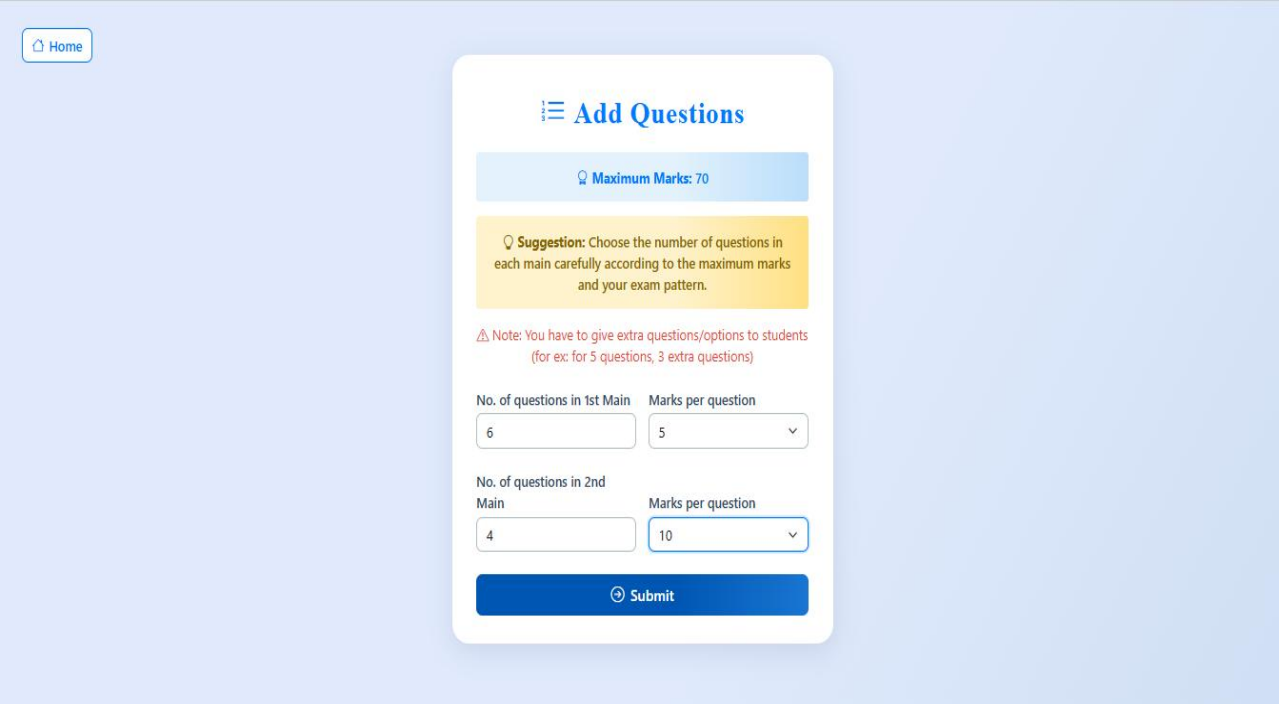
Index page



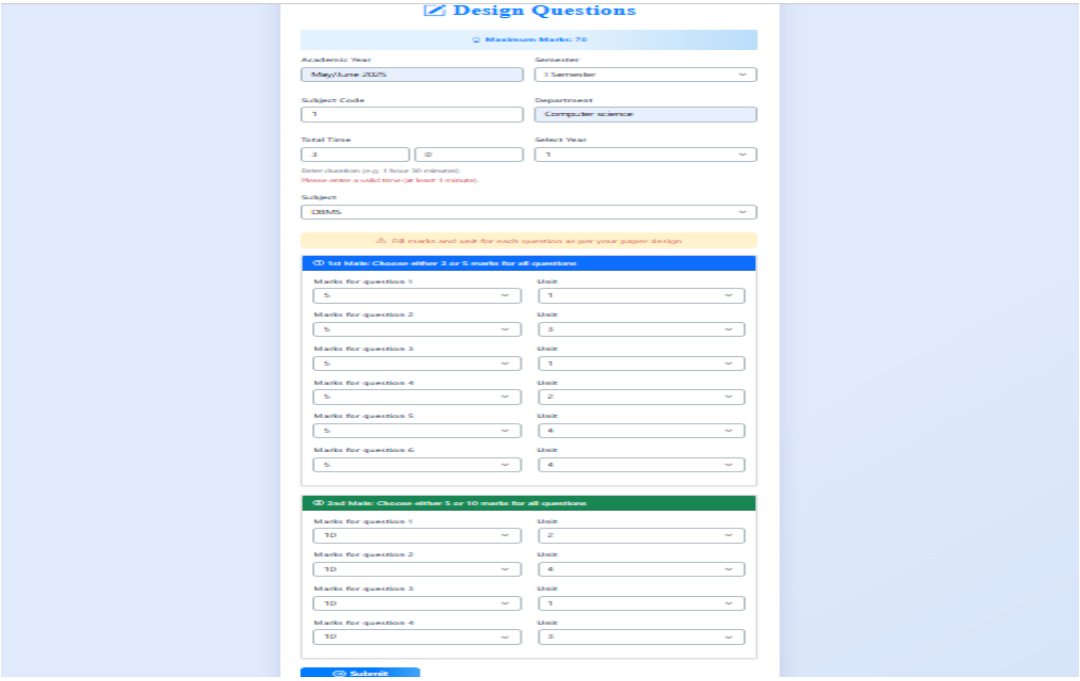
Generate paper (Design paper)



