

A Project Report
On
Manual and Automatic Question Paper Generation System

Submitted to
Acharya Nagarjuna University

In Partial Fulfillment of the Requirement for the Award of
Bachelor's Degree in

Information Technology

by

Leemoji Reddy Appala	Y20AIT462
S.Balakrishna Reddy	Y20AIT498
M.Pujitha	Y20AIT467
V.Venkat Srinivas	Y20AIT515

Under the guidance of
Dr. B. Krishnaiah, M.E, Ph. D
Assistant Professor



Department of Information Technology
Bapatla Engineering College (Autonomous)
Mahatmaji Puram, Bapatla - 522102
2023-2024
Affiliated to
Acharya Nagarjuna University

Bapatla Engineering College

(Autonomous)

Department of Information Technology

Mahatmaji Puram, Bapatla -522102



CERTIFICATE

This is to certify that the project entitled

“Manual and Automatic Question Paper Generation System”

Leemoji Reddy Appala Y20AIT462

is a record of Bonafide work carried out by him, in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (Information Technology) at BAPATLA ENGINEERING COLLEGE, Bapatla under Acharya Nagarjuna University during the academic year 2023-2024, under our guidance.

Date:

Asst. Prof. Dr. B. Krishnaiah

Project Guide

Asst. Prof. Dr. B. Krishnaiah

Project coordinator

Dr. N. Sivaram Prasad
Prof & H.O.D, Dept of IT

Sign of External Examiner

DECLARATION

We declare that this project report titled “**Manual and Automatic Question Paper Generation**” submitted in partial fulfillment of the degree of **B. Tech in Information Technology** is a record of original work carried out by me under the supervision of **Dr. B. Krishnaiah, Asst.Professor** and has not formed the basis for the award of any other degree or diploma, in this or any other Institution or University. In keeping with the ethical practice in reporting scientific information, due acknowledgments have been made wherever the findings of others have been cited.

Date:

Leemoji Reddy Appala (Y20AIT62)

Place: Bapatla

S.Balakrishna Reddy(Y20AIT498)

M.Pujitha(Y20AIT467)

V.Venkat Srinivas(Y20AIT515)

ACKNOWLEDGEMENT

We are profoundly grateful to **Dr. B. Krishnaiah, Asst.Professor** for his expert guidance and continuous encouragement throughout to see that this Term Paper rights its target since its commencement to its completion.

We would like to express deepest appreciation towards **Dr. Nazeer Shaik**, Principal, Bapatla Engineering College, **Dr. N. Sivaram Prasad**, Head of Department of Information Technology and **Dr. B. Krishnaiah, Asst.Professor** Project Coordinator whose invaluable guidance supported us in completing this term paper.

At last we must express our sincere heartfelt gratitude to all the staff members of Information Technology Department who helped us directly or indirectly during this course of work.

Leemoji Reddy Appala (Y20AIT62)

S.Balakrishna Reddy(Y20AIT498)

M.Pujitha(Y20AIT467)

V.Venkat Srinivas(Y20AIT515)

ABSTRACT

The "Manual and Automatic Question Paper Generation System" is a web-based application designed to streamline the process of creating question papers for academic assessments. The system offers both manual and automatic modes for generating question papers, catering to the diverse needs of educators and administrators.

In manual mode, users have the flexibility to craft custom question papers by selecting and arranging questions from a variety of categories. On the other hand, the automatic mode leverages a database of pre-existing questions to generate question papers efficiently, saving time and effort.

The user interface is intuitive and user-friendly, featuring a responsive design to ensure compatibility across different devices. Navigation is facilitated through a sidebar menu and a top navigation bar, providing easy access to key functionalities such as creating question papers and managing user profiles.

Additionally, the system incorporates visually appealing elements, such as card-based layouts and inspirational quotations, to enhance user engagement and motivation.

By combining manual and automatic question paper generation capabilities within a cohesive and visually appealing interface, this system aims to streamline the assessment process for educational institutions, empowering educators to focus more on teaching and learning.

TABLE OF CONTENTS

S.NO	DESCRIPTION	PAGE NO
	CERTIFICATE -----	i
	DECLARATION -----	ii
	ACKNOWLEDGEMENT -----	iii
	ABSTRACT -----	iv
	LIST OF FIGURES -----	vii
	LIST OF ABBREVIATIONS -----	viii
1	INTRODUCTION	1
	1.1 Purpose-----	1
	1.2 Scope-----	2
	1.3 Overview-----	2
2	LITERATURE SURVEY	3
	2.1 PostgreSQL-----	3
	2.2 Django-----	3
	2.3 HTML,CSS,JavaScript(Frontend Technologies)-----	3
	2.4 Bootstrap-----	3
	2.5 Existing Question Paper Generation Systems-----	4
3	SYSTEM ANALYSIS	6
	3.1 Existing System-----	6
	3.2 Proposed System-----	7
4	REQUIREMENT ANALYSIS	8
	4.1 Software Requirements-----	8
	4.1.1 Frontend-----	8
	4.1.2 Backend-----	8
	4.1.3 Database-----	8
	4.1.4 Software Installations-----	9
	4.1.4.1 PostgreSQL-----	9
	4.1.4.2 Visual Studio Code-----	12
	4.2 Hardware Requirements-----	15
	4.2.1 PostgreSQL Workbench-----	15
	4.2.2 Django-----	15
	4.2.3 HTML,CSS,JS & Bootstrap-----	15
5	SYSTEM DESIGN	16
	5.1 UML Diagrams-----	16
	5.1.1 Use Case Diagram-----	16
	5.1.2 Data Flow Diagram-----	16
	5.1.3 Activity Diagram-----	17
	5.1.4 Component Diagram-----	18

6	PROJECT PLANNING	19
6.1	User Perspective-----	19
6.2	Admin Perspective-----	20
7	IMPLEMENTATION	21
7.1	Main Component-----	21
7.1.1	Automatic_Question_Paper_Generation_sub.html-----	21
7.1.2	notes_generation.html-----	34
7.1.3	models.py-----	39
7.1.4	views.py-----	42
7.1.5	urls.py-----	51
8	SYSTEM TESTING	52
8.1	Testing Methodologies-----	52
8.1.1	Unit Testing-----	52
8.1.2	Integration Testing-----	52
8.1.3	System Testing-----	52
9	RESULT & DISCUSSION	53
10	CONCLUSION & FUTURE SCOPE	58
10.1	Conclusion-----	58
10.2	Future Scope-----	58
	REFERENCES	59

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE NO
4.1	Installer file and an installation wizard	9
4.2	Locate the installer file you just downloaded and double-click it to run the installer.	9
4.3	Selection Component.Data Directory	10
4.4	Choose the database directory to store the data	10
4.5	Specify a port number on which the PostgreSQL database server.	11
4.6	Select the default locale for the PostgreSQL server	11
4.7	Installing Progress	12
4.8	Download Windows Version of VS Code.	12
4.9	Acceptance of Agreement.	13
4.10	Acceptance of Agreement Stage-2.	13
4.11	Installation Setup Process.	14
4.12	Dashboard of VS Code	14
5.1	Use case diagram	16
5.2	Data Flow diagram	16
5.3	Activity of Notes Generation from Database Module Flow	17
5.4	Activity Generate Question Paper from Database Module Flow	17
5.5	Activity of Manual Generating Question Paper Module Flow	18
5.6	Component Diagram	18
9.1	Singin/Signup page	53
9.2	Dashboard Page	53
9.3	Profile page	54
9.4	Manual Question Paper Creation Page	54
9.5	Automatic Question Paper Generation	55
9.6	Question Papers	55
9.7	Notes Generation	56
9.8	Preview	56
9.9	Question paper download	57

LIST OF ABBREVIATIONS

ABBREVIATION

HTML
CSS
SQL
ORM
UML
JSON
DOM
AJAX
CPU
RAM

FULL FORM

Hypertext Markup Language
Cascading Style Sheets
Structured Query Language
Object-Relational Mapper
Unified Modelling Language
JavaScript Object Notation
Document Object Model
Asynchronous JavaScript and XML
Central Processing Unit
Random-Access Memory

CHAPTER 1

INTRODUCTION

The Manual and Automatic Question Paper Generation System is a sophisticated web-based application aimed at revolutionizing the process of creating academic question papers. Educational institutions are often burdened with the task of generating question papers that are diverse, comprehensive, and align with curriculum standards. Traditional methods of manual question paper creation can be time-consuming and prone to errors, while existing automatic generation systems may lack flexibility and customization options.

This project seeks to bridge this gap by offering a comprehensive solution that combines the benefits of both manual and automatic question paper generation. By leveraging modern web technologies such as Django, PostgreSQL, HTML, CSS, and JavaScript, this system provides educators and administrators with a user-friendly platform to efficiently generate high-quality question papers tailored to their specific requirements.

The system's frontend is built using HTML, CSS, and JavaScript, ensuring a responsive and intuitive user interface. Through features like filters, sorting options, and interactive elements, users can easily navigate through the system and customize their question papers according to various parameters such as subject, difficulty level, and question type.

At the backend, Django, a powerful Python-based web framework, serves as the middleware tool for handling communication between the frontend and the database. PostgreSQL, a robust open-source relational database management system, securely stores and manages all question paper data, ensuring reliability and scalability.

The Manual and Automatic Question Paper Generation System not only simplifies the process of creating question papers but also offers valuable insights into the educational assessment process. By analyzing past question paper data and performance metrics, educators can identify patterns, assess student learning outcomes, and make data-driven decisions to improve teaching and learning.

1.1 Purpose

The Manual and Automatic Question Paper Generation System is developed with the primary purpose of revolutionizing the process of creating academic question papers within educational institutions. By combining manual expertise with automated features, the system aims to streamline and enhance the efficiency of question paper generation. This purpose is driven by the recognition of the time-consuming nature of traditional methods and the need for more flexible and customizable solutions.

1.2 Scope

Question Paper Generation Modes:

Manual Mode: Users can create custom question papers by selecting and arranging questions from various categories, allowing for greater control and flexibility.

Automatic Mode: The system automatically generates question papers using a database of pre-existing questions, significantly reducing the time required for paper preparation.

Question Database:

A comprehensive question bank that allows users to add, edit, or remove questions. The questions are categorized based on subject, difficulty level, and type (e.g., multiple-choice, short answer, essay, etc.).

The database facilitates efficient search and retrieval of questions, enabling seamless paper generation.

Additional Functionalities:

Notes Generation:

In addition to generating question papers, the system provides a functionality for creating educational notes. Users can compile notes related to specific topics, allowing educators to distribute supplementary learning material alongside question papers.

Export and Print Options:

The ability to export question papers and notes in various formats (e.g., PDF) for ease of printing or digital distribution.

Collaboration and Sharing:

The system may allow collaborative question paper creation and note-sharing among educators, promoting teamwork and consistent assessment practices.

1.3 Overview

Built on modern web technologies such as Django, PostgreSQL, HTML, CSS, and JavaScript, the system offers a user-friendly interface for seamless navigation and customization. Through intuitive features such as filters, sorting options, and interactive elements, users can easily tailor question papers to their specific requirements. The backend infrastructure ensures robust data management and security, facilitating reliable storage and retrieval of question paper data. Furthermore, the system's data analysis capabilities provide valuable insights into educational assessment processes, enabling educators to make informed decisions based on performance metrics and past question paper data.

CHAPTER 2

LITERATURE SURVEY

2.1 PostgreSQL

PostgreSQL is a robust open-source relational database management system (RDBMS) known for its reliability, extensibility, and SQL compliance. It offers advanced features such as ACID (Atomicity, Consistency, Isolation, Durability) transactions, robust concurrency control, and support for complex data types like JSON and arrays. PostgreSQL's scalability capabilities allow for efficient handling of large datasets and high traffic volumes through features like table partitioning and parallel query execution[1].

2.2 Django

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It follows the Model-View-Controller (MVC) architectural pattern and includes built-in features such as an object-relational mapper (ORM), URL routing, form handling, and authentication. Django's "batteries-included" philosophy provides developers with a rich set of tools and libraries, allowing for the creation of complex web applications with minimal code duplication[2].

2.3 HTML, CSS, JavaScript (Frontend Technologies)

HTML, (Hypertext Markup Language), CSS (Cascading Style Sheets), and JavaScript form the core technologies for building the frontend of web applications[3]. HTML provides the structure of web pages, CSS styles the presentation[4], and JavaScript adds interactivity and dynamic behavior[5]. Together, these technologies enable the creation of visually appealing and responsive user interfaces. Frameworks and libraries such as Bootstrap and jQuery further enhance frontend development by providing pre-designed components and simplifying DOM manipulation.

2.4 Bootstrap

Bootstrap is a popular front-end framework for building responsive and mobile-first web projects. It offers a collection of CSS and JavaScript components, including grids, forms, buttons, and navigation bars, that streamline the development process and ensure consistency across different devices and screen sizes. Bootstrap's modular and customizable nature allows developers to create visually appealing and user-friendly interfaces with minimal effort[6].

This literature survey provides an overview of the key technologies and frameworks utilized in the "Manual and Automatic Question Paper Generation System" project, encompassing backend database management (PostgreSQL), web development framework (Django), frontend development (HTML, CSS, JavaScript), and frontend libraries (Bootstrap, jQuery). These technologies collectively contribute to the development of a robust, scalable, and user-friendly application for generating question papers effectively.

2.5 Existing Question Paper Generation Systems

The “Question Paper Generator System” has provided a ready to use built-in question bank [7]. The paper aptly describes CQZ (Cloze Question Generation) putting more emphasis on the actual type of the questions.

The paper on “An Integrated Automated Paperless Academic Module for Education Institutes” has stated the importance and working of switching from Paper-based systems to Paperless Systems [8]. The importance of automation is very well documented in the context of Task Engineering. The paper also clearly defines the importance of Information and Communication Technology (ICT) in academics and educational organizations.

The research paper “Framework for Automatic Examination Paper Generation System” has provided a thorough insight into the process of automated paper generation [9]. As the manual generation of a balanced question paper by an individual is quite complex, the blending of technology into teaching and learning process is inevitable.

The paper on “Automatic Question Paper Generation System using Randomization Algorithm” describes a system which uses a shuffling algorithm (existing algorithm) as a randomization technique [10].

Another paper on “Automatic Test Paper Generation Based on Ant Colony Algorithm” has implemented a complex but highly efficient Ant Colony Algorithm [11]. It requires the building of a mathematical model of constraint according to the requirements of the paper. This paper provides an efficient solution with their algorithm.

In exploring sentiment classification, Abdi et al. employ deep learning methods with multi-feature fusion[12], while Agarwal delves into question generation systems and evaluation guidelines, emphasizing cloze and open cloze formats[13]. Agarwal and Mannem focus on automatic gap-fill question generation from textbooks[14], Aldabe and Maritxalar discuss automatic distractor

generation for domain-specific texts[15], and Alruwais et al. examine the advantages and challenges of using e-assessment[16].

Amidei, Piwek, and Willis explore evaluation methodologies in automatic question generation from 2013 to 2018[17], focusing on techniques discussed at the 11th International Natural Language Generation Conference. Antol et al. present VQA, a system for visual question answering, showcased at the IEEE International Conference on Computer Vision[18]. Bhatia, Kirti, and Saha delve into the automatic generation of multiple-choice questions using Wikipedia as a resource, discussed at the Pattern Recognition and Machine Intelligence conference[19].

On a separate note, Bilker et al. develop abbreviated nine-item forms of the Raven's standard progressive matrices test, aiming at efficient assessment techniques[20]. Bin et al. propose automated essay scoring using the KNN algorithm, presented at the 2008 International Conference on Computer Science and Software Engineering[21].

CHAPTER 3

SYSTEM ANALYSIS

3.1 Existing System

The current system utilized by our college for question paper creation relies predominantly on traditional methods, notably utilizing word processing software like Microsoft Word. While this approach has served its purpose, it presents certain limitations and drawbacks:

- **Manual Creation Process:** Question papers are manually created by faculty members or administrative staff using word processing software. This manual process involves formatting questions, organizing sections, and ensuring proper layout and presentation.
- **Limited Collaboration:** Collaboration among faculty members or departments in question paper creation is constrained by the manual nature of the process. Sharing and editing documents can be cumbersome and time-consuming, leading to inefficiencies in the overall process.
- **Version Control Challenges:** Keeping track of different versions of question papers, especially during revisions or updates, can be challenging. Without proper version control mechanisms, there is a risk of errors or inconsistencies in the question papers.
- **Time-Consuming Revisions:** Making revisions or updates to question papers requires manual editing, which can be time-consuming. Faculty members or staff need to manually make changes to the document, review them, and ensure accuracy before finalizing the revised version.
- **Limited Flexibility:** Word processing software provides limited flexibility in terms of dynamic content generation or customization. Question papers may lack interactive elements or dynamic features that could enhance the assessment process.

