Acknowledgement

We are most grateful to our Supervisor, Dr. Amitabha Nath, Assistant Professor in the Department of Information Technology. We are extremely grateful and indebted to him for sharing his expertise and providing sincere and valuable guidance and encouragement.

We are also indebted to the Head of the Department, Prof. Debdatta Kandar, Department of Information Technology, for permitting us to pursue this project.

We would like to take this opportunity to thank all the respected teachers of this department for being a perennial source of inspiration and showing the right path at times of necessity. We acknowledge that this project was completed entirely by our group and not by someone else.

Abstract

Academic departments generate annual reports to document faculty achievements, research contributions, and departmental activities. The traditional manual approach is time-consuming, error-prone, and inefficient. Our project presents an Automated System for Generating Department Annual Reports, which streamlines data collection, organization, and report generation.

The system includes a web-based frontend where faculty members can enter their details, including research publications, projects, events, patents, and academic collaborations. A backend database efficiently stores and manages these data, ensuring accuracy and consistency. The system also facilitates report generation by compiling the stored data into a structured format, reducing manual workload, and improving efficiency.

The automated approach ensures that reports are generated in a standardized format with minimal manual intervention, making the process more reliable and time-efficient.

This system provides an efficient and scalable solution for academic reporting, which is beneficial to faculty and administrators alike.

Contents

1 Introduction

1.1 Overview

In academic institutions, generating an annual report is a crucial but time-consuming task. Departments are required to compile data related to faculties. Traditionally, this process is manual and involves spreadsheets, word processing, and extensive proofreading, which often leads to errors, inefficiencies, and inconsistencies.

To address these challenges, our project proposes an Automated System for Generating Department Annual Reports that streamlines data collection, storage, and report generation. The system ensures accuracy, reduces manual effort, and provides a structured way to efficiently generate annual reports.

1.2 Problem Definition

The current method of report generation in many institutions involves collecting data from faculty members, compiling it into a document, and formatting it manually. This process is as follows:

- Time-consuming – Requires repeated data entry and formatting.
- Error-prone – Human errors can lead to inconsistencies in reports.
- Difficult to update – Any modifications require re-editing the entire document.
- Lacks standardization – Formatting inconsistencies may arise due to manual adjustments.

Thus, an automated system is required to digitize the process, enhance accuracy, and improve efficiency.

1.3 Objectives

The primary objectives of this project are:

- Automate Report Generation – Minimize manual effort by auto-compiling faculty data.
- Ensure Data Accuracy – Reduce human errors by using a structured database.
- Standardized Formatting – Maintain uniform report structure as per institutional guidelines.
- Easy Data Management – Store, retrieve, and update faculty records effortlessly.
- Enhance Usability – Provide a user-friendly interface for faculty and administrators.

1.4 Proposed Solution

The system will have a centralized database where faculty members can enter their academic activities (e.g., research papers, books, patents, awards). An automated report generator will extract this data and format it into a well-structured annual report. The key components include:

- Database Management – Stores faculty profiles, research contributions, and events.
- Web-based Interface – Allows faculty and administrators to input and review data.
- Report Generation Module – Automatically compiles data into a formatted document.

1.5 Organization of the report

The remaining chapters of this report are structured as follows:

- Chapter 2: Requirement Specifications - Discusses the requirements.
- Chapter 3: System Design – Discusses the system architecture.
- Chapter 4: Implementation – Covers technologies used, modules, and system functionalities.
- Chapter 5: Results – Presents sample generated reports.
- Chapter 6: Conclusion and Future Work – Summarizes achievements and suggests possible enhancements.

2 Requirement Specifications

2.1 Introduction

This chapter outlines the functional and non-functional requirements of the Automated System for Generating Department Annual Reports. The system is designed to automate the collection, storage, and formatting of faculty and department data for seamless report generation.

2.2 Functional Requirements

Functional requirements define the specific features and functionalities the system must support.

2.2.1 User Roles

The system will have different user roles with specific permissions:

1. Head of Department(HoD)
    - Upload official department reports/documents.
    - View and manage(reject/approve) faculty details and contributions.
    - Generate the final annual report.
2. Faculty Members
    - Submit details related to research publications, projects, patents, etc.
    - Edit or update their submitted information.
    - Download department reports (uploaded by the HoD).