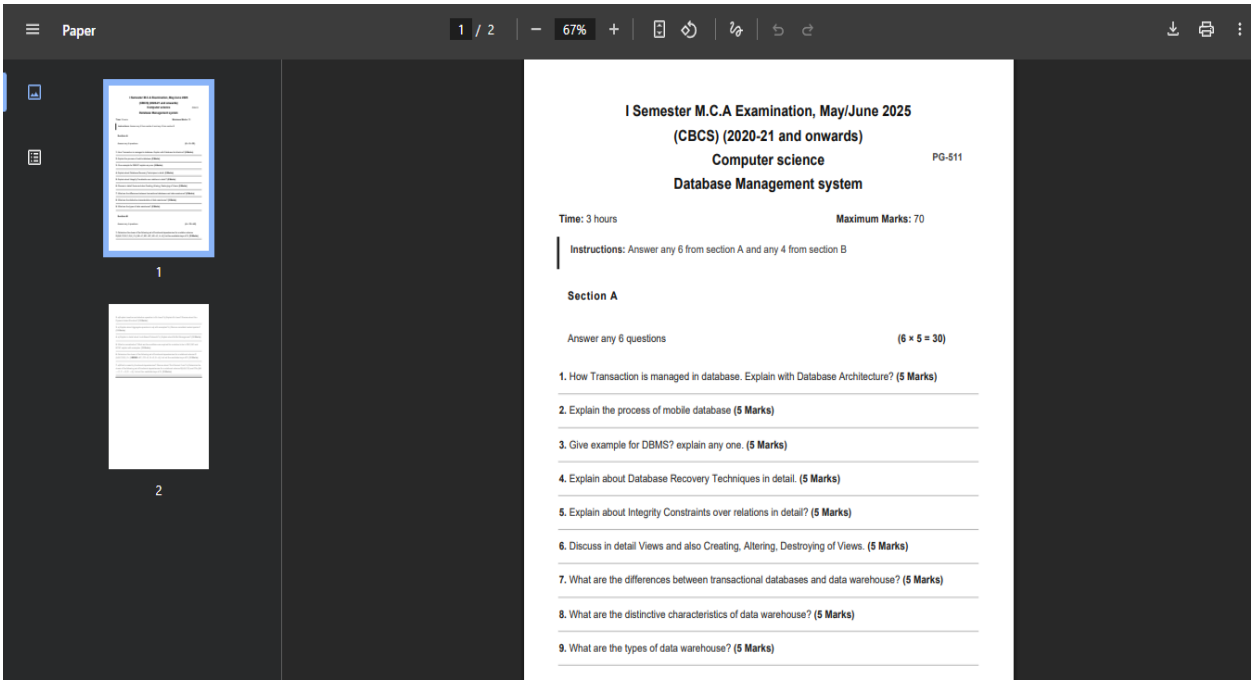
Generate paper (Add marks and number of questions in each main)



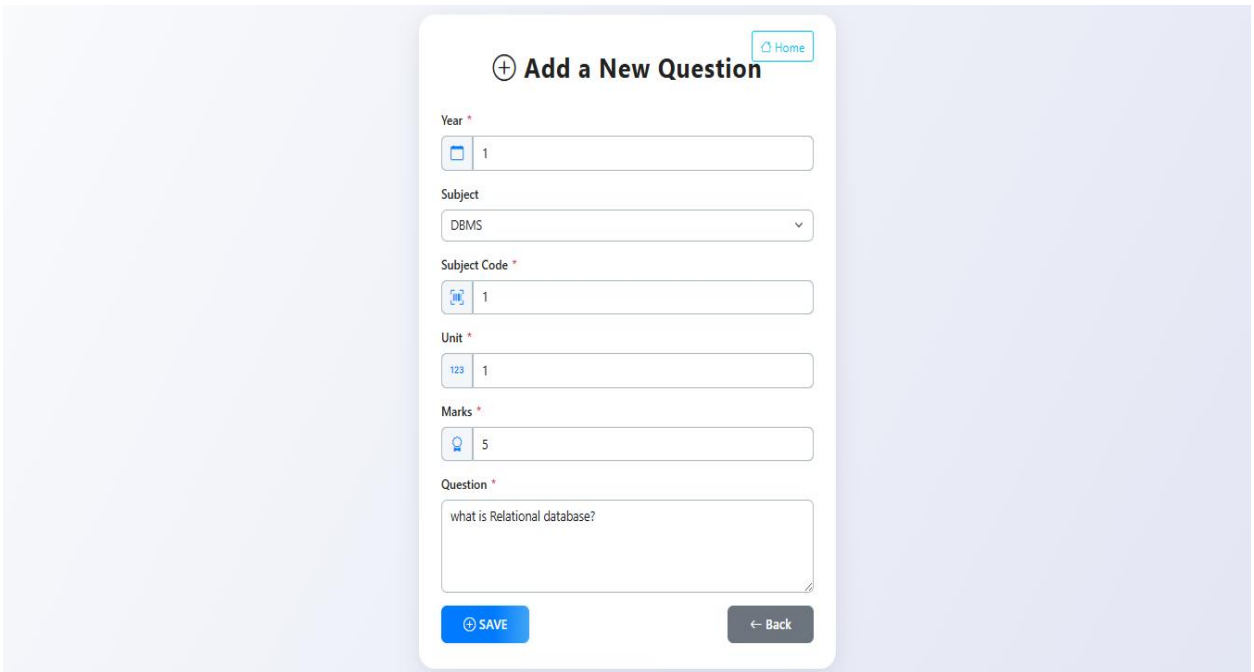
Generate paper (Design questions)