Overall, while the current system facilitates the creation of question papers, it is limited by manual processes, lack of collaboration features, version control challenges, and limited flexibility. Upgrading to a more advanced and automated system could address these limitations and improve efficiency, collaboration, and accuracy in question paper creation.

3.2 Proposed System

The proposed system, the "Manual and Automatic Question Paper Generation System," aims to revolutionize the process of question paper creation by introducing advanced features and automation capabilities. Key components of the proposed system include:

- **User-Friendly Interface:** The system will feature an intuitive and user-friendly interface accessible via web browsers. Faculty members and administrative staff will be able to easily navigate the system and access its functionalities.
- **Manual Question Paper Creation:** The system will allow users to manually create question papers using a rich text editor integrated into the platform. Users will have the flexibility to format questions, add sections, and customize the layout according to their preferences.
- **Automatic Question Paper Generation:** Leveraging advanced algorithms and data processing techniques, the system will offer automatic question paper generation capabilities. Users will have the option to generate question papers based on predefined criteria such as topic, difficulty level, question type, and more.
- **Integration with Database:** The system will be integrated with a centralized database containing a repository of questions, topics, and other relevant data. This integration will enable users to access a vast pool of resources for question paper creation and streamline the process.
- **Collaboration and Sharing:** The system will facilitate collaboration among faculty members and departments by providing features for sharing, reviewing, and editing question papers in real-time. Version control mechanisms will ensure that all changes are tracked and documented.
- **Advanced Filtering and Sorting:** Users will have access to advanced filtering and sorting options to effectively manage and organize question papers. They will be able to filter question papers based on criteria such as subject, course, date, and more, enabling efficient retrieval of relevant information.
- **Dynamic Content Generation:** The system will support dynamic content generation, allowing users to include interactive elements such as multimedia, images, and diagrams in question papers. This will enhance the quality and interactivity of question papers, resulting in a more engaging assessment experience.

Overall, the proposed system represents a significant advancement over existing methods of question paper creation, offering enhanced functionality, automation, collaboration, and security. By adopting this system, our college can streamline the process of question paper creation, improve efficiency, and enhance the overall assessment experience for students.

CHAPTER 4

REQUIREMENT ANALYSIS

4.1 Software Requirements

The software requirements for the Manual and Automatic Question Paper Generation System project outline the necessary tools, technologies, and libraries for developing and deploying the web-based application.

4.1.1 Frontend

- **HTML:** HTML will be used to structure the web pages and provide semantic elements for the portal's interface, ensuring accessibility and proper document structure.
- **CSS:** CSS is utilized for styling HTML elements, ensuring a visually appealing and cohesive user interface.
- **JavaScript:** JavaScript is employed to add dynamic behavior to the frontend, enabling features such as form validation, interactive elements, and real-time updates.
- **Bootstrap:** Bootstrap is integrated into the project to expedite frontend development, offering pre-designed components and responsive layouts for streamlined UI implementation.
- **jQuery:** jQuery can be used for simplifying JavaScript code and handling DOM manipulation, event handling, and AJAX requests more efficiently

4.1.2 Backend

- **Django:** Django, a Python-based web framework, serves as the backbone of the backend, providing a robust infrastructure for handling server-side logic, routing, and data management.
- **Python:** Python acts as the primary programming language for backend development, offering flexibility, ease of use, and extensive libraries for diverse functionalities.

4.1.3 Database

- **PostgreSQL:** PostgreSQL is designated as the preferred database management system for storing and managing student data. Its advanced features, reliability, and scalability make it well-suited for handling the project's requirements.
- **PgAdmin:** PgAdmin is utilized as a graphical administration tool for PostgreSQL, enabling database administrators to perform tasks such as database design, query execution, and performance optimization.

4.1.4 Software Installations

4.1.4.1 PostgreSQL

1. PostgreSQL Installer

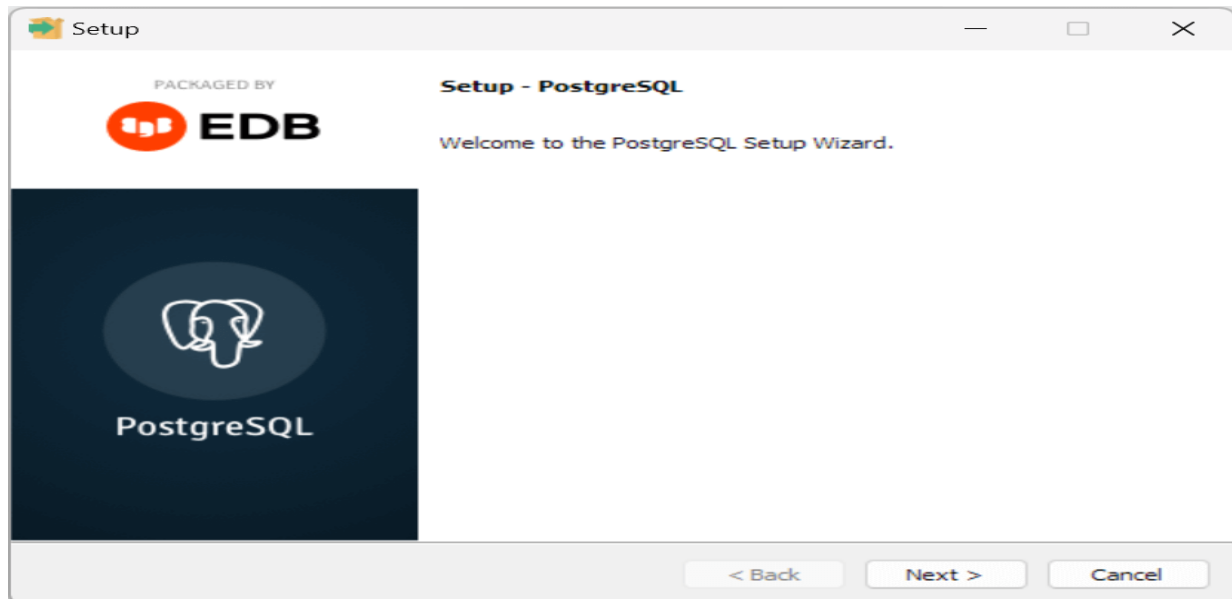


Figure 4.1 Installer file and an installation wizard

2. Installation Directory

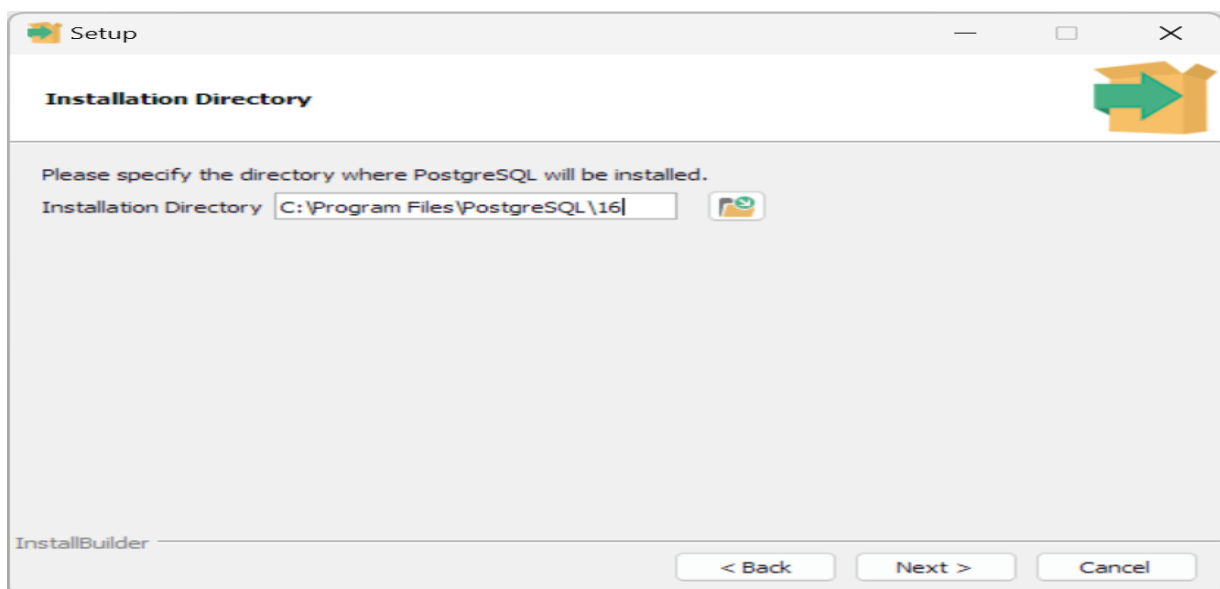


Figure 4.2 Locate the installer file you just downloaded and double-click it to run the installer.

3. Select Components

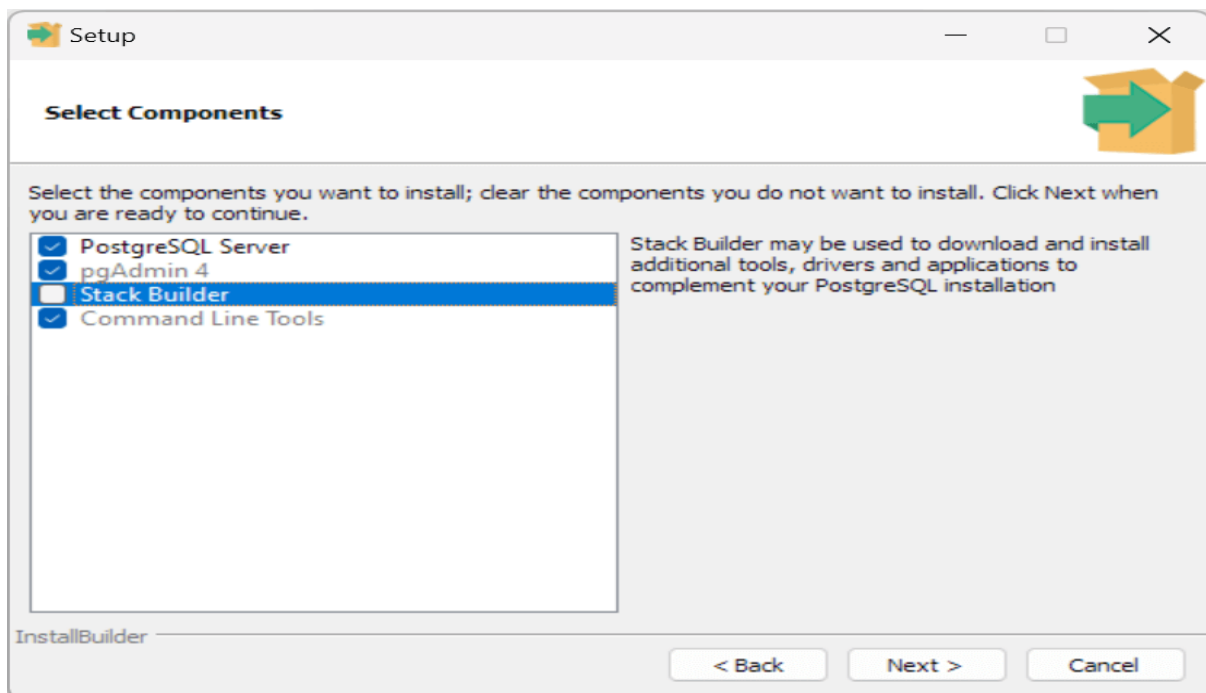


Figure 4.3 Selection Component.Data Directory

4.Data Directory

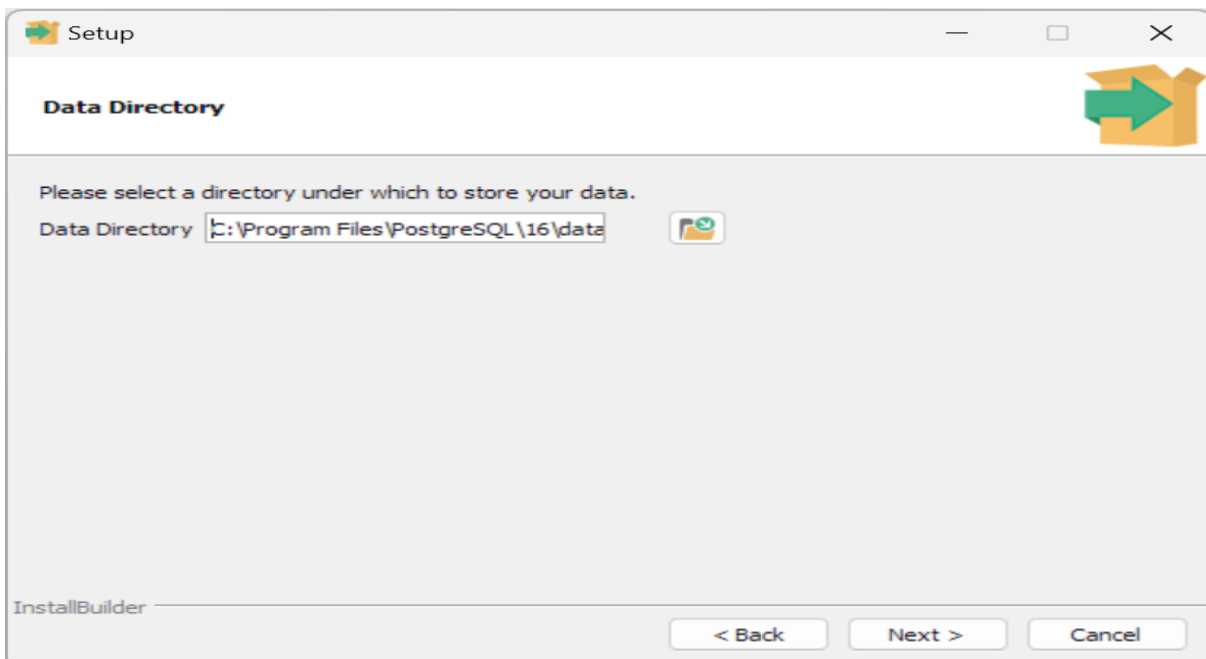


Figure 4.4 Choose the database directory to store the data

5. Specify a port number

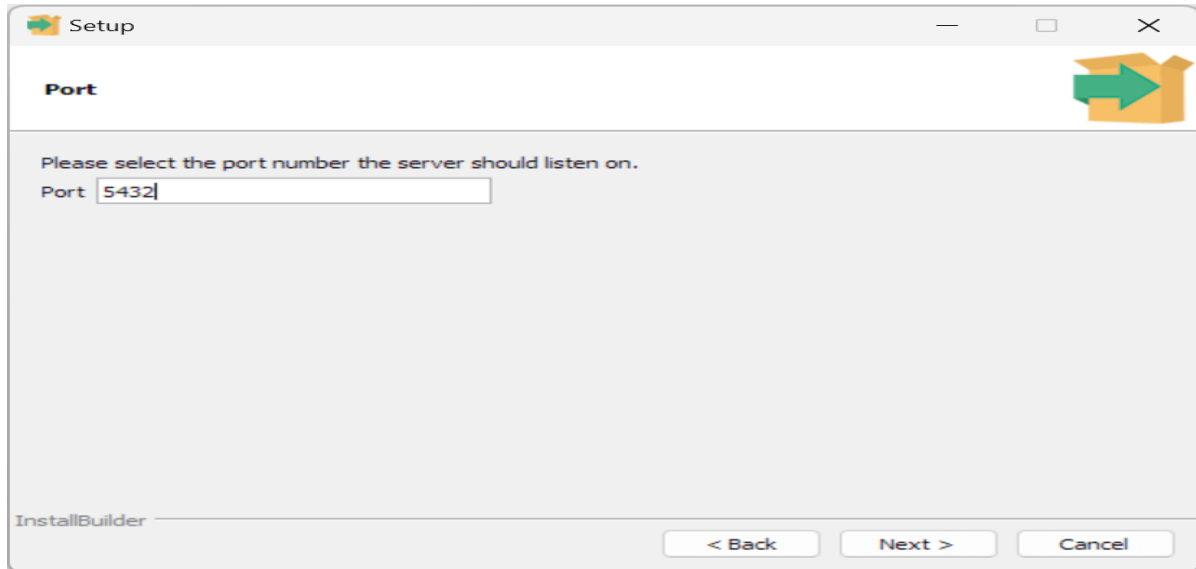


Figure 4.5 Specify a port number on which the PostgreSQL database server.