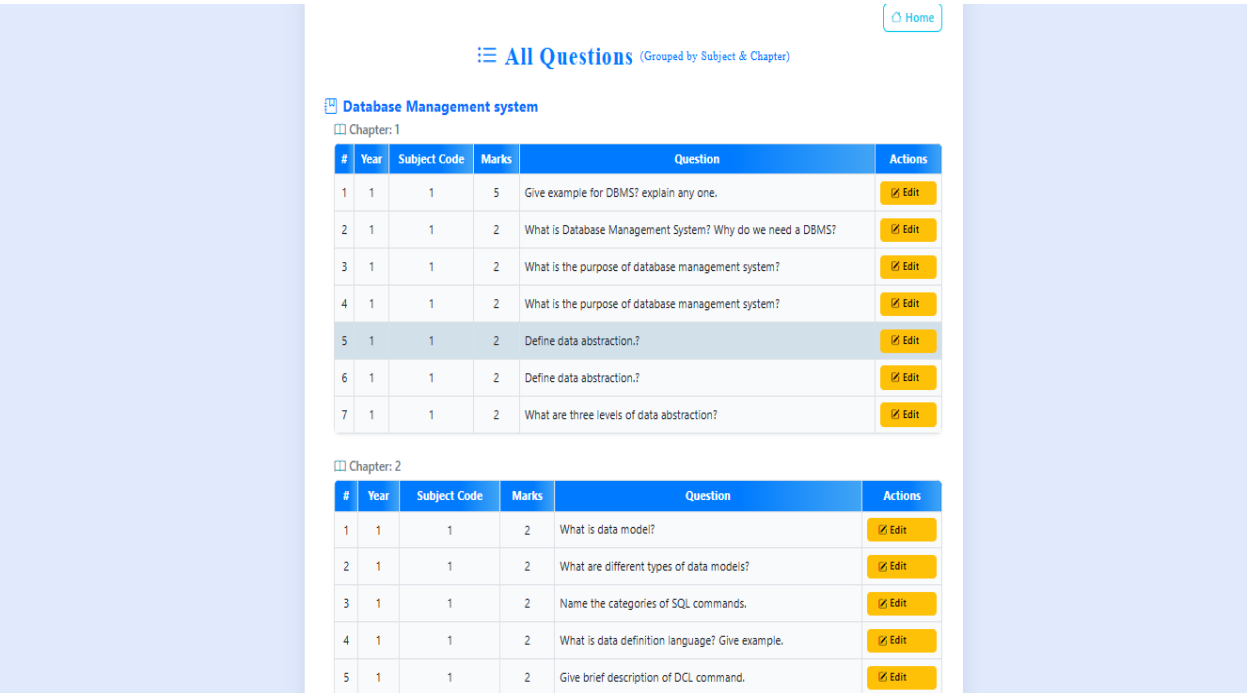
Generate paper (Question paper generated PDF)



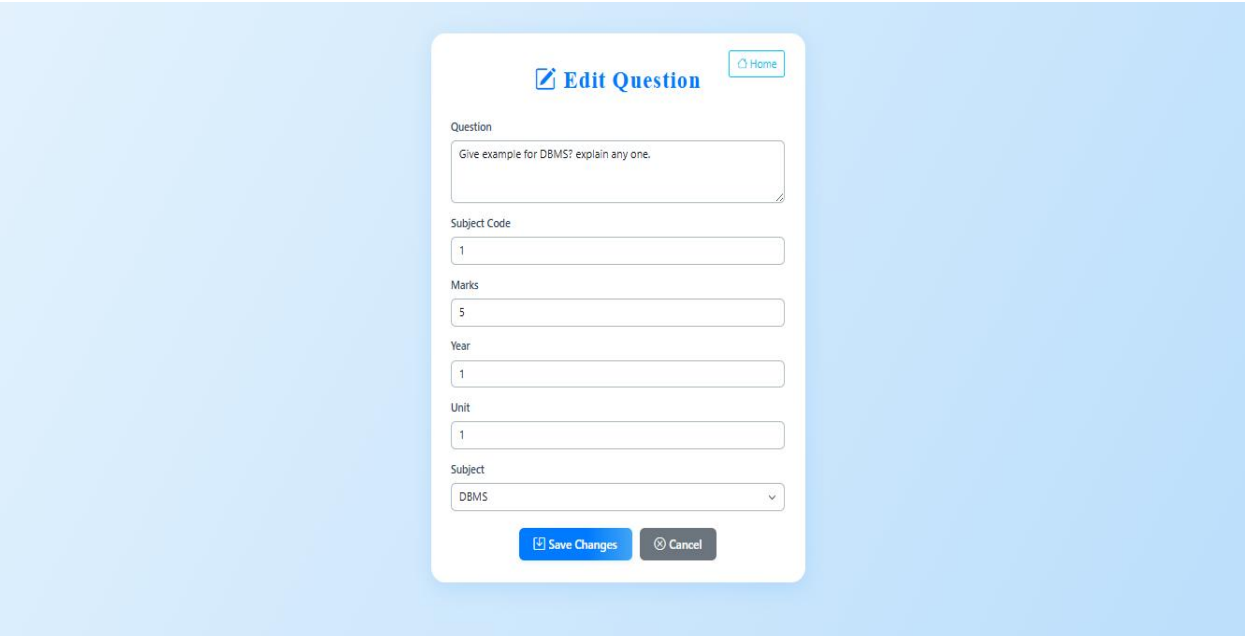
Index page (Add questions)