6. Default locale for the PostgreSQL

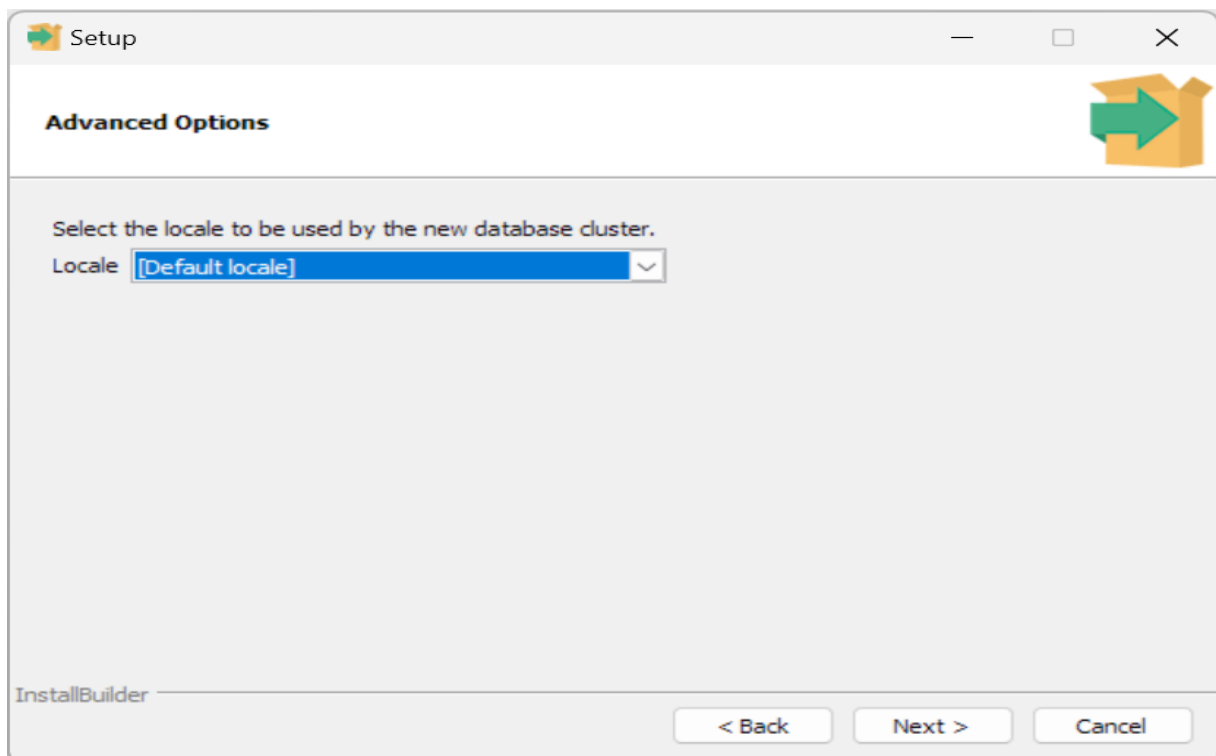


Figure 4.6 Select the default locale for the PostgreSQL server

7.Final Installation

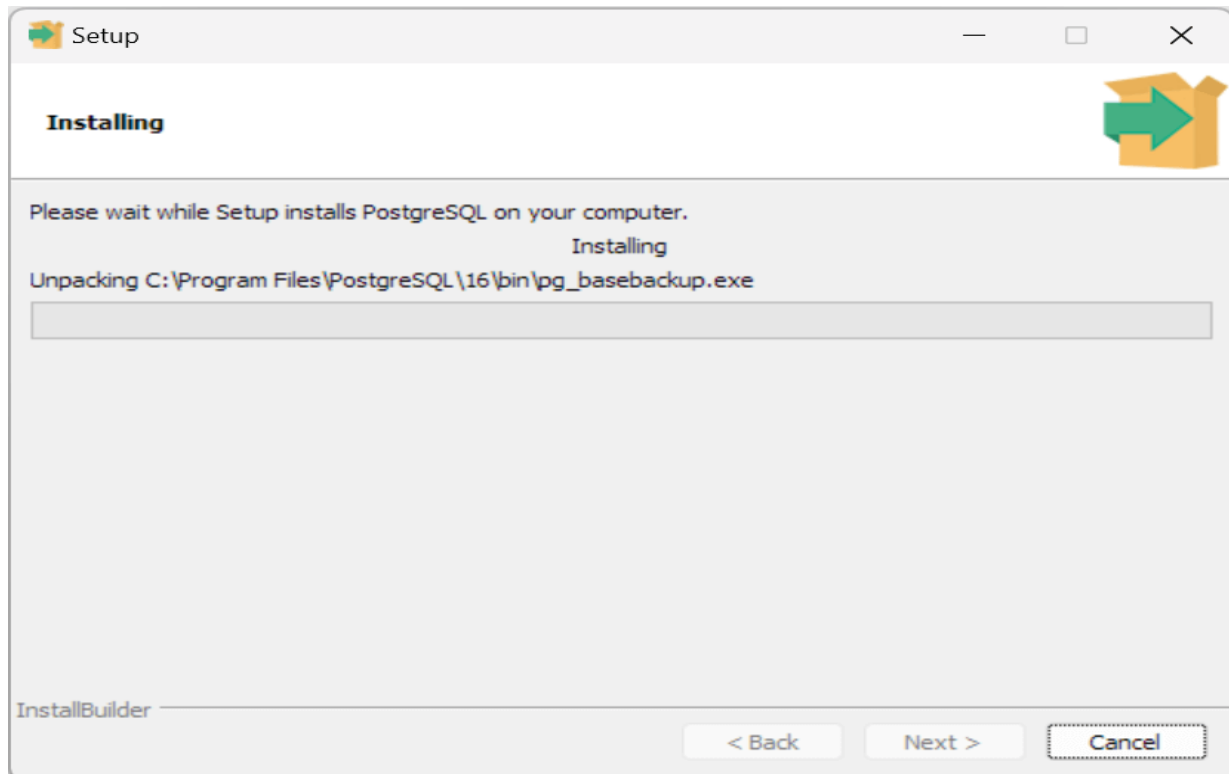


Figure 4.7 Installing Progress

4.1.4.2 Visual Studio Code

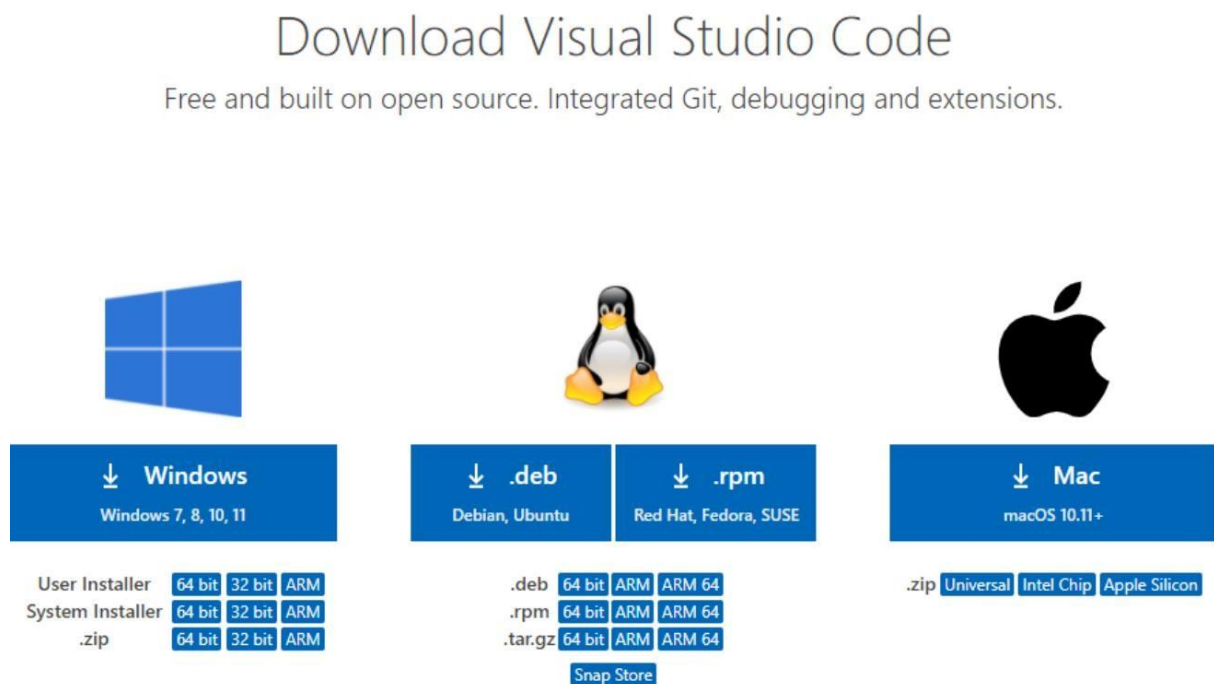


Figure 4.8 Download Windows Version of VS Code.

1. Accept The Agreement

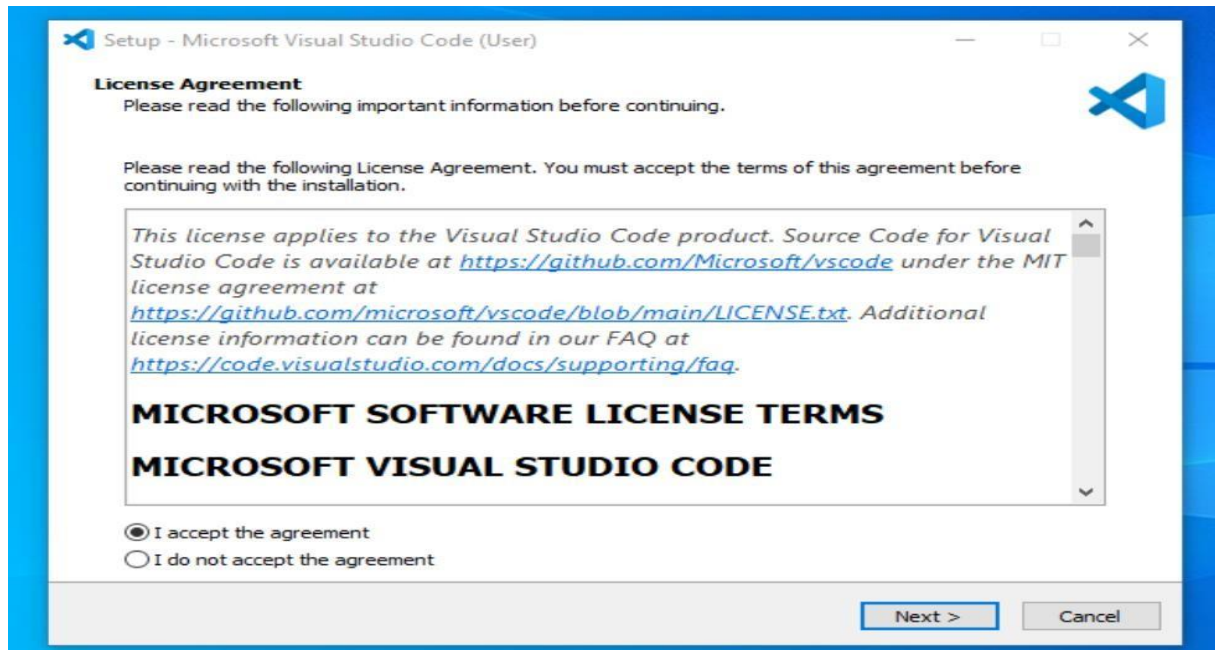


Figure 4.9 Acceptance of Agreement.

2. Choosing Location

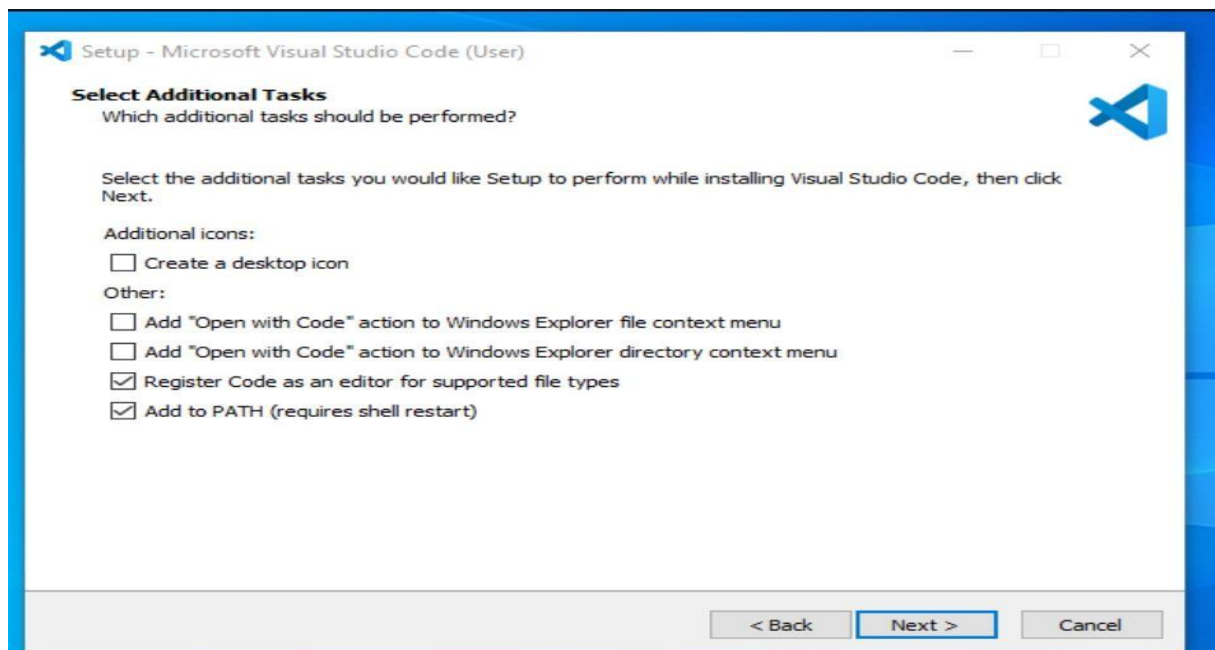


Figure 4.10 Acceptance of Agreement Stage-2.

3. Begin the Installation Setup

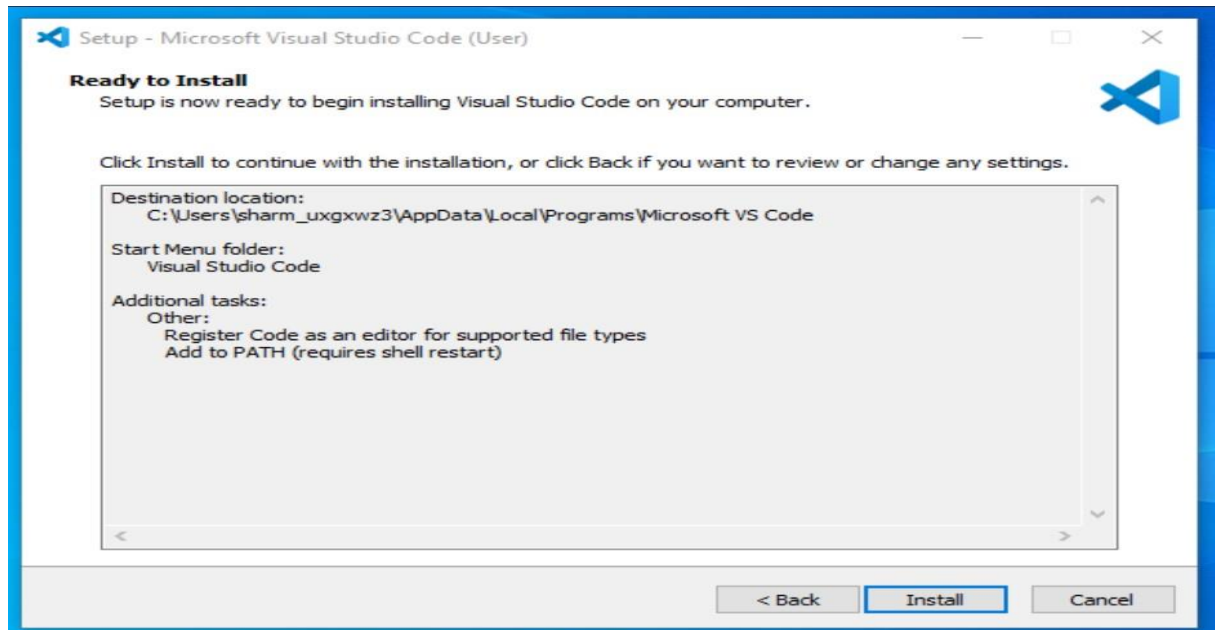


Figure 4.11 Installation Setup Process.