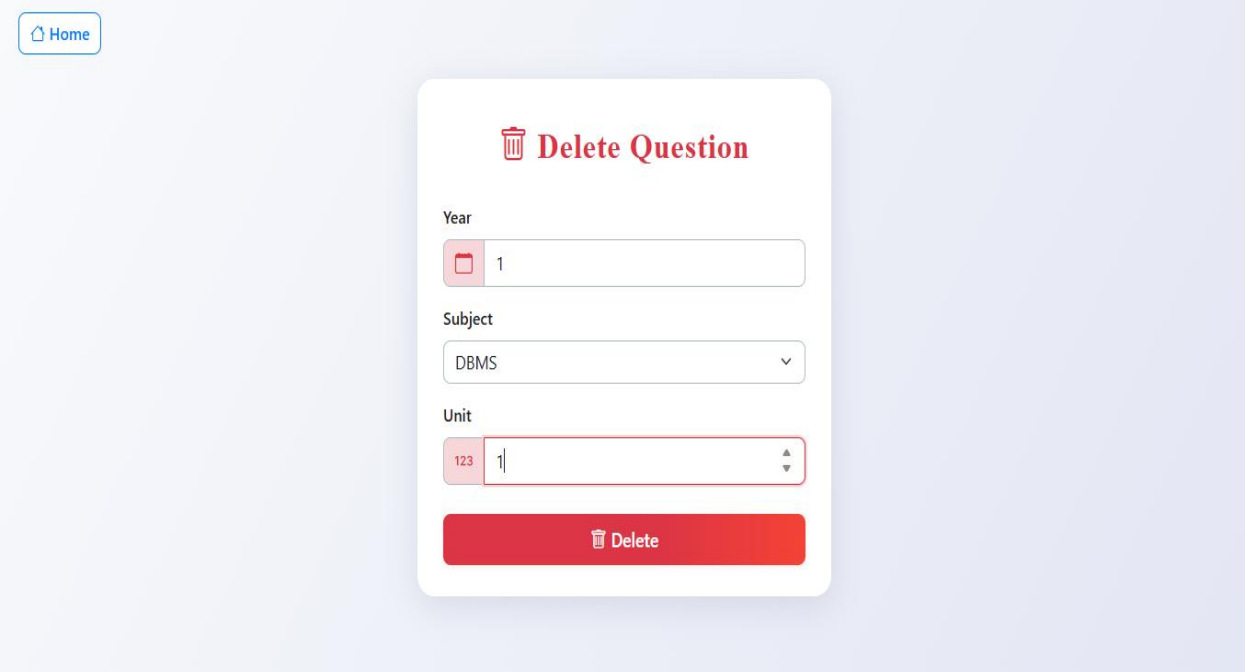
Add questions (View Questions)



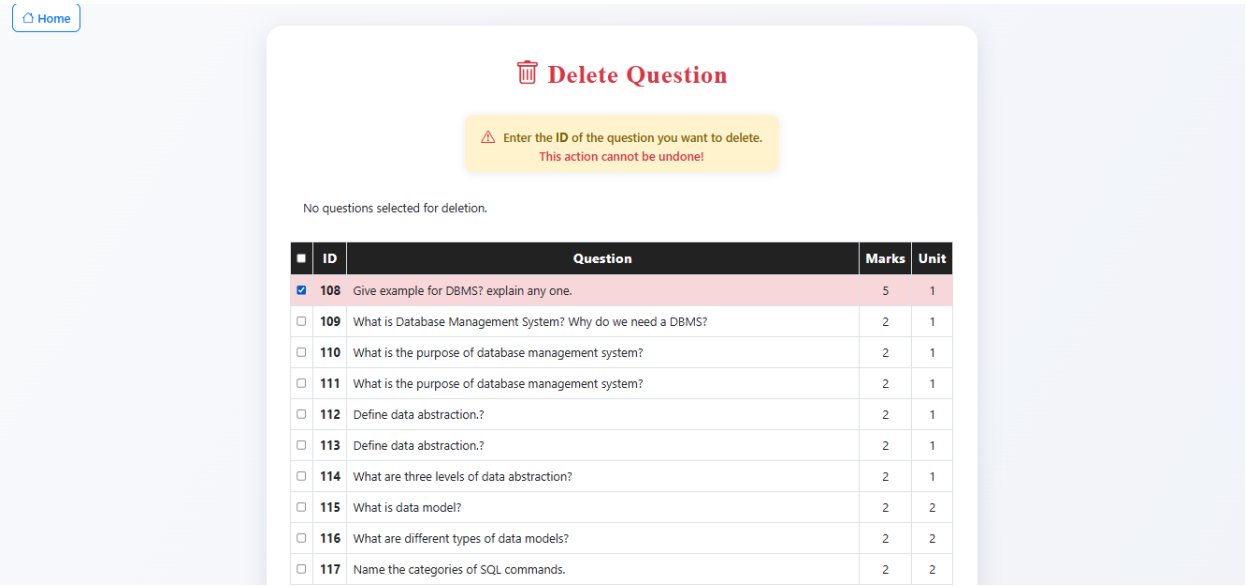
Add questions (Edit Questions)



Index page (Delete questions)

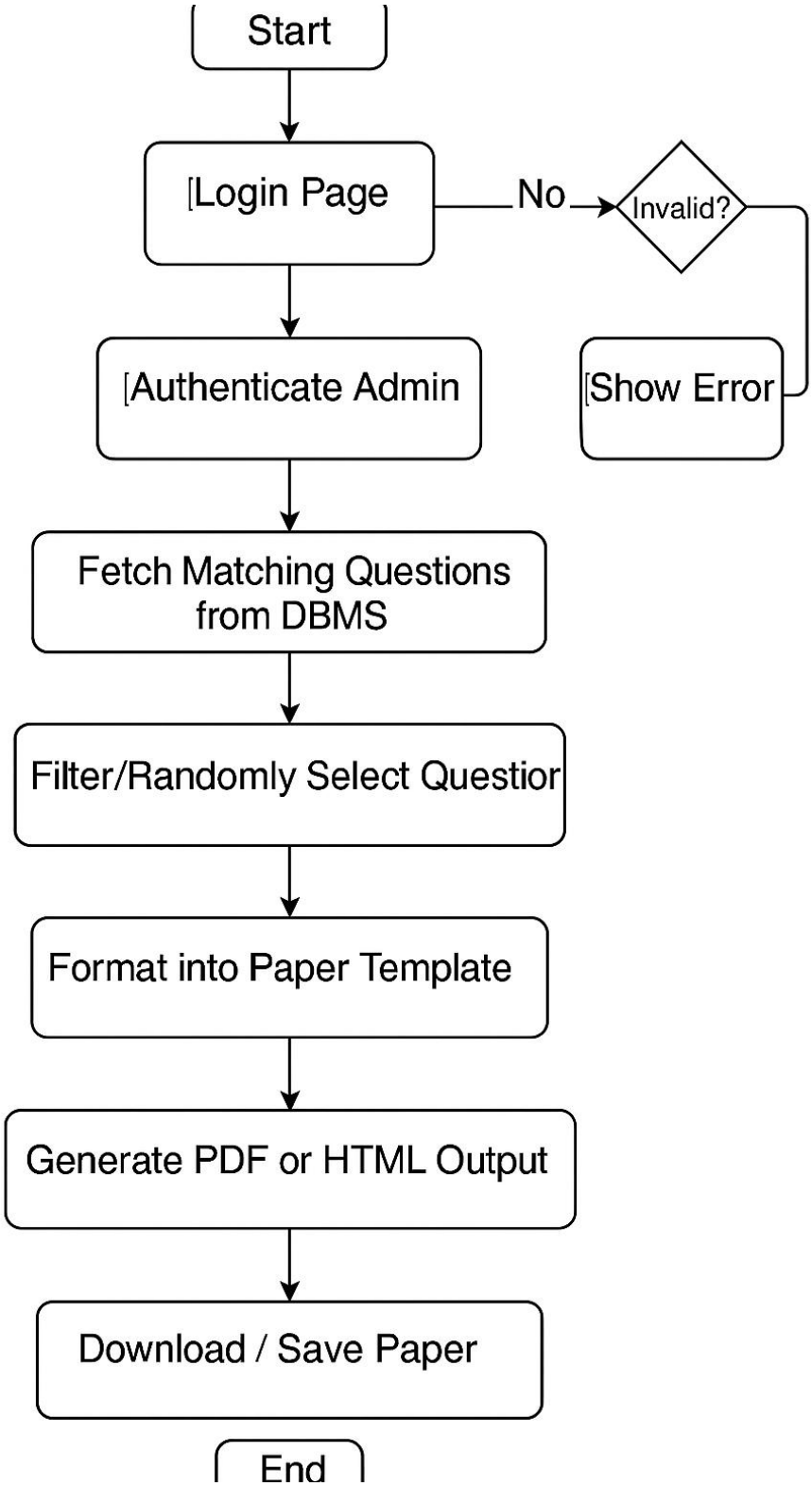


Delete questions (Questions to delete)



	ID	Question	Marks	Unit
☑	108	Give example for DBMS? explain any one.	5	1
☐	109	What is Database Management System? Why do we need a DBMS?	2	1
☐	110	What is the purpose of database management system?	2	1
☐	111	What is the purpose of database management system?	2	1
☐	112	Define data abstraction.?	2	1
☐	113	Define data abstraction.?	2	1
☐	114	What are three levels of data abstraction?	2	1
☐	115	What is data model?	2	2
☐	116	What are different types of data models?	2	2
☐	117	Name the categories of SQL commands.	2	2