4. Visual Studio Dashboard

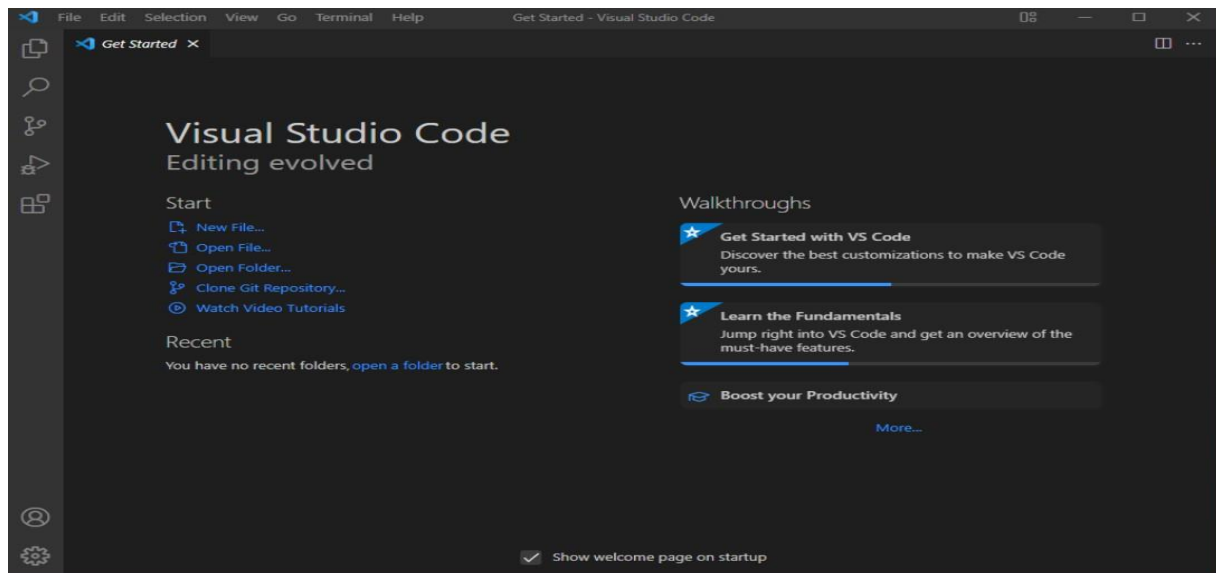


Figure 4.12 Dashboard of VS Code

4.2 Hardware Requirements

4.2.1 PostgreSQL Workbench

- **CPU:** Intel Core or Xeon 3GHz (or Dual Core 2GHz) or equal AMD CPU.
- **Cores:** Single (Dual/Quad Core is recommended)
- **RAM:** 4 GB (6 GB recommended)
- **Graphic Accelerators:** nVidia or ATI with support of OpenGL 1.5 or higher.
- **Display Resolution:** 1280×1024 is recommended, 1024×768 is minimum.

4.2.2 Django

- **CPU:** Any modern multi-core CPU is sufficient for running Django applications.
- **RAM:** Minimal RAM needed, around 1 GB for simple applications; more for complex ones.
- **Storage:** Storage requirements depend on the application; ensure sufficient space for data and logs.
- **Web Server:** Use a production-grade server like Nginx or Apache instead of Flask's development server.
- **Python Version:** Flask supports Python 3.7 and above.

4.2.3 HTML,CSS,JS & Bootstrap

- **CPU:** Any modern CPU can handle HTML, CSS, and JavaScript. The performance requirements depend on the complexity of the code and the website.
- **RAM:** For development, at least 4 GB of RAM is recommended for smooth performance when running a code editor and browser simultaneously..
- **Graphics:** A standard graphics card is sufficient for development and running websites. For complex visual effects, better graphics cards may improve performance.
- **Browser:** Modern browsers such as Chrome, Firefox, Safari, and Edge are needed to run HTML, CSS, JavaScript, and Bootstrap efficiently.
- **Operating System:** These technologies are compatible with any operating system, including Windows, macOS, and Linux.

CHAPTER 5

SYSTEM DESIGN

5.1 UML Diagram

A UML diagram is a diagram based on the UML (Unified Modeling Language) with the purpose of visually representing a system along with its main actors, roles, actions, artifacts, or classes, in order to better understand, alter, maintain, or document information about the system.