10. FLOWCHART (FLOW OF PROJECT)



11. CODE SNIPPETS

mysite

surls.py

```
from django.contrib import admin
from django.urls import path, include
from django.conf.urls import url
from mysite import views
from .views import GeneratePdf
from django.views.static import serve
from django.conf import settings
urlpatterns = [
    path('admin', admin.site.urls),
    path('',views.home, name='home'),
    path('generatePaper', views.generatePaper,name='generatePaper'),
    path('loginapp/', include('loginapp.urls')),
    path('delete',views.delete, name='delete'),
    path('deleteQuestion',views.deleteQuestion, name='deleteQuestion'),
    path('deleteSuccess',views.deleteSuccess, name='deleteSuccess'),
    path('pdf/paper.html', GeneratePdf.as_view()),
    path('intermediate.html',views.intermediate, name='intermediate'),
    path('intermediate2.html',views.intermediate2, name='intermediate2'),
    path('loginpage', views.loginpage, name="lpage"),
    path('login',views.view_login, name='login'),
    path('logout',views.logout_user, name='logout'),
```

```
    path('index', views.index, name='index'),
    path('add_question/', views.add_question, name='add_question'),
    path('view-questions/', views.view_questions, name='view_questions'),
    path('edit-question/<int:id>/', views.edit_question, name='edit_question'),
    # url(r'^media/(?P<path>.*)$', serve,{'document_root':       settings.MEDIA_ROOT}),
    # url(r'^static/(?P<path>.*)$', serve,{'document_root': settings.STATIC_ROOT}),
]
```