5.1.1 Use Case Diagram

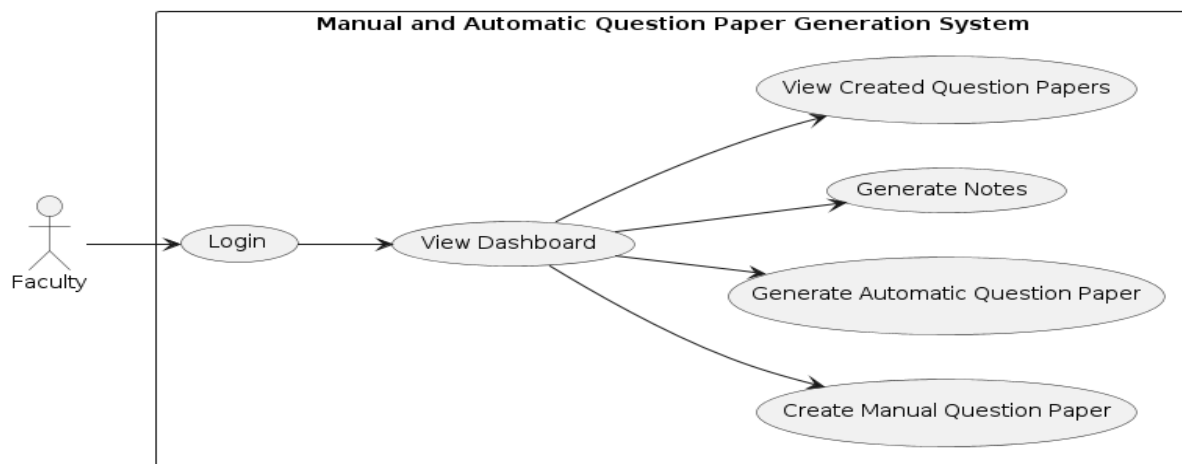


Figure 5.1 Use case diagram

5.1.2 Data Flow Diagram

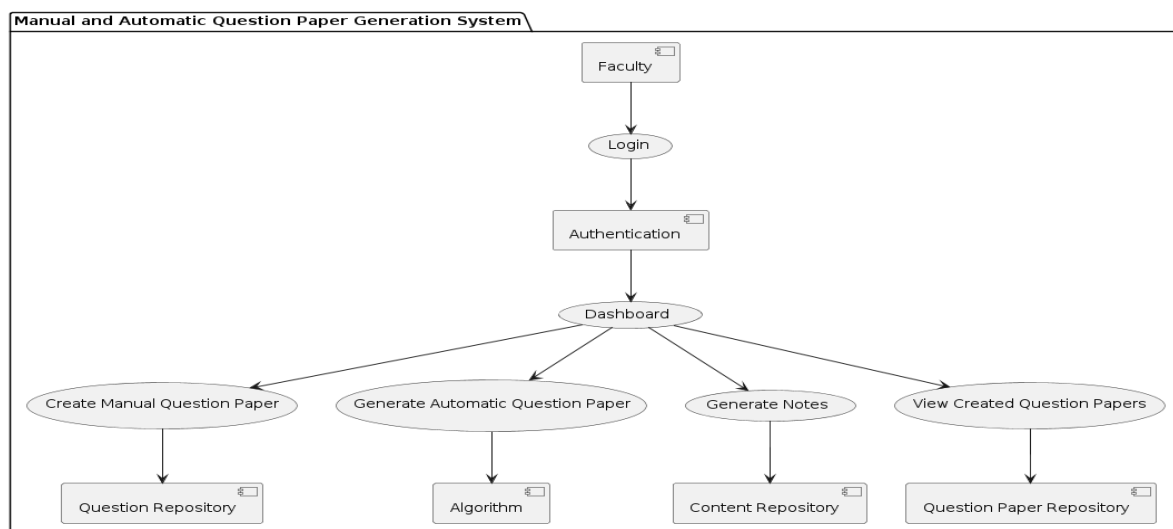


Figure 5.2 Data Flow diagram

5.1.3 Activity Diagram

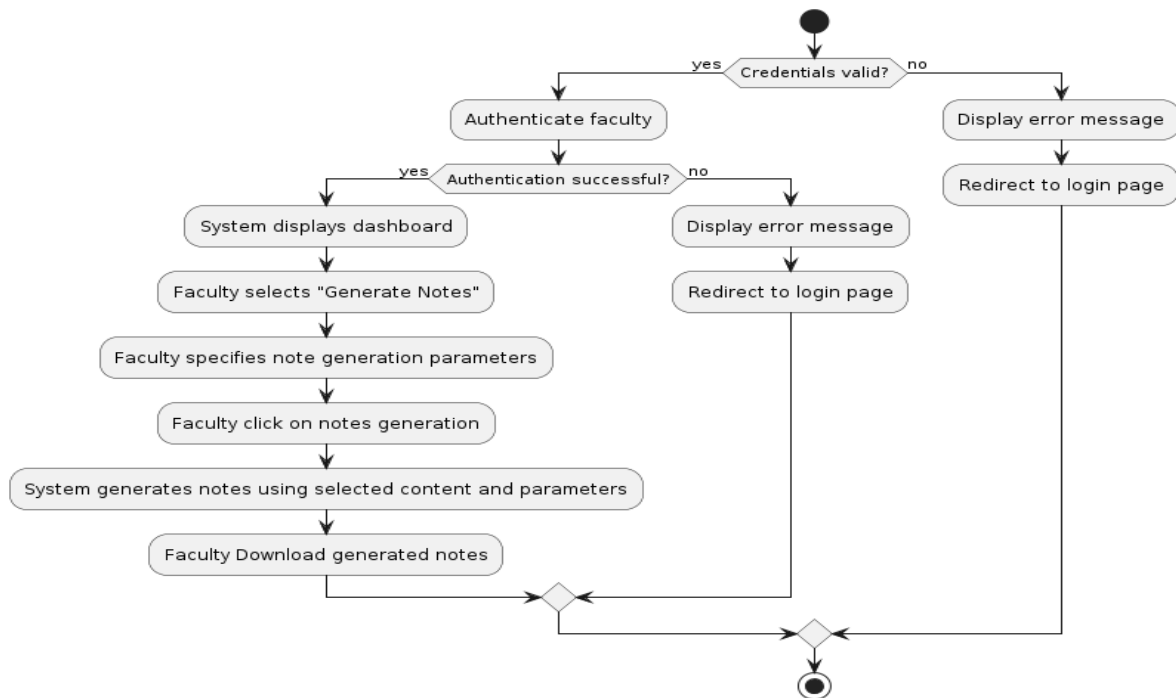


Figure 5.3 Activity of Notes Generation from Database Module Flow

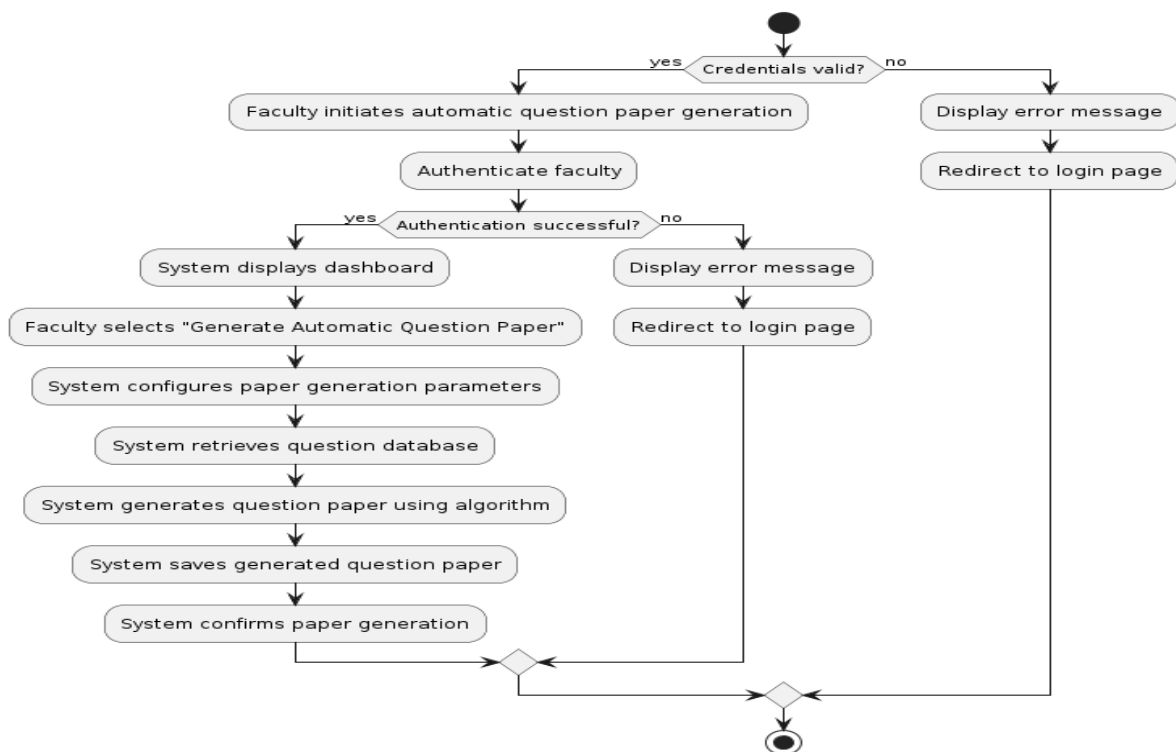


Figure 5.4 Activity Generate Question Paper from Database Module Flow

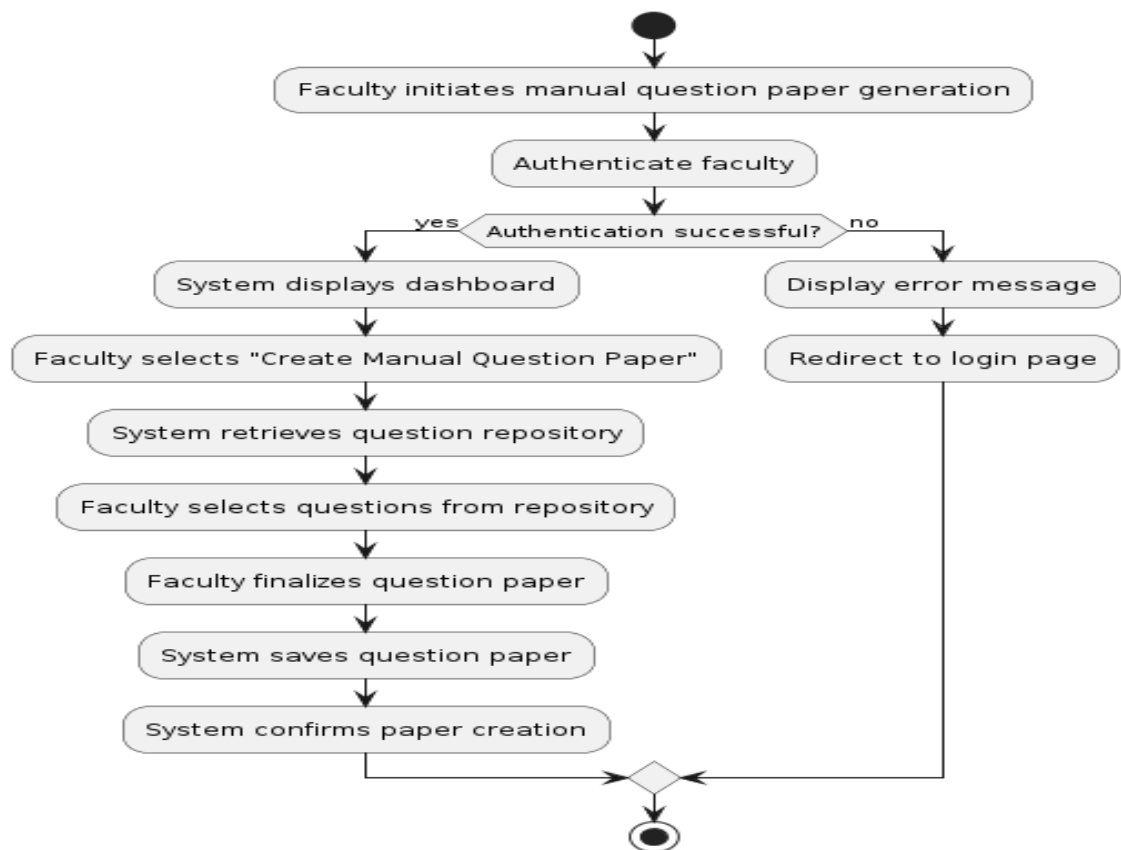


Figure 5.5 Activity of Manual Generating Question Paper Module Flow

5.1.4 Component Diagram

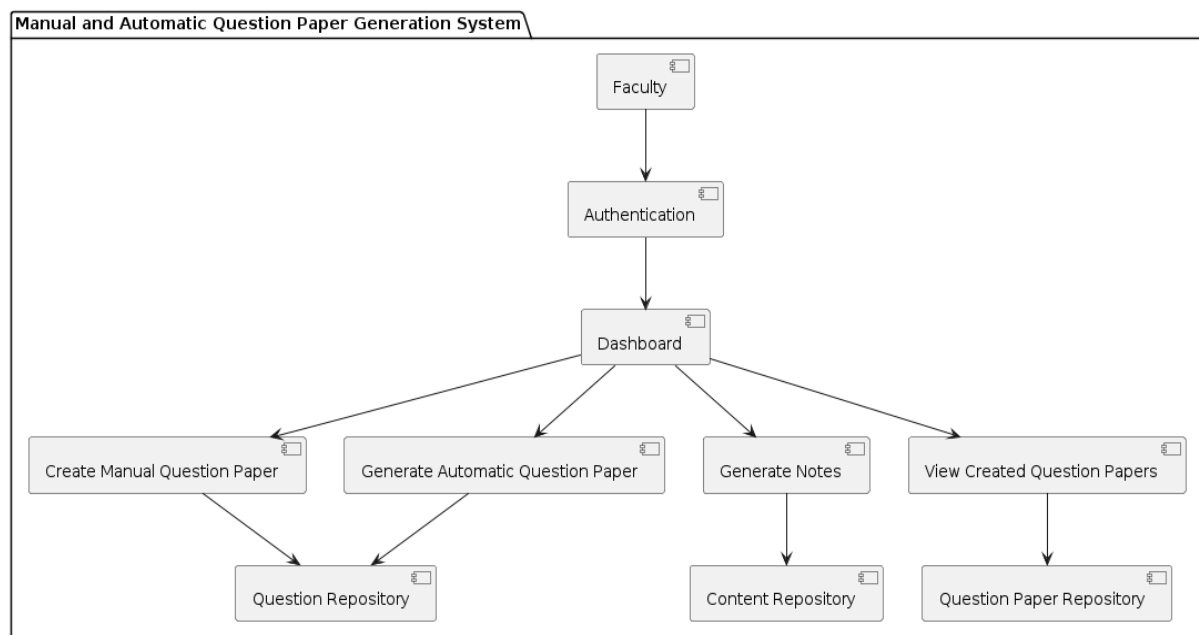


Figure 5.6 Component Diagram

CHAPTER 6

Project Planning

The Manual and Automatic Question Paper Generation System serves as a vital tool for educators and administrators, offering efficient access to question paper creation and management. Faculty members can utilize the system to swiftly generate question papers tailored to specific exams, subjects, and academic levels. The system's customization options enable educators to design question papers that align with learning objectives and assessment criteria, enhancing the overall educational experience for students.

The development plan for the system encompasses several key phases:

Requirements Gathering and Analysis: Engaging with faculty to gather insights and define project objectives. This phase involves understanding the specific needs of educators, administrators, and students regarding question paper generation and management.

System Design and Architecture: Developing a comprehensive blueprint for the system's structure and functionality. This includes selecting appropriate technologies for the frontend interface, backend infrastructure, and database management system. Additionally, designing intuitive user interfaces and efficient data processing mechanisms are crucial aspects of this phase.

Development: Implementing the frontend interface, backend logic, and database integration based on the established design specifications. This phase involves coding, testing, and refining various system components to ensure seamless functionality and user experience.

Testing and Quality Assurance: Conducting rigorous testing procedures to validate the system's performance, functionality, and security. This includes unit testing, integration testing, and user acceptance testing to identify and address any bugs or issues. Additionally, ensuring compliance with relevant quality standards and regulations is essential during this phase.

6.1 User Perspective

From the user perspective, the Question Paper Generation System offers a user-friendly platform for creating, managing, and analyzing question papers. Key features include:

- **Intuitive Interface:** The system provides an intuitive interface with easy navigation menus,

allowing users to quickly access essential features and functionalities.

- **Comprehensive Question Paper Creation:** Users can create detailed question papers, including main questions and sub-questions, with options to specify question types, marks, Bloom's taxonomy levels, and units.
- **Advanced Filters:** The system offers filtering options to customize question paper generation based on criteria such as question type, marks, and Bloom's taxonomy level.
- **Interactive Previews:** Users can preview question papers in real-time, enabling them to visualize the structure and content before finalizing.
- **Export Options:** Users can export question papers in various formats (e.g., PDF, Word) for further editing or sharing purposes.

6.2 Admin Perspective

From the admin perspective, the Question Paper Generation System provides robust tools for managing and overseeing the question paper generation process efficiently. Key features for administrators include:

- **Comprehensive Management:** Admins have full control over the system, including adding, editing, and deleting question paper templates and user accounts.
- **User Permissions:** Admins can manage user roles and permissions to control access to system functionalities and data.
- **Performance Monitoring:** Admins can monitor the usage and performance of the system, track question paper generation metrics, and identify trends or issues.
- **Custom Reports:** The system allows admins to generate custom reports and analytics based on various criteria, such as question paper usage, user activity, and system performance.
- **Data Security:** Admins can implement security measures to protect sensitive data, ensuring compliance with regulations and safeguarding user information.

System Maintenance: Admins are responsible for system updates, maintenance, and troubleshooting to ensure smooth operation and address any technical issues promptly.

CHAPTER 7

IMPLEMENTATION

7.1 Main Component

7.1.1 Automatic_Question_Paper_Generation_sub.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Automatic Question Paper Database </title>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link
href="https://fonts.googleapis.com/css2?family=Jacquard+24&family=Jersey+15+Charted&display=
swap" rel="stylesheet">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.15.4/css/all.min.css">
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f0f0f0;
            margin: 0;
            padding: 0;
        }
        .jersey-15-charted-regular {
            font-family: "Jersey 15 Charted", sans-serif;
            font-weight: 400;
            font-style: normal;
        }
        .container {
            display: flex;
            min-height: 100vh;
        }
```

```
.sidebar {  
    width: 250px;  
    background-color: white;  
    color: black;  
    padding: 20px;  
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);  
}  
.sidebar-card {  
    margin-top: 5px;  
    background-color: white;  
    border-radius: 10px;  
    padding: 20px;  
}  
.nav-link {  
    display: block;  
    padding: 10px 15px;  
    text-decoration: none;  
    color: black;  
    transition: background-color 0.3s;  
    border-radius: 5px;  
    margin-bottom: 8px;  
    font-size: large;  
    position: relative;  
}  
.nav-link:hover {  
    background-color: #e0e0e0;  
    border-bottom: 2px solid blue;  
}  
.content {  
    flex: 1;  
    padding: 20px;  
}  
.form-container {  
    max-width: 800px;  
    margin: 50px auto; /* Center the form */
```

```

    background-color: #fff;
    padding: 30px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    width: calc(100% - 60px); /* Adjusted width */
}

.form-container h2 {
    text-align: center;
    margin-bottom: 30px;
}

form {
    /* No need to specify width here */
}

label {
    display: block;
    font-weight: bold;
    margin-bottom: 5px;
    width: 100%; /* Make labels take full width */
}

input[type="text"],
input[type="number"],
select,
textarea {
    padding: 10px;
    margin-bottom: 20px;
    border: 1px solid #ccc;
    border-radius: 5px;
    width: 100%;
    box-sizing: border-box; /* Include padding and border in the element's total width and height
*/
}

input[type="date"],
input[type="time"] {
    /* Adjust padding for date and time inputs */
    padding: 8px;

```



```
}

input[type="text"],
input[type="number"],
select,
textarea {
    padding: 8px; /* Adjust padding */
    margin-bottom: 20px;
    border: none; /* Remove border */
    border-bottom: 1px solid #ccc; /* Add bottom border */
    width: calc(100% - 16px); /* Adjusted width */
    background-color: transparent; /* Transparent background */
    transition: border-bottom-color 0.3s; /* Smooth transition */
}

input[type="text"]:focus,
input[type="number"]:focus,
select:focus,
textarea:focus {
    border-bottom-color: #007bff; /* Change border color on focus */
    outline: none; /* Remove outline */
}

input{
    font-size: large;
    color:gray;
}

.btn-container {
    text-align: center;
    width: 100%; /* Ensure buttons take full width */
}

button {
    padding: 10px 20px;
    background-color: #007bff;
    color: #fff;
```

```
border: none;
border-radius: 4px;
cursor: pointer;
transition: background-color 0.3s ease;
width: 15%; /* Adjusted width */
margin: 0 5px; /* Add margin between buttons */
}

button:hover {
    background-color: #75ed6f;
}

.section {
    margin-bottom: 20px;
    border: 1px solid #ccc;
    border-radius: 8px; /* Adjust border radius */
    padding: 20px;
    width: 80%; /* Ensure sections take full width */
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); /* Add shadow */
}

.questions-container {
    margin-top: 20px;
}

.questions-container ul {
    list-style-type: none;
    padding: 0;
}

.questions-container ul li {
    margin-bottom: 10px;
}

.questions-container p {
```

```
margin-top: 10px;
color: #888;
font-style: italic;
}
.navb {
height: 50px;
background-color: white;
display: flex;
justify-content: space-between; /* Align items to the start and end of the navbar */
align-items: center;
box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
position: fixed;
top: 0;
width: 100%;
padding-left: 2%;
height: 10%;
z-index: 1000;
color: black;
}
.nav-icon{
color: black;
padding: 5px;
border-radius: 5px;
font-weight: bold;
}
.nav-list{
list-style-type: none;
}
.nav-icon:hover{
padding: 5px;
border-radius: 5px;
background-color: #e0e0e0;
border-bottom: 2px solid blue;
list-style-type: none;
}
```

```
.jacquard-24-regular {
font-family: "Jacquard 24", system-ui;
font-weight: 400;
font-style: normal;
}
.pacifico-regular {
font-family: "Pacifico", cursive;
font-weight: 400;
font-style: normal;
}
.active{
background-color: #e0e0e0;
border-bottom: 2px solid blue;
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<nav class="navb">
```

```
<h2 class="pacifico-regular">Manual and Automatic Question Paper Generation System</h2>
```

```
<!-- Add font-family style here -->
```

```
<ul>
```

```
<li class="nav-list"><a class="nav-icon" href="/profile" style="margin-right: 50px;"><i
class="fas fa-user"></i> Profile</a></li>
```

```
</ul>
```

```
</nav>
```

```
<div style="margin-top: 4%;">
```

```
<div class="container">
```

```
<div class="sidebar">
```

```
<div class="logo" style="margin-left: 20%;">
```

```
<a href="/dashboard" ></a>
```

```
</div>
```

```
<div class="sidebar-card">
```

```
<ul style="list-style-type:none; padding: 0;">
```

```
<li><a href="/dashboard" class="nav-link"><i class="fas fa-tachometer-alt"></i>
```

Dashboard

<i class="fas fa-file" style="margin-right: 10px;"></i>Manual Question Paper Creation

<i class="fas fa-database" style="margin-right: 10px;"></i>Automatic Question Paper Generation

<i class="fas fa-file-alt" style="margin-right: 10px;"></i>

Question Papers

<i class="fas fa-book" style="margin-right: 10px;"></i>

Notes Generation

<i class="fas fa-sign-out-alt"></i> Log out

</div>

</div>

<div class="content">

<form class="form-container" action="" method="POST" id="questionForm" style="margin-top: 10px;"> {% csrf_token %}

<h2 style="text-align: center;">Automatic Question Paper Generation</h2>

<label for="dept">Select Department:</label>

<select id="dept" name="dept">

<option value="Select">Select</option>

<option value="Department of INFORMATION TECHNOLOGY">Department of INFORMATION TECHNOLOGY</option>

<option value="Department of ELECTRONICS & COMMUNICATION ENGINEERING">Department of ELECTRONICS & COMMUNICATION ENGINEERING</option>

<option value="Department of ELECTRONICS & INSTRUMENTATION ENGINEERING">Department of ELECTRONICS & INSTRUMENTATION ENGINEERING</option>

<option value="Department of ELECTRICAL & ELECTRONICS ENGINEERING">Department of ELECTRICAL & ELECTRONICS ENGINEERING</option>

<option value="Department of COMPUTER SCIENCE & ENGINEERING">Department of COMPUTER SCIENCE & ENGINEERING</option>

```

        <option value="Department of MECHANICAL ENGINEERING">Department of
MECHANICAL ENGINEERING</option>
        <option value="Department of DATA SCIENCE">Department of DATA
SCIENCE</option>
        <option value="Department of CIVIL ENGINEERING">Department of CIVIL
ENGINEERING</option>
        <option value="Department of CYBER SECURITY">Department of CYBER
SECURITY</option>
        <option value="Department of ARTIFICIAL INTELLIGENCE & MACHINE
LEARNING">Department of ARTIFICIAL INTELLIGENCE & MACHINE LEARNING</option>
    </select>
    <label for="sem">Select Semester:</label>
    <select id="sem" name="sem">
        <option value="Select">Select</option>
        <option value="I-sem">I</option>
        <option value="II-sem">II</option>
        <option value="III-sem">III</option>
        <option value="IV-sem">IV</option>
        <option value="V-sem">V</option>
        <option value="VI-sem">VI</option>
        <option value="VII-sem">VII</option>
        <option value="VIII-sem">VIII</option>
    </select>

    <label for="name of exam">Name of the Exam:</label>
    <select id="exam" name="exam">
        <option value="Select">Select</option>
        <option value="Assignment-1">Assignment-1</option>
        <option value="Assignment-2">Assignment-2</option>
        <option value="Mid-term Examination-1">Mid-term Examination-1</option>
        <option value="Mid-term Examination-2">Mid-term Examination-2</option>
        <option value="Supplementary">Supplementary</option>
    </select>

    <label for="subject">Name of the Subject:</label>

```

```

<input type="text" id="subject" name="subject" required>

<!--<label for="paper-name">Question Paper Name:</label>
<input type="text" id="paper-name" name="paper-name" value="default" required>-->

<label for="paper-code">Question Paper Code:</label>
<input type="text" id="paper-code" name="paper-code" required>

<label for="exam-date">Date of the Exam:</label>
<input type="date" id="exam-date" name="exam-date" required>

<label for="exam-time">Duration of the Exam:</label>
<input type="time" id="exam-time" name="exam-time" required>
<label for="num-sections">Number of Sections:</label>
<input type="number" id="num-sections" name="num-sections" min="1" required>
<div id="sections-container">
  <!-- Sections will be added dynamically using JavaScript -->
</div>
<!-- Buttons -->
<div class="btn-container">
  <button type="submit">Submit</button>
</div>
</form>
</div>
</div>
</div>
<script>
document.getElementById("num-sections").addEventListener("change", function() {
  const numSections = this.value;
  const sectionsContainer = document.getElementById("sections-container");
  sectionsContainer.innerHTML = ""; // Clear previous sections
  // Initialize global counter for main questions
  let mainQuestionCounter = 0;
  for (let i = 1; i <= numSections; i++) {
    const section = document.createElement("div");

```

```

section.classList.add("section");
const sectionLabel = document.createElement("label");
sectionLabel.textContent = Section ${i};;
const sectionNameInput = document.createElement("input");
sectionNameInput.type = "text";
sectionNameInput.name = Section ${i};
sectionNameInput.placeholder = "Enter section Name ";
sectionNameInput.required = true;
section.appendChild(document.createElement("br"));
section.appendChild(document.createElement("br"));
section.appendChild(sectionLabel);
section.appendChild(sectionNameInput);
const numMainQuestionsInput = document.createElement("input");
numMainQuestionsInput.type = "number";
numMainQuestionsInput.name = num-main-questions-${i};
numMainQuestionsInput.placeholder = "Number of main questions";
numMainQuestionsInput.required = true;
section.appendChild(document.createElement("br"));
section.appendChild(numMainQuestionsInput);
const mainQuestionsContainer = document.createElement("div");
numMainQuestionsInput.addEventListener("change", function() {
    const numMainQuestions = parseInt(this.value);
    const previousMainQuestions = mainQuestionsContainer.querySelectorAll(".main-
question").length;
    const difference = numMainQuestions - previousMainQuestions;
    if (difference > 0) {
        for (let j = 1; j <= difference; j++) {
            mainQuestionCounter++;
            // Alphabet counter for sub-questions
            let subQuestionAlphabet = 'a';
            const mainQuestionDiv = document.createElement("div");
            mainQuestionDiv.classList.add("main-question");
            const mainQuestionLabel = document.createElement("label");
            mainQuestionLabel.textContent = Main Question ${mainQuestionCounter};;
            const numSubQuestionsInput = document.createElement("input");

```