views.py

```
from django.http import HttpResponse
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth import authenticate, login, logout
from loginapp.models import questionBank
from django.contrib.auth.decorators import login_required
from django.views.generic import View
from mysite.utils import render_to_pdf
from django.template.loader import get_template
from io import BytesIO
from xhtml2pdf import pisa
import random
import datetime
from django.contrib import messages
def home(request):
    if request.method == "POST":
        logout(request)
        messages.success(request, "Successfully Logged out!")
        return redirect('home')
    return render(request, 'home.html')
def loginpage(request):
    return render(request, 'login.html')


def view_login(request):
    if request.method == "POST":
        loginusername = request.POST['username']
        loginpass = request.POST['password']
        user = authenticate(username=loginusername, password=loginpass)
        if user is not None:
```

```
            login(request, user)
            messages.success(request, "Successfully Logged in!")
            return redirect('index')
    messages.warning(request, "Login Failed! Please Try Again")
    return render(request, 'login.html')

@login_required(login_url='home')
def logout_user(request):
    if request.method == "POST":
        logout(request)
        messages.success(request, "Successfully Logged out!")
        return redirect('home')

@login_required(login_url='login')
def intermediate(request):
    max_marks = request.POST.get('max_marks', '')
    nques = request.POST.get('nques')
    nq1 = int(request.POST.get('nq1', 0))
    nq2 = int(request.POST.get('nq2', 0))
    nq3 = int(request.POST.get('nq3', 0))
    nq4 = int(request.POST.get('nq4', 0))
    nq5 = int(request.POST.get('nq5', 0))
    params = {
        "nques": nques,
        "max_marks": max_marks,
        "nq1": nq1,
        "nq2": nq2,
        "nq3": nq3,
        "nq4": nq4,
        "nq5": nq5,
    }
    return render(request, 'intermediate.html', params)

@login_required(login_url='login')
def intermediate2(request):
    max_marks = request.POST.get('max_marks', '')
    nq1 = int(request.POST.get('nq1', 0))
    nq2 = int(request.POST.get('nq2', 0))
    nq3 = int(request.POST.get('nq3', 0))
    nq4 = int(request.POST.get('nq4', 0))
    nq5 = int(request.POST.get('nq5', 0))
```

```
    marks1 = int(request.POST.get('marks1', 0))
    marks2 = int(request.POST.get('marks2', 0))
    marks3 = int(request.POST.get('marks3', 0))
    marks4 = int(request.POST.get('marks4', 0))
    marks5 = int(request.POST.get('marks5', 0))
    question_range = range(1, 11)
    return render(request, 'intermediate2.html', {
        'nq1': nq1,
        'nq2': nq2,
        'nq3': nq3,
        'nq4': nq4,
        'nq5': nq5,
        'marks1': marks1,
        'marks2': marks2,
        'marks3': marks3,
        'marks4': marks4,
        'marks5': marks5,
        'question_range': question_range,
        'max_marks': max_marks,
    })
class GeneratePdf(View):
    def get(self, request, *args, **kwargs):
        print('GeneratePaperRequest2')
        tmarks = 0
        dat = datetime.date.today().strftime("%d/%m/%Y")

        # Fetching parameters from the request
        max_marks_input = request.GET.get('max_marks_input', '')
        Year = int(request.GET.get('Year', 0))
        sub = request.GET.get('subject', '')
        ayear = request.GET.get('ayear', '')
        dep = request.GET.get('dep', '')
        test_name = request.GET.get('test_name', '')
        term = request.GET.get('term', '')
        subcode = request.GET.get('subcode', '')
        div = request.GET.get('div', '')
        ttime = request.GET.get('ttime', '')

        # Fetch hours and minutes from GET
        hours = request.GET.get('hours', '0')
        minutes = request.GET.get('minutes', '0')
```

```
    try:
        hours = int(hours)
        minutes = int(minutes)
    except ValueError:
        hours = 0
        minutes = 0

    if hours == 0 and minutes == 0:
        ttime = ""
    elif hours == 0:
        ttime = f"{minutes} minutes"
    elif minutes == 0:
        ttime = f"{hours} hours"
    else:
        ttime = f"{hours} hours {minutes} minutes"

    # Labels up to 10 (a) to (j))
    my_list = ['a)', 'b)', 'c)', 'd)', 'e)', 'f)', 'g)', 'h)', 'i)', 'j)']
    global_selected_questions = []
    # Initialize question lists and counters for each section
    fquestions = {i: [] for i in range(1, 6)}
    qno = {i: 0 for i in range(1, 6)}

    # Dynamically generate question input keys: m1_1 to m5_10, q1_1 to q5_10
    question_inputs = [
        (f"m{section}_{q}", f"q{section}_{q}")
        for section in range(1, 6)
        for q in range(1, 11)
    ]
    for idx, (marks_key, unit_key) in enumerate(question_inputs, start=1):
        marks = request.GET.get(marks_key)
        unit = request.GET.get(unit_key)

        if marks and unit and marks.isdigit() and unit.isdigit():
            a = questionBank.objects.filter(
                year=Year, subname=sub, unit=int(unit), marks=int(marks)
            )
            if not a.exists():
                print(f"No questions found for {marks_key}, {unit_key}")
                continue
            # Select a random question
```

```
            try:
                question = random.choice(list(a))
            except IndexError:
                continue
            section = (idx - 1) // 10 + 1  # 10 questions per section
            if question not in global_selected_questions:
                label = my_list[qno[section]] if qno[section] < len(my_list) else f"{chr(97 + qno[section])})"
                fquestions[section].append((question, label))
                global_selected_questions.append(question)

                tmarks += int(marks)
                qno[section] += 1
        # --- Add 3 extra questions for each main (if available) ---
        for section in range(1, 6):
            current_count = len(fquestions[section])
            extra_needed = 3
            # Try to fetch extra questions for the same marks/unit as inputted
            for i in range(current_count):
                if extra_needed <= 0:
                    break
                orig_question, _ = fquestions[section][i]
                # Find more questions with same year, subname, unit, marks, not already selected
                extra_qs = questionBank.objects.filter(
                    year=Year,
                    subname=sub,
                    unit=orig_question.unit,
                    marks=orig_question.marks
                ).exclude(id__in=[q.id for q, _ in fquestions[section]] + [q.id for q in global_selected_questions])
                for eq in extra_qs:
                    if extra_needed <= 0:
                        break
                    label = my_list[qno[section]] if qno[section] < len(my_list) else f"{chr(97 + qno[section])})"
                    fquestions[section].append((eq, label))
                    global_selected_questions.append(eq)
                    qno[section] += 1
                    extra_needed -= 1
        nq1 = int(request.GET.get('nq1', 0))
        nq2 = int(request.GET.get('nq2', 0))
```

```
nq3 = int(request.GET.get('nq3', 0))
nq4 = int(request.GET.get('nq4', 0))
nq5 = int(request.GET.get('nq5', 0))

marks1 = int(request.GET.get('marks1', 0))
marks2 = int(request.GET.get('marks2', 0))
marks3 = int(request.GET.get('marks3', 0))
marks4 = int(request.GET.get('marks4', 0))
marks5 = int(request.GET.get('marks5', 0))

main1_total = nq1 * marks1
main2_total = nq2 * marks2
main3_total = nq3 * marks3
main4_total = nq4 * marks4
main5_total = nq5 * marks5

year_ = {1: 'First Exam', 2: 'Second Exam', 3: 'Third Exam'}.get(Year, 'BE')

data = {
    "sub": sub,
    "date": dat,
    "ayear": ayear,
    "dep": dep,
    "test_name": test_name,
    "term": term,
    "tmarks": tmarks,
    "subcode": subcode,
    "div": div,
    "ttime": ttime,
    "year": year_,
    "fquestions1": fquestions[1],
    "fquestions2": fquestions[2],
    "fquestions3": fquestions[3],
    "fquestions4": fquestions[4],
    "fquestions5": fquestions[5],
    "max_marks_input": max_marks_input,
    "nq1": nq1,
    "nq2": nq2,
    "nq3": nq3,
    "nq4": nq4,
    "nq5": nq5,
```

```
            "marks1": marks1,
            "marks2": marks2,
            "marks3": marks3,
            "marks4": marks4,
            "marks5": marks5,
            "main1_total": main1_total,
            "main2_total": main2_total,
            "main3_total": main3_total,
            "main4_total": main4_total,
            "main5_total": main5_total,
        }
        print("Data passed to template:", data)

        pdf = render_to_pdf('pdf/invoice.html', data)
        if pdf:
            return HttpResponse(pdf, content_type='application/pdf')
        else:
            return HttpResponse("Error generating PDF", status=500)
@login_required(login_url='login')
def index(request):
    return render(request, 'index.html')
@login_required(login_url='login')
def generatePaper(request):
    print('GeneratePaperRequest')
    return render(request, 'generatePaper.html')
@login_required(login_url='login')
def delete(request):
    return render(request, 'delete.html')
@login_required(login_url='login')
def deleteQuestion(request):
    year = request.POST.get('year')
    subname = request.POST.get('subname')
    unit = request.POST.get('unit')

    # Validate input data
    if not year and not subname and not unit:
        a = questionBank.objects.all()
    elif not year and not subname:
        a = questionBank.objects.filter(unit=unit)
    elif not subname and not unit:
        a = questionBank.objects.filter(year=year)
```

```
    elif not year and not unit:
        a = questionBank.objects.filter(subname=subname)
    elif not unit:
        a = questionBank.objects.filter(year=year, subname=subname)
    elif not year:
        a = questionBank.objects.filter(subname=subname, unit=unit)
    else:
        a = questionBank.objects.filter(year=year, subname=subname, unit=unit)  # Properly
indented

    print(a)
    params = {"a": a, "subname": subname, "year": year, "unit": unit}
    return render(request, 'deleteQuestion.html', params)
@login_required(login_url='login')
def deleteSuccess(request):
    if request.method == "POST":
        ids = request.POST.getlist('ids_to_delete')
        if ids:
            for id in ids:
                try:
                    questionBank.objects.get(id=id).delete()
                except questionBank.DoesNotExist:
                    pass
            messages.success(request, "Selected questions deleted successfully.")
        else:
            messages.warning(request, "No questions selected for deletion.")
        return redirect('deleteQuestion')
    return redirect('deleteQuestion')
@login_required(login_url='login')
def add_question(request):
    if request.method == "POST":
        year = request.POST.get('year')
        subject = request.POST.get('subject')
        subject_code = request.POST.get('subject_code')
        unit = request.POST.get('unit')
        marks = request.POST.get('marks')
        question_text = request.POST.get('question')

        # Validate input data
        if not (year and subject and subject_code and unit and marks and question_text):
            messages.error(request, "All fields are required!")
```

```
            return redirect('add_question')
        try:
            # Save the question to the database
            question = questionBank(
                question=question_text,
                subject_code=subject_code,
                marks=marks,
                unit=unit,
                year=year,
                subname=subject
            )
            question.save()

            # Add a success message
            messages.success(request, "Question saved successfully!")
        except Exception as e:
            # Handle any unexpected errors
            messages.error(request, f"An error occurred: {str(e)}")

        # Redirect back to the add_question page
        return redirect('add_question')

    return render(request, 'add_question.html')
def view_questions(request):
    questions = questionBank.objects.all()
    return render(request, 'view_questions.html', {'questions': questions})
def edit_question(request, id):
    question = get_object_or_404(questionBank, id=id)
    if request.method == "POST":
        question.question = request.POST.get('question')
        question.subject_code = request.POST.get('subject_code')
        question.marks = request.POST.get('marks')
        question.year = request.POST.get('year')
        question.unit = request.POST.get('unit')
        question.subname = request.POST.get('subject')  # or question.subject = ... if your model
uses 'subject'
        question.save()
        messages.success(request, "Question updated successfully!")
        return redirect('view_questions')
    # For GET request, render the edit form
    return render(request, 'edit_question.html', {'question': question})
```

12. CONCLUSION AND FUTURE WORK

Conclusion:

The **Smart question paper generator** is a valuable web application designed to assist administrators in quickly and efficiently creating question papers by selecting from a centralized database of stored questions. Developed using Django, a robust Python web framework, and MySQL as the backend database, the system streamlines the entire paper creation process, saving significant time and effort. Administrators can specify key parameters such as the subject and the number of questions required, and the system automatically generates well-structured question papers that can be easily reviewed and exported in a neat PDF format, ready for printing. This automation not only accelerates the workflow but also helps maintain consistency and organization, reducing the stress typically associated with manual paper setting. The application's clean and user-friendly web interface, coupled with cloud hosting capabilities, ensures that it is accessible from anywhere, making it especially suitable for colleges and educational institutions with multiple campuses or remote administration needs. By maintaining a centralized question bank, the system minimizes the risks of repeating questions or omitting important topics, thereby improving the overall quality and fairness of the exam papers. This approach also facilitates better quality control and ensures balanced coverage of subjects and difficulty levels. Overall, the Smart question paper generator enhances the efficiency, accuracy, and accessibility of exam preparation, making it an essential tool for modern educational environments.

Future Work:

Future updates can improve the Intelligent Smart question paper generator system by adding user roles like admin and teacher, better organizing questions by topic and difficulty, and enabling bulk uploads through Excel. AI can help auto-select balanced questions, while a review system ensures quality. Support for images, equations, and multiple languages like Hindi, Kannada makes it more versatile. Collaboration features, reports, and a mobile app will enhance usability and accessibility.

13. BIBLIOGRAPHY

The implementation of the Smart question paper generator project was made possible through the use of several key technologies and reference materials. The official Django documentation offered by the Django Software Foundation was vital in understanding the framework's Model-View-Template (MVT) architecture, authentication system, and admin panel capabilities. MySQL documentation from Oracle provided detailed guidance for managing relational databases and executing SQL queries efficiently. For frontend development, resources such as W3Schools and Mozilla Developer Network (MDN) were used extensively to implement HTML, CSS, and JavaScript for creating responsive and user-friendly interfaces. The xhtml2pdf library documentation helped in the generation of printable PDF documents. Additionally, educational websites like GeeksforGeeks and community forums like Stack Overflow were invaluable for solving practical issues and learning best practices in Django, Python, and database integration. The Python Software Foundation's official documentation also supported the development with clear explanations of core Python concepts and libraries used throughout the project.