```

numSubQuestionsInput.type = "number";
numSubQuestionsInput.name = num-sub-questions-${i}-${j};
numSubQuestionsInput.placeholder = "Number of sub-questions";
numSubQuestionsInput.required = true;
mainQuestionDiv.appendChild(document.createElement("br"));
mainQuestionDiv.appendChild(mainQuestionLabel);
mainQuestionDiv.appendChild(numSubQuestionsInput);
const subQuestionsContainer = document.createElement("div");

numSubQuestionsInput.addEventListener("change", function() {
  const numSubQuestions = parseInt(this.value);
  subQuestionsContainer.innerHTML = ""; // Clear previous sub-questions
  for (let k = 1; k <= numSubQuestions; k++) {
    // Question number for the preview (Main Question Number + Alphabet)
    const subQuestionNumber = ${subQuestionAlphabet};
    // Increment alphabet counter for next sub-question
    subQuestionAlphabet =
String.fromCharCode(subQuestionAlphabet.charCodeAt(0) + 1);
    const subQuestionDiv = document.createElement("div");
    subQuestionDiv.classList.add("sub-question");
    const subQuestionLabel = document.createElement("label");
    subQuestionLabel.textContent = Sub-Question ${subQuestionNumber};;
    const questionTypeSelect = document.createElement("select");
    questionTypeSelect.name = question-type-${i}-${j}-${k};
    questionTypeSelect.innerHTML = `
      <option value="Select">Select</option>
      <option value="Descriptive">Descriptive</option>
      <option value="MCQ">MCQ</option>
      <option value="FillInTheBlank">Fill in the Blank</option>
      <option value="TrueOrFalse">True or False</option>
    `;
    const bloomTaxonomySelect = document.createElement("select");
    bloomTaxonomySelect.name = bloom-level-${i}-${j}-${k};
    const levels = ["L1", "L2", "L3", "L4", "L5", "L6"];
    levels.forEach(level => {

```

```

        const option = document.createElement("option");
        option.value = level;
        option.textContent = level;
        bloomTaxonomySelect.appendChild(option);
    });
    const unitInput = document.createElement("select");
    unitInput.name = unit-${i}-${j}-${k};
    unitInput.innerHTML = `
        <option value="Select">Select</option>
        <option value="co1">CO1</option>
        <option value="co2">CO2</option>
        <option value="co3">CO3</option>
        <option value="co4">CO4</option>
    `;

    const marksInput = document.createElement("input");
    marksInput.type = "number";
    marksInput.name = marks-${i}-${j}-${k};
    marksInput.placeholder = "Marks";
    marksInput.required = true;

    subQuestionDiv.appendChild(document.createElement("br"));
    subQuestionDiv.appendChild(subQuestionLabel);
    subQuestionDiv.appendChild(document.createElement("br"));
    subQuestionDiv.appendChild(questionTypeSelect);
    subQuestionDiv.appendChild(document.createElement("br"));
    subQuestionDiv.appendChild(unitInput);
    subQuestionDiv.appendChild(document.createElement("br"));
    subQuestionDiv.appendChild(bloomTaxonomySelect);
    subQuestionDiv.appendChild(document.createElement("br"));
    subQuestionDiv.appendChild(marksInput);
    subQuestionsContainer.appendChild(subQuestionDiv);
}

});
mainQuestionDiv.appendChild(subQuestionsContainer);
mainQuestionsContainer.appendChild(mainQuestionDiv);
}

```

```

    }
    else if (difference < 0) {
        for (let j = 0; j > difference; j--) {
            mainQuestionCounter--;
            mainQuestionsContainer.removeChild(mainQuestionsContainer.lastChild);
        }
    }
    });
    section.appendChild(mainQuestionsContainer);
    sectionsContainer.appendChild(section);
}
});
</script>
</body>
</html>

```

7.1.2 notes_generation.html

```

{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Notes Genereation</title>
    <!-- Bootstrap CSS -->
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
rel="stylesheet">
    <!-- FontAwesome CSS -->
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/font-
awesome/5.15.4/css/all.min.css">
    <style>
        /* Add your custom CSS styles here */
        .pacifico-regular {
            font-family: "Pacifico", cursive;
            font-weight: 400;
            font-style: normal;

```

```
font-size: 24px;
}
.maincontainer {
  display: flex;

}
.sidebar {
  width: 250px;
  background-color: white;
  color: black;
  padding: 20px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

.sidebar-card {
  margin-top: 5px;
  background-color: white;
  border-radius: 10px;
  padding: 10px;
}

.nav-link {
  display: block;
  padding: 10px 15px;
  text-decoration: none;
  color: black;
  transition: background-color 0.3s;
  border-radius: 5px;
  margin-bottom: 8px;
  font-size: large;
  position: relative;
}

.nav-link:hover {
  background-color: #e0e0e0;
  color: black;
  border-bottom: 2px solid blue;
```

```
}  
.navb {  
    height: 50px;  
    background-color: white;  
    display: flex;  
    justify-content: space-between; /* Align items to the start and end of the navbar */  
    align-items: center;  
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);  
    position: fixed;  
    top: 0;  
    width: 100%;  
    padding-left: 2%;  
    height: 10%;  
    z-index: 1000;  
    color: black;  
}  
.nav-icon{  
    color: black;  
    padding: 5px;  
    border-radius: 5px;  
    font-weight: bold;  
}  
.nav-list{  
    list-style-type: none;  
    margin-top: 15px;  
}  
.nav-icon:hover{  
    padding: 5px;  
    text-decoration: none;  
    border-radius: 5px;  
    background-color: #e0e0e0;  
    border-bottom: 2px solid blue;  
    list-style-type: none;  
}  
.card {
```

```

margin-bottom: 20px;
border: 1px solid #ccc;
border-radius: 5px;
box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
.card-body {
padding: 20px;
}
.active{
background-color: #e0e0e0;
border-bottom: 2px solid blue;
}
</style>
</head>
<body>
<nav class="navb">
<h2 class="pacifico-regular">Manual and Automatic Question Paper Generation System</h2>
<!-- Add font-family style here -->
<ul>
<li class="nav-list"><a class="nav-icon" href="/profile" style="margin-right: 50px;"><i
class="fas fa-user"></i> Profile</a></li>
</ul>
</nav>
<div class="maincontainer" style="margin-top: 4%;">
<div class="sidebar">
<div class="logo" style="margin-left: 20%;">
<a href="/dashboard" ></a>
</div>
<div class="sidebar-card">
<ul style="list-style-type:none; padding: 0;">
<li><a href="/dashboard" class="nav-link"><i class="fas fa-tachometer-alt"></i>
Dashboard</a></li>
<li><a href="/create" class="nav-link"><i style="margin-right: 10px;" class="fas fa-
file"></i>Manual Question Paper Creation</a></li>
<li><a href="/automatic" class="nav-link"><i style="margin-right: 10px;" class="fas fa-

```

```

database"></i> Automatic Question Paper Generation</a></li>
    <li><a href="/question_papers" class="nav-link" ><i class="fas fa-file-alt"
style="margin-right: 10px;"></i>
        Question Papers</a></li>
    <li><a href="/notesgeneration" class="nav-link active" ><i class="fas fa-book"
style="margin-right: 10px;"></i>
        Notes Generation</a></li>
    <li><a href="/logout" class="nav-link"><i class="fas fa-sign-out-alt"></i> Log
out</a></li>
</ul>
</div>
</div>
<div class="container">
    <h1 class="mt-3 mb-3">Notes Generation</h1>
    <!-- Search Bar -->
    <form method="GET" action="" class="mb-5">
        <div class="card d-flex align-items-center">
            <div class="w-75 m-5">
                <select class="form-control" name="subject">
                    <option value="All Subjects">All Subjects</option>
                    { % for subject in subjects % }
                    <option value="{{ subject.Name }}">{{ subject.Name }}</option>
                    { % endfor % }
                </select>
            </div>
            <div class="w-75 m-5">
                <select class="form-control" name="type">
                    <option value="">All Types</option>
                    <option value="MCQ">Multiple Choice</option>
                    <option value="TrueOrFalse">True or False</option>
                    <option value="FillInTheBlank">Fill in the Blank</option>
                    <option value="Descriptive">Descriptive</option>
                </select>
            </div>
            <div class="w-75 m-5">

```

```

        <select class="form-control" name="unit">
            <option value="">All Units</option>
            <option value="co1">CO1</option>
            <option value="co2">CO2</option>
            <option value="co3">CO3</option>
            <option value="co4">CO4</option>
        </select>
    </div>

    <div class="input-group-append m-2 mb-5">
        <button type="submit" class="btn btn-primary">Generate Notes</button>
    </div>
</div>
</form>
</div>
</div>
<!-- Bootstrap JS (optional) -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>

```

7.1.3 models.py

```

from django.db import models

# Create your models here.

from django.contrib.auth.models import User

class Users(models.Model):
    UserID = models.AutoField(primary_key=True)
    Username = models.CharField(max_length=255)
    Password = models.CharField(max_length=255)
    Role = models.CharField(max_length=50)

class Categories(models.Model):
    CategoryID = models.AutoField(primary_key=True)
    Name = models.CharField(max_length=255)

```



```

class Subjects(models.Model):
    SubjectID = models.AutoField(primary_key=True)
    Name = models.CharField(max_length=255)

class BloomsTaxonomy(models.Model):
    TaxonomyID = models.CharField(max_length=2, primary_key=True)
    LevelName = models.CharField(max_length=50, unique=True)

class QuestionPapers(models.Model):
    QuestionPaperID = models.AutoField(primary_key=True)
    PaperName = models.CharField(max_length=255)
    PaperCode = models.CharField(max_length=255) # Add this field
    ExamDate = models.DateField() # Add this field
    ExamTime = models.TimeField() # Add this field
    Department = models.CharField(max_length=255, null=False) # Add this field
    Semester = models.CharField(max_length=255, null=False) # Add this field
    TotalMarks = models.IntegerField()
    Faculty = models.ForeignKey(User, on_delete=models.CASCADE)
    Subject = models.ForeignKey(Subjects, on_delete=models.CASCADE)
    Category = models.ForeignKey(Categories, on_delete=models.CASCADE)
    CreatedDate = models.DateTimeField(auto_now_add=True)

    def _str_(self):
        return self.PaperName

class PaperSections(models.Model):
    PaperID = models.ForeignKey(QuestionPapers, on_delete=models.CASCADE)
    SectionNumber = models.IntegerField()
    SectionName = models.CharField(max_length=255)
    NumMainQuestions = models.IntegerField()
    TotalMarksInSection = models.IntegerField()

    class Meta:
        constraints = [
            models.UniqueConstraint(fields=['PaperID', 'SectionNumber'], name='unique_paper_section')
        ]

    def _str_(self):
        return f"Section {self.SectionNumber} - {self.SectionName}"

class MainQuestions(models.Model):
    PaperID = models.ForeignKey(QuestionPapers, on_delete=models.CASCADE)

```

```

MainQuestionNumber = models.IntegerField()
MainQuestionID = models.AutoField(primary_key=True)
Section = models.ForeignKey(PaperSections, on_delete=models.CASCADE)
NumSubQuestions = models.IntegerField()
def _str_(self):
    return f"Main Question ID: {self.MainQuestionID}, Section: {self.Section.SectionName},
NumSubQuestions: {self.NumSubQuestions}"
class Questions(models.Model):
    QuestionID = models.AutoField(primary_key=True)
    QuestionText = models.TextField()
    Answer = models.TextField(null=True, blank=True)
    OptionA = models.CharField(max_length=255, null=True, blank=True)
    OptionB = models.CharField(max_length=255, null=True, blank=True)
    OptionC = models.CharField(max_length=255, null=True, blank=True)
    OptionD = models.CharField(max_length=255, null=True, blank=True)
    CorrectOption = models.CharField(max_length=255, null=True, blank=True)
    Unit = models.CharField(max_length=7, null=True, blank=True)
    SubjectID = models.ForeignKey(Subjects, on_delete=models.CASCADE)
    FacultyID = models.ForeignKey(User, on_delete=models.CASCADE)
    Marks = models.IntegerField()
    Type = models.CharField(max_length=50)
    BloomTaxonomyID = models.CharField(max_length=2)
class Meta:
    unique_together = ('QuestionText', 'Type')
def _str_(self):
    return f"Question ID: {self.QuestionID}, Text: {self.QuestionText}, Type: {self.Type}"
class QuestionImage(models.Model):
    Question = models.ForeignKey(Questions, on_delete=models.CASCADE,
related_name='question_images')
    Image = models.ImageField(upload_to='question_images/')

class AnswerImage(models.Model):
    Question = models.ForeignKey(Questions, on_delete=models.CASCADE,
related_name='answer_images')
    Image = models.ImageField(upload_to='answer_images/')

```

```

class QuestionPaperQuestions(models.Model):
    QuestionPaperID = models.ForeignKey(QuestionPapers, on_delete=models.CASCADE)
    QuestionID = models.ForeignKey(Questions, on_delete=models.CASCADE)
    SectionNumber = models.IntegerField()
    MainQuestionsID = models.ForeignKey(MainQuestions, on_delete=models.CASCADE)
    class Meta:
        unique_together = ('QuestionPaperID', 'QuestionID')
    def __str__(self):
        return f"QuestionPaperID: {self.QuestionPaperID}, QuestionID: {self.QuestionID}"
class UserProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    profile_image = models.ImageField(upload_to='profile_images', blank=True, null=True)

```

7.1.4 views.py

```

# views.py
from django.shortcuts import render
from .models import Users, QuestionPapers, PaperSections,
Questions, QuestionPaperQuestions, Subjects, Categories, QuestionImage, MainQuestions, AnswerImage
from django.db.models import Sum
from django.shortcuts import redirect
from django.utils import timezone
from django.contrib.auth.models import User
from datetime import timedelta
import random
from django.db.models import Q
def question_paper_preview(request, question_paper_id):
    if request.user.is_authenticated:
        question_paper = QuestionPapers.objects.get(pk=question_paper_id)
        sections = PaperSections.objects.filter(PaperID=question_paper_id)
        qpq=QuestionPaperQuestions.objects.filter(QuestionPaperID=question_paper_id)
        # Fetch main questions and their sub-questions for each section
        main_questions = MainQuestions.objects.filter(PaperID=question_paper_id)
        question_images = QuestionImage.objects.all()
        print(main_questions)

```

```

for mainquestion in main_questions:
    print(mainquestion)
for i in qpq:
    print(i)
    print(i.QuestionID)
questions = []
for paper_question in qpq:
    questions.append(paper_question.QuestionID)
print(question_paper)
print(questions)
context = {
    'question_paper': question_paper,
    'sections': sections,
    'questions': questions,
    'qpq':qpq,
    'mainquestions': main_questions,
    'question_images':question_images,
}
return render(request, 'html/question_paper_preview.html', context)
else:
    return redirect('/')
def create_question_paper(request):
    if request.user.is_authenticated:
        if request.method == 'POST':
            # Extract form data
            category_name = request.POST.get('exam')
            subject_name = request.POST.get('subject')
            paper_name = "Default"
            paper_code = request.POST.get('paper-code')
            exam_date = request.POST.get('exam-date')
            exam_time = request.POST.get('exam-time')
            department = request.POST.get('dept')
            semester = request.POST.get('sem')
            num_sections = int(request.POST.get('num-sections'))
            # Get or create the Category instance

```

```

category, _ = Categories.objects.get_or_create(Name=category_name)
# Get or create the Subject instance
subject, _ = Subjects.objects.get_or_create(Name=subject_name)
# Get the current authenticated user
faculty = request.user
# Create a new QuestionPaper instance with creation time
question_paper = QuestionPapers.objects.create(
    PaperName=paper_name, TotalMarks=0, Category=category, Subject=subject,
    PaperCode=paper_code, ExamDate=exam_date, ExamTime=exam_time,
    Department=department, Semester=semester, Faculty=faculty,
    CreatedDate=timezone.now() # Add creation time
)
# Iterate over sections
for i in range(1, num_sections + 1):
    section_name = request.POST.get(f'section-name-{i}')
    num_main_questions = int(request.POST.get(f'num-main-questions-{i}'))
    total_section_marks = 0 # Initialize total marks for the section
    # Create a new PaperSections instance for each section
    section = PaperSections.objects.create(
        PaperID=question_paper, SectionNumber=i, SectionName=section_name,
        NumMainQuestions=num_main_questions, TotalMarksInSection=0
    )
    # Iterate over main questions in the section
    for j in range(1, num_main_questions + 1):
        num_sub_questions = int(request.POST.get(f'num-sub-questions-{i}-{j}'))
        total_main_question_marks = 0 # Initialize total marks for the main question
        # Create a new MainQuestions instance for each main question
        main_question = MainQuestions.objects.create(
            PaperID=question_paper, Section=section, MainQuestionNumber=j,
            NumSubQuestions=num_sub_questions
        )
        # Iterate over sub-questions in the main question
        for k in range(1, num_sub_questions + 1):
            question_text = request.POST.get(f'question-{i}-{j}-{k}')
            marks = int(request.POST.get(f'marks-{i}-{j}-{k}'))

```

```

question_type = request.POST.get(f'question-type-{i}-{j}-{k}')
bloom_level = request.POST.get(f'bloom-level-{i}-{j}-{k}')
unit = request.POST.get(f'unit-{i}-{j}-{k}')
# Additional processing based on question type
answer = None
option_a, option_b, option_c, option_d = None, None, None, None
correct_option = None
# Get the image(s) for the question
question_images = request.FILES.getlist(f'image-upload-{i}-{j}-{k}')
answer_images = request.FILES.getlist(f'answer-image-upload-{i}-{j}-{k}')
if question_type == 'Descriptive':
    # Get answer for descriptive question
    answer = request.POST.get(f'answer-{i}-{j}-{k}')
elif question_type == 'MCQ':
    # Get options and correct option for MCQ
    option_a = request.POST.get(f'option-text-{i}-{j}-1')
    option_b = request.POST.get(f'option-text-{i}-{j}-2')
    option_c = request.POST.get(f'option-text-{i}-{j}-3')
    option_d = request.POST.get(f'option-text-{i}-{j}-4')
    correct_option = request.POST.get(f'option-{i}-{j}')
elif question_type == 'FillInTheBlank':
    # Get answer for fill-in-the-blank question
    answer = request.POST.get(f'answer-{i}-{j}-{k}')
elif question_type == 'TrueOrFalse':
    # Treat true or false as options
    option_a = 'True'
    option_b = 'False'
    correct_option = request.POST.get(f'truefalse-{i}-{j}-{k}') # Get selected option
# Check if the question already exists
existing_question = Questions.objects.filter(QuestionText=question_text,
Type=question_type).first()
if existing_question:
    question = existing_question
else:
    # Create a new Questions instance for each question

```

```

        question = Questions.objects.create(
            QuestionText=question_text, Marks=marks,
Type=question_type, Answer=answer,
            SubjectID=subject, FacultyID=faculty,
            OptionA=option_a, OptionB=option_b, OptionC=option_c, OptionD=option_d,
            CorrectOption=correct_option, Unit=unit, BloomTaxonomyID=bloom_level
        )
# Associate question images with the question
for image in question_images:
    question_image = QuestionImage.objects.create(
        Question=question, Image=image
    )
# Associate answer images with the question
for image in answer_images:
    answer_image = AnswerImage.objects.create(
        Question=question, Image=image
    )
# Add the question to the PaperQuestions table
QuestionPaperQuestions.objects.create(
    QuestionPaperID=question_paper, QuestionID=question, SectionNumber=i,
MainQuestionsID=main_question
)
# Add marks of current question to the total marks of the main question
total_main_question_marks += marks
# Add the total marks of the main question to the total marks of the section
total_section_marks += total_main_question_marks
# Update the total marks for the section
section.TotalMarksInSection = total_section_marks
section.save()
# Calculate total marks for the question paper
total_marks =
PaperSections.objects.filter(PaperID=question_paper).aggregate(total_marks=Sum('TotalMarksInSec
tion'))['total_marks']
question_paper.TotalMarks = total_marks or 0
question_paper.save()

```

```

        return redirect('preview', question_paper_id=question_paper.pk)
# If the request method is not POST, render the form template
return render(request, 'html/create-question-paper-sub.html')
else:
    return redirect('/')
def automatic_view(request):
    if request.user.is_authenticated:
        if request.method == 'POST':
            # Extract form data
            category_name = request.POST.get('exam')
            subject_name = request.POST.get('subject')
            paper_name = "Default"
            paper_code = request.POST.get('paper-code')
            exam_date = request.POST.get('exam-date')
            exam_time = request.POST.get('exam-time')
            department = request.POST.get('dept')
            semester = request.POST.get('sem')
            num_sections = int(request.POST.get('num-sections'))
            # Get or create the Category instance
            category, _ = Categories.objects.get_or_create(Name=category_name)
            # Get or create the Subject instance
            subject, _ = Subjects.objects.get_or_create(Name=subject_name)
            # Get the current authenticated user
            faculty = request.user
            # Create a new QuestionPaper instance with creation time
            question_paper = QuestionPapers.objects.create(
                PaperName=paper_name, TotalMarks=0, Category=category, Subject=subject,
                PaperCode=paper_code, ExamDate=exam_date, ExamTime=exam_time,
                Department=department, Semester=semester, Faculty=faculty,
                CreatedDate=timezone.now() # Add creation time
            )
            # Iterate over sections
            for i in range(1, num_sections + 1):
                section_name = request.POST.get(f'Section {i}')
                num_main_questions = int(request.POST.get(f'num-main-questions-{i}'))

```



```

total_section_marks = 0 # Initialize total marks for the section
# Create a new PaperSections instance for each section
section = PaperSections.objects.create(
    PaperID=question_paper, SectionNumber=i, SectionName=section_name,
    NumMainQuestions=num_main_questions, TotalMarksInSection=0
)
# Iterate over main questions in the section
for j in range(1, num_main_questions + 1):
    num_sub_questions = int(request.POST.get(f'num-sub-questions-{i}-{j}'))
    total_main_question_marks = 0 # Initialize total marks for the main question
    # Create a new MainQuestions instance for each main question
    main_question = MainQuestions.objects.create(
        PaperID=question_paper, Section=section, MainQuestionNumber=j,
        NumSubQuestions=num_sub_questions
    )
    # Iterate over sub-questions in the main question
    for k in range(1, num_sub_questions + 1):
        # Retrieve BloomTaxonomyID, Unit, Type, and Marks for each sub-question
        bloom_level = request.POST.get(f'bloom-level-{i}-{j}-{k}')
        unit = request.POST.get(f'unit-{i}-{j}-{k}')
        question_type = request.POST.get(f'question-type-{i}-{j}-{k}')
        marks = int(request.POST.get(f'marks-{i}-{j}-{k}'))
        # Initialize a set to keep track of selected question IDs to avoid repetition
        selected_question_ids = set()
        # Retrieve questions based on criteria until a unique question is found
        while True:
            # Get a random question based on the specified criteria
            questions = Questions.objects.filter(
                SubjectID=subject, BloomTaxonomyID=bloom_level, Unit=unit,
                Type=question_type, Marks=marks
            ).exclude(QuestionID__in=selected_question_ids)
            # If there are no more available questions that meet the criteria, break the loop
            if not questions.exists():
                break
            # Select a random question from the filtered queryset

```

```

        question = random.choice(questions)
        # If the question ID is not in the set of selected question IDs, break the loop
        if question.QuestionID not in selected_question_ids:
            selected_question_ids.add(question.QuestionID)
            break
        # Add the selected question to the PaperQuestions table
        QuestionPaperQuestions.objects.create(
            QuestionPaperID=question_paper, QuestionID=question, SectionNumber=i,
MainQuestionsID=main_question
        )
        total_main_question_marks += marks
        # Add the total marks of the main question to the total marks of the section
        total_section_marks += total_main_question_marks
        # Update the total marks for the section
        section.TotalMarksInSection = total_section_marks
        section.save()
        # Calculate total marks for the question paper
        total_marks =
PaperSections.objects.filter(PaperID=question_paper).aggregate(total_marks=Sum('TotalMarksInSec
tion'))['total_marks']
        if category_name in ['Assignment-1', 'Assignment-2']:
            question_paper.TotalMarks = 10
        elif category_name in ['Mid-term Examination-1', 'Mid-term Examination-2']:
            question_paper.TotalMarks = 35
        elif category_name in ['Supplementary','Sem End Examination']:
            question_paper.TotalMarks = 70
        else:
            question_paper.TotalMarks = 0 # Default value
        #question_paper.TotalMarks = total_marks or 0
        question_paper.save()
        return redirect('preview', question_paper_id=question_paper.pk)
        # If the request method is not POST, render the form template
        return render(request, 'html/Automatic_Question_Paper_Generation_sub.html')
    else:
        return redirect('/')

```

```

def question_papers(request):
    if request.user.is_authenticated:
        # Fetch question papers created by the currently logged-in user
        subjects = Subjects.objects.all()
        user = request.user
        question_papers = QuestionPapers.objects.filter(Faculty=user)
        # Adjusting CreatedDate by subtracting 5.5 hours
        search_query = request.GET.get('search')
        if search_query:
            question_papers = question_papers.filter(Subject_Nameicontains=search_query)
        for paper in question_papers:
            paper.CreatedDate += timedelta(hours=5, minutes=30) # Subtracting 5.5 hours
        context = {'question_papers': question_papers, 'subjects': subjects}
        return render(request, 'html/question_papers.html', context)
    else:
        return redirect('/')

def notesgeneration(request):
    if request.user.is_authenticated:
        subject_name = request.GET.get('subject')
        question_type = request.GET.get('type')
        unit = request.GET.get('unit')
        if (question_type or subject_name or unit) and subject_name != "All Subjects" or
question_type=="":
            query = Q()
            if subject_name and subject_name != "All Subjects":
                # Assuming SubjectID is the foreign key to the Subject model
                query &= Q(SubjectID__Name=subject_name)
            if question_type:
                query &= Q(Type=question_type)
            if unit:
                query &= Q(Unit=unit)
            questions = Questions.objects.filter(query)
            # Now you have filtered questions based on the selected subject and type
            # Proceed to generate notes
            return render(request, 'html/notes_template.html', {'questions':

```

```

questions,'subject':subject_name,'questiontype':question_type,'unit':unit})
    if subject_name == "All Subjects":
        subject_name=None
        questions = Questions.objects.all()
        return render(request, 'html/notes_template.html', {'questions':
questions,'subject':subject_name,'questiontype':question_type,'unit':unit})
    else:
        # Handle when no subject or type is selected
        # Redirect or render an appropriate response
        subjects = Subjects.objects.all()
        context = {'subjects': subjects}
        return render(request, 'html/notes_generation.html',context)
else:
    return redirect('/')

```

7.1.5 urls.py

```

from django.contrib import admin
from django.urls import path
from django.conf import settings
from django.conf.urls.static import static
from Login_signupApp import views
from questionpapercreation.views import
question_paper_preview,create_question_paper,automatic_view,question_papers,notesgeneration
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", views.login_view,name='login_signup'),
    path('forgetpassword', views.forgetpass_view,name='forgetpass_view'),
    path('dashboard/', views.dashboard,name='dashboard'),
    path('preview/<int:question_paper_id>/', question_paper_preview,name='preview'),
    path('create/',create_question_paper,name='create'),
    path('automatic/',automatic_view,name='automatic'),
    path('logout/', views.logout_view, name='logout'),
    path('profile/', views.profile_view, name='profile'),
    path('question_papers/',question_papers, name='question_papers'),
    path('notesgeneration/',notesgeneration, name='notesgeneration'),
]+ static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

```

CHAPTER 8

SYSTEM TESTING

8.1 Testing Methodologies

Testing is an integral part of the development process to ensure the quality, reliability, and functionality of the Manual and Automatic Question Paper Generation System. The project employs a comprehensive testing methodology encompassing various levels of testing, including unit testing, integration testing, and system testing.

8.1.1 Unit Testing:

Python Unit Testing Framework: The built-in Python unit test framework is utilized for writing and executing unit tests for individual components, functions, and modules of the application. Test cases are designed to validate the correctness of code logic, edge cases, and boundary conditions.

8.1.2 Integration Testing:

Django Testing Framework: Django's testing framework is leveraged for integration testing, focusing on validating the interaction between different components and ensuring seamless integration of modules, views, and database interactions. Integration tests cover scenarios such as URL routing, form submissions, and database operations.

8.1.3 System Testing:

End-to-End (E2E) Testing: End-to-end testing is conducted to evaluate the system's behavior and functionality from the user's perspective. Selenium WebDriver is employed to automate browser-based tests, simulating user interactions and verifying the correctness of user flows, form submissions, and data retrieval.

By following this testing methodology, the Manual and Automatic Question Paper Generation System aims to deliver a robust, reliable, and user-friendly application that meets the requirements and expectations of its stakeholders while adhering to quality standards and best practices

CHAPTER 9

RESULT & DISCUSSION

Signin/Signup Page:

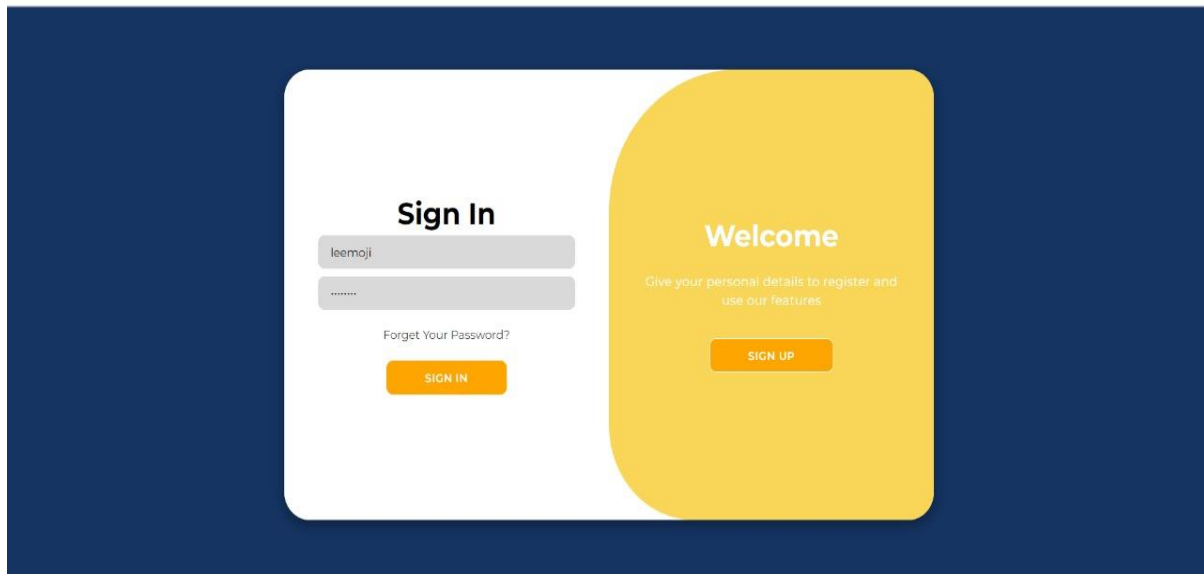


Figure 9.1 Singin/Signup page

Dashboard Page:

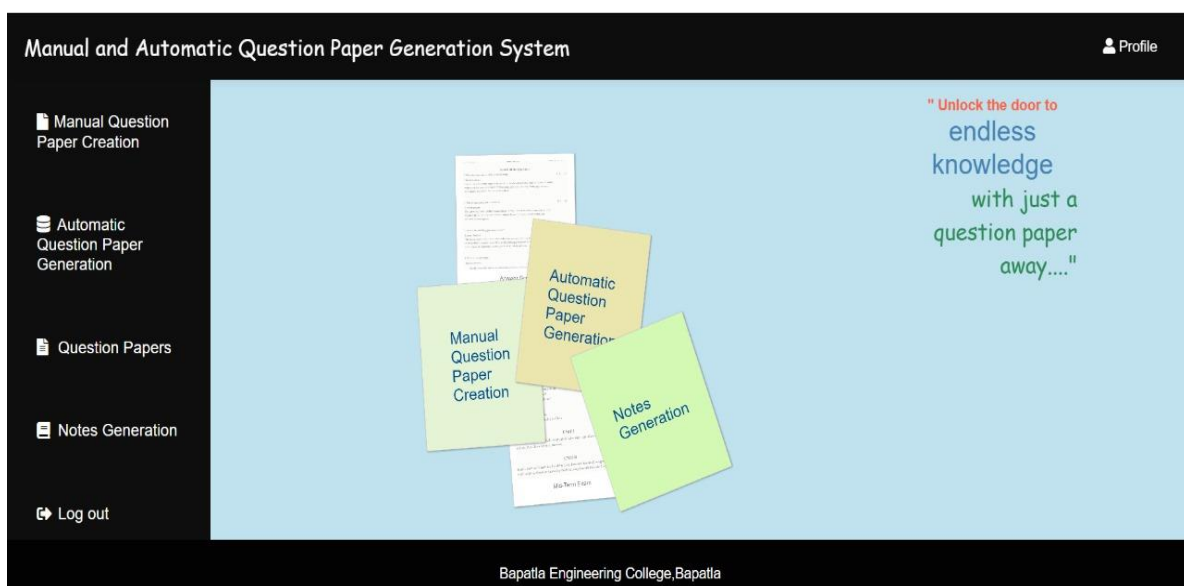



Figure 9.2 Dashboard Page


Profile Page:

Manual and Automatic Question Paper Generation System
Profile



- Dashboard
- Manual Question Paper Creation
- Automatic Question Paper Generation
- Question Papers
- Notes Generation

User Profile



FirstName:
Leemoji Reddy

LastName:
Appala

Email:
leemoji.reddy@gmail.com


[Edit Info](#)

[Log out](#)

Figure 9.3 Profile page

Manual Question Paper Creation Page :

Manual and Automatic Question Paper Generation System
Profile



- Dashboard
- Manual Question Paper Creation
- Automatic Question Paper Generation
- Question Papers
- Notes Generation

Manual Question Paper Creation

Select Department:
Select

Select Semester:
Select

Name of the Exam:
Select

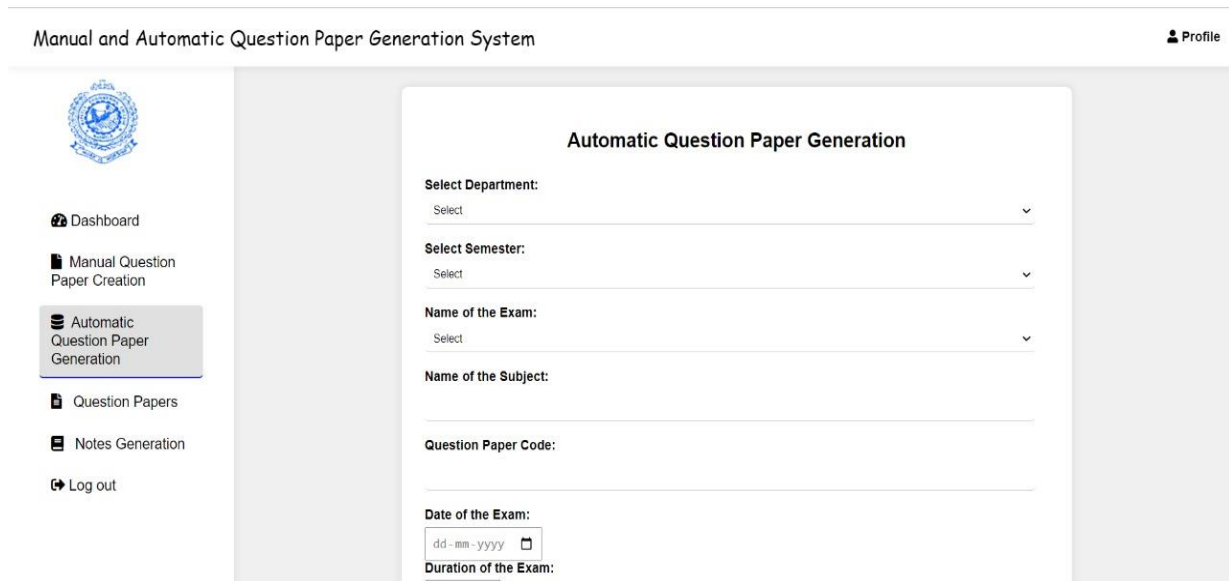
Name of the Subject:

Question Paper Code:

Date of the Exam:
dd-mm-yyyy

Figure 9.4 Manual Question Paper Creation Page

Automatic Question Paper Generation Page:



Manual and Automatic Question Paper Generation System Profile

Automatic Question Paper Generation

Select Department:

Select Semester:

Name of the Exam:

Name of the Subject:

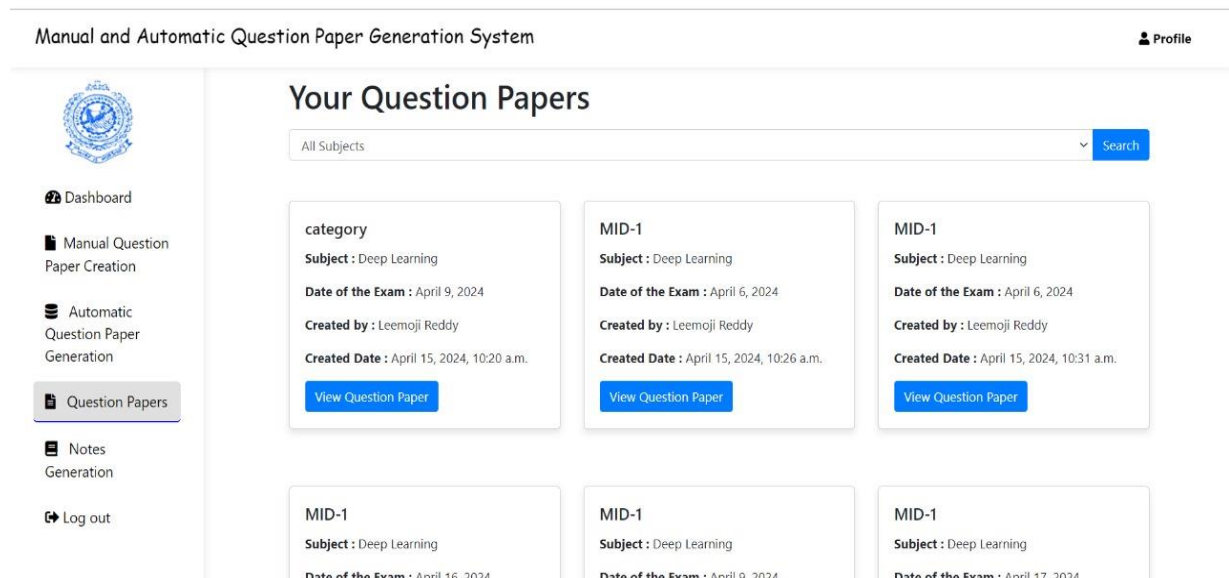
Question Paper Code:

Date of the Exam:

Duration of the Exam:

Figure 9.5 Automatic Question Paper Generation

Question Papers Page:



Manual and Automatic Question Paper Generation System Profile

Your Question Papers


All Subjects

category Subject : Deep Learning Date of the Exam : April 9, 2024 Created by : Leemoji Reddy Created Date : April 15, 2024, 10:20 a.m. <input type="button" value="View Question Paper"/>	MID-1 Subject : Deep Learning Date of the Exam : April 6, 2024 Created by : Leemoji Reddy Created Date : April 15, 2024, 10:26 a.m. <input type="button" value="View Question Paper"/>	MID-1 Subject : Deep Learning Date of the Exam : April 6, 2024 Created by : Leemoji Reddy Created Date : April 15, 2024, 10:31 a.m. <input type="button" value="View Question Paper"/>
MID-1 Subject : Deep Learning Date of the Exam : April 16, 2024	MID-1 Subject : Deep Learning Date of the Exam : April 9, 2024	MID-1 Subject : Deep Learning Date of the Exam : April 17, 2024

Figure 9.6 Question Papers

Notes Generation Page:

Manual and Automatic Question Paper Generation System
Profile



- Dashboard
- Manual Question Paper Creation
- Automatic Question Paper Generation
- Question Papers
- Notes Generation**
- Log out


Notes Generation

All Subjects
All Types
All Units

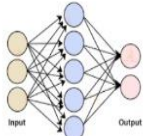
Generate Notes

Figure 9.7 Notes Generation

Preview Page



B.Tech I-sem
Total Time:17:26

1. a. q14

b. hello maq
1. Opt1
2. Opt2
3. Opt3
4. Opt4

Bapatla Engineering College:: Bapatla
(Autonomous)
Department of INFORMATION TECHNOLOGY
Assignment-1
Deep Learning
Total Marks:21
unit-1

20ITH7N
Date:April 11, 2024
[co1] [L2] [14]
[co1] [L2] [7]

Download Question Paper
Download Answer Script

Figure 9.8 Preview

Question Paper Download Page

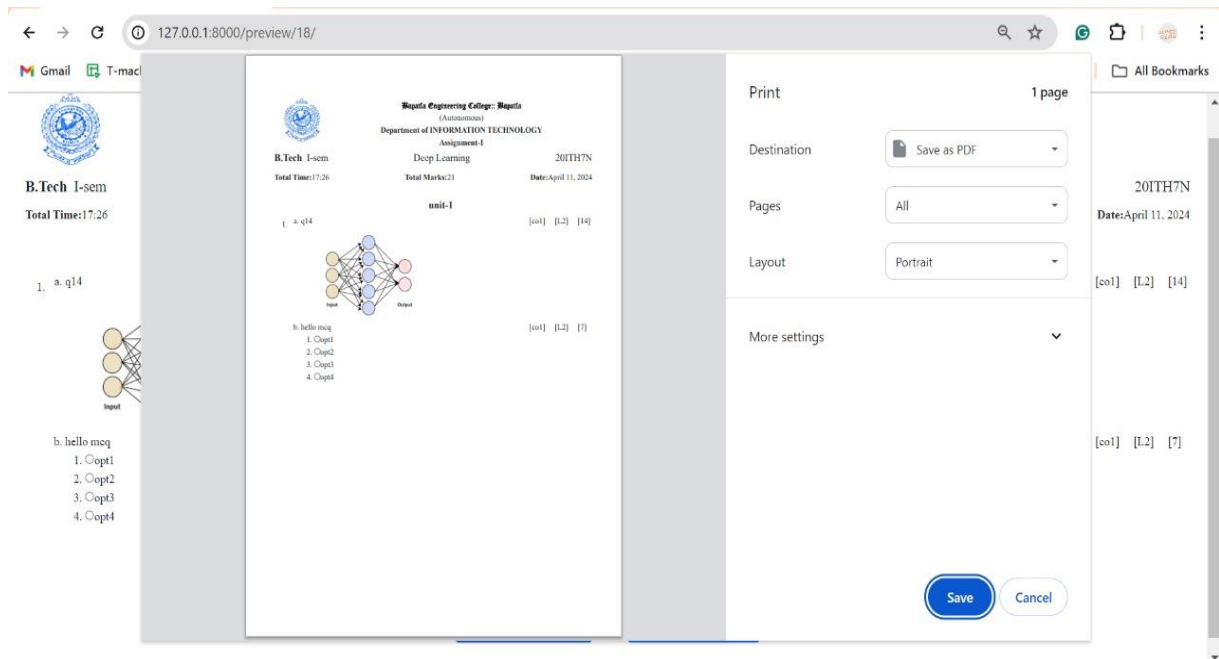


Figure 9.9 Question paper download

CHAPTER 10

CONCLUSION & FUTURE SCOPE

10.1 Conclusion

In conclusion, the manual and automatic Question Paper Generation System represents a significant advancement in the process of creating, managing, and analyzing question papers. By harnessing modern technologies and efficient database management, the system provides educators with a comprehensive and user-friendly platform for generating high-quality question papers.

The system's intuitive interface and advanced features empower educators to customize question papers according to their specific requirements, including question types, marks allocation, Bloom's taxonomy levels, and other parameters. Additionally, the system offers the functionality to generate automatic question papers based on predefined criteria such as the number of sections, main questions, sub-questions, and desired question types.

This automation streamlines the question paper generation process, saving educators time and effort while ensuring consistency and adherence to assessment standards. Administrators can monitor system usage and performance, implement security measures, and provide support to users as necessary.

Overall, the Question Paper Generation System enhances the efficiency and effectiveness of the question paper creation process, whether manual or automatic. Its implementation has the potential to streamline assessment practices, promote academic excellence, and contribute to the overall success of educational institutions.

10.2 Future Scope

Conducting Exams:

Implementation of features necessary for conducting exams, including timer functionality, question navigation, and submission options.

Dashboard Development:

Development of a dashboard interface for instructors and administrators.

Visualization of student scores, performance metrics, and other relevant data.

Editable Question Papers:

Providing editable options for instructors to modify already created question papers to meet specific needs and requirements.

REFERENCES

- [1] PostgreSQL-PostgreSQL Global Development Group, "*PostgreSQL Documentation 14*," Published by PostgreSQL Global Development Group, 2023
- [2] Django-Django Software Foundation, "*Django Documentation 5.0.4*," Published by Django Software Foundation, 2023.
- [3] HTML5 - Bruce Lawson and Remy Sharp," *Introducing HTML5*", Published by New Riders, 2010.
- [4] CSS3-Tiffany B. Brown, Estelle Weyl, and Louis Lazaris, "*CSS3 for Web Designers*," Published by A Book Apart, 2015.
- [5] JavaScript-David Flanagan, "*JavaScript: The Definitive Guide*," Published by O'Reilly Media, 2020.
- [6] Bootstrap5- Mark Otto and Jacob Thornton, "*Bootstrap 5 Documentation*," Published by Bootstrap, 2021.
- [7] Surbhi Choudhary, Abdul Rais Abdul Waheed, Shrutika Gawandi and Kavita Joshi, "*Question Paper Generator System*," International Journal of Computer Science Trends and Technology, vol. 3, issue 5, Sept – Oct 2015.
- [8] Prita Patil and Kavita Shirsat, "*An Integrated Automated Paperless Academic Module for Education Institutes*," International Journal of Engineering Science Invention Research and Development, vol. I, issue IX, March 2015.
- [9] Ashok Immanuel and Tulasi.B, "*Framework for Automatic Examination Paper Generation System*," International Journal of Computer Science Trends and Technology, vol. 6, issue 1, Jan - March 2015.
- [10] Kapil Naik, Shreyas Sule, Shruti Jadhav and Surya Pandey, "*Automatic Question Paper Generation using Randomization Algorithm*," International Journal of Engineering and Technical Research, vol. 2, issue 12, December 2014.
- [11] Dan Liu, Jianmin Wang and Lijuan Zheng, "*Automatic Test Paper Generation Based on Ant Colony Algorithm*," Journal of Software, vol. 8, no. 10, October 2013.
- [12] Abdi, A., Shamsuddin, S.M., Hasan, S., Piran, J. (2019). Deep learning-based sentiment classification of evaluative textbased on multi-feature fusion. *Information Processing & Management*, 56(4), 1245–1259.

- [13] Agarwal, M. (2012). Cloze and open cloze question generation systems and their evaluation guidelines. Master's thesis. *International Institute of Information Technology, (IIIT)*, Hyderabad, India.
- [14] Agarwal, M., & Mannem, P. (2011). Automatic gap-fill question generation from text books, In *Proceedings of the 6th Workshop on Innovative Use of NLP for Building Educational Applications* (pp. 56–64). Portland: Association for Computational Linguistics.
- [15] Aldabe, I., & Maritxalar, M. (2010). Automatic distractor generation for domain specific texts, In *Proceedings of the 7th International Conference on Advances in Natural Language Processing* (pp. 27–38). Berlin: Springer-Verlag.
- [16] Alruwais, N., Wills, G., Wald, M. (2018). Advantages and challenges of using e-assessment. *International Journal of Information and Education Technology*, 8(1), 34–37.
- [17] Amidei, J., Piwek, P., Willis, A. (2018). Evaluation methodologies in automatic question generation 2013-2018, In *Proceedings of The 11th International Natural Language Generation Conference* (pp. 307–317). Tilburg University: Association for Computational Linguistics.
- [18] Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Lawrence Zitnick, C., Parikh, D. (2015). Vqa: Visual question answering, In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 2425–2433).
- [19] Bhatia, A.S., Kirti, M., Saha, S.K. (2013). Automatic generation of multiple choice questions using wikipedia, In *Proceedings of the Pattern Recognition and Machine Intelligence* (pp. 733–738). Berlin: Springer-Verlag.
- [20] Bilker, W.B., Hansen, J.A., Brensinger, C.M., Richard, J., Gur, R.E., Gur, R.C. (2012). Development of abbreviated nine-item forms of the Raven's standard progressive matrices test. *Assessment*, 19(3), 354–369.
- [21] Bin, L., Jun, L., Jian-Min, Y., Qiao-Ming, Z. (2008). Automated essay scoring using the KNN algorithm, In *2008 International Conference on Computer Science and Software Engineering*, 1 (pp. 735–738). Washington, DC: IEEE.