2.2.2 Data Entry and Management

Faculty members can enter/update data under categories like:

- Research Publications (Journals, Conferences, Books, etc.)
- Patents (Published/Granted)
- Projects (Completed/Ongoing)
- Events Organized and Attended
- Awards and Recognitions

The system will store and organize data in a structured format using a relational database.

2.2.3 Report Generation and export

- Automatically compile faculty data into a well-structured annual report.
- Generate reports in PDF format.

2.3 Non-Functional Requirements

Non-functional requirements define the quality attributes of the system.

2.3.1 Usability Requirements

- The user interface should be intuitive and easy to navigate.
- Minimal training should be required to use the system.

2.3.2 Scalability Requirements

- Should be adaptable for use by multiple departments if required.

2.3.3 Reliability Requirements

- Should be highly reliable, ensuring file uploads and report generation without crashing.
- In case of system failures, should provide meaningful error messages for troubleshooting.

2.4 Software Requirements

- Frontend: React.js (for a user-friendly interface).
- Backend: Node.js with Express.js (to handle API requests).
- Database: MySQL (for structured data storage).

2.5 Constraints

- Internet connectivity is required for accessing and updating records.
- Faculty members must enter accurate data to maintain report integrity.

3 System Design

3.1 Introduction

System design is a critical phase where the architecture, data flow, and interactions between different components are defined. This chapter provides an overview of the system's architecture, database design, and process flow to ensure efficient automated report generation for academic departments.

3.2 System Architecture

The system follows a three-tier architecture, which consists of:

1. Presentation Layer (Frontend):
    - Developed using React.js, providing an intuitive UI.
    - Users (faculty, HoD) interact through forms and dashboards.
2. Application Layer (Backend):
    - Node.js with Express.js handles API requests.
    - Processes data, enforces security and interacts with the database.
3. Database Layer:
    - MySQL stores structured faculty and department data.
    - Ensures data integrity, indexing, and efficient retrieval.

3.2.1 System Architecture Diagram

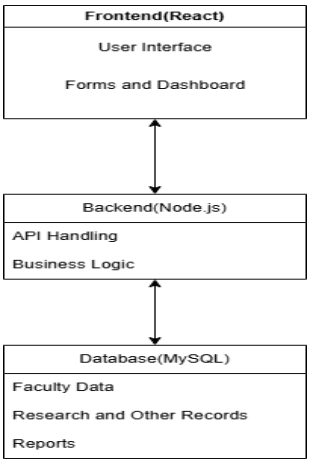

Figure 3.2.1: System Architecture

3.3 Data FLow Diagram(DFD)

3.3.1 DFD Level 0 (Context Diagram)

This represents the system at a high level.

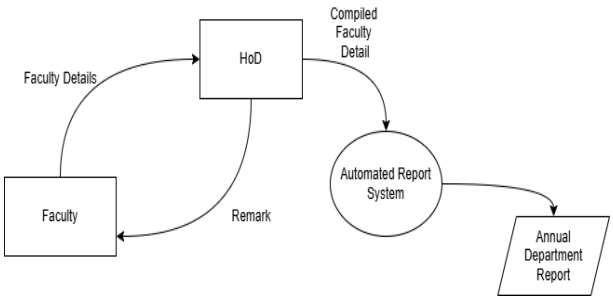

Figure 3.3.1: Context Diagram

3.3.2 DFD Level 1 (Detailed Data Flow

Expanding the system further:

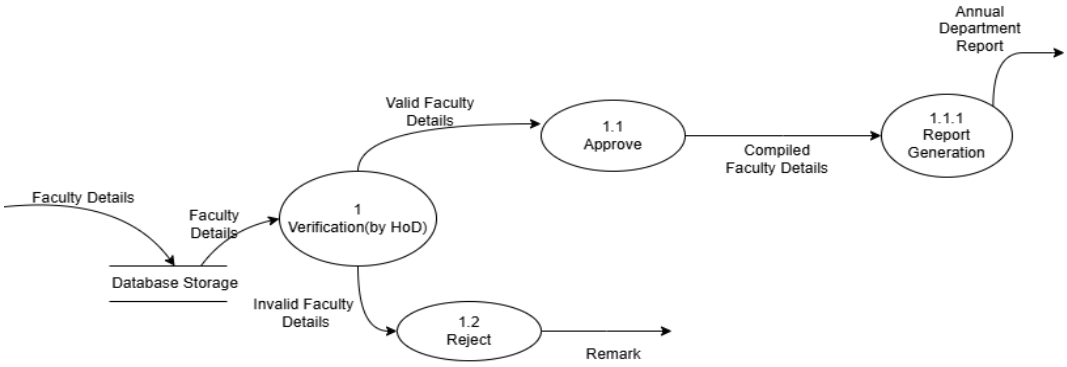

Figure 3.3.2: Level 1 DFD

3.4 Relation Schema Design

Here's how the system's database schema is structured:

1. Table: users

   Columns:

   faculty_id varchar(50) PK

   password varchar(100)

   role enum('HoD','Faculty')

   department enum('Information Technology','Others')

2. Table: faculty_details

   Columns:

   faculty_id varchar(50) PK

   name varchar(100)

   designation varchar(50)

   qualification text

   specialization text

   avatar varchar(255)

3. Table: edited_books

   Columns:

   id int AI PK

   faculty_id varchar(50)

   editor varchar(255)

   title varchar(255)

   publisher varchar(255)

   year varchar(4)

   isbn varchar(20)

4. Table: book_chapters

   Columns:

   id int AI PK

   faculty_id varchar(255)

   author varchar(255)

   title varchar(255)

   booktitle varchar(255)

   editor varchar(255)

   publisher varchar(255)

   year varchar(4)

   isbn varchar(20)

   chaptersPages varchar(10)

5. Table: popular_articles

   Columns:

   id int AI PK

   faculty_id varchar(255)

   author varchar(255)

   title varchar(255)

   newspaperMagazine varchar(255)

date date

6. Table: consultancy_projects

   Columns:

   faculty_id varchar(255)

   title varchar(255)

   totalOutlay decimal(10,2)

   status enum('new','on-going','completed')

   consultantName varchar(255)

   instituteType enum('multi','single')

   fundingAgency varchar(255)

   fundReceived decimal(10,2)

   tenure varchar(20)

7. Table: patents

   Columns:

   id int AI PK

   application_no varchar(255)

   title varchar(255)

   date date

   type enum('National','International')

   published_granted enum('Published','Granted')

   faculty_id varchar(255)

8. Table: events

   Columns:

   id int AI PK

   eventTitle varchar(255)

   date date

   faculty_id varchar(255)

   coordinatorName varchar(255)

9. Table: attended_faculty

   Columns:

   id int AI PK

   eventTitle varchar(255)

   eventAuthor varchar(255)

   eventDetails text

   date date

   faculty_id varchar(255)

10. Table: attended_events

    Columns:

    id int AI PK

    scholar_name varchar(100)

    event_title varchar(255)

    event_author varchar(100)

    event_details text

    event_date date

    faculty_id varchar(255)

11. Table: collaborations

    Columns:

    id int AI PK

    collaborator varchar(100)

    institution varchar(100)

    projectTitle varchar(100)

    type enum('National','International')

    faculty_id varchar(255)

12. Table: activities

    Columns:

    id int AI PK

    faculty_id varchar(255)

    name varchar(255)

    details text

    status enum('Pending','Ongoing','Complete')

13. Table: honours

    Columns:

    id int AI PK

    faculty_id varchar(50)

    name varchar(100)

    title varchar(255)

    organization varchar(255)

    award varchar(255)

    awardYear varchar(4)

14. Table: faculty_submissions

    Columns:

    id int AI PK

    faculty_id varchar(255)

    faculty_name varchar(255)

    designation varchar(255)

    qualification text

    specialization text

    status enum('Approved','Pending','Rejected')

    remark text

15. Table: research_projects

    Columns:

    id int AI PK

    faculty_id varchar(50)

    projectTitle varchar(255)

    totalOutlay decimal(10,2)

    status enum('New','Ongoing','Completed')

    name varchar(255)

    instituteType enum('Single','Multi')

    fundingAgency varchar(255)

    fundReceived decimal(10,2)

    created_at timestamp

    updated_at timestamp

    tenure varchar(20)

4 User Interface and Functionalities

4.1 Login

This is the Login page where user can login using login ID, password, role and department. The web application only applies to the Department of Information Technology currently but can be further expanded in the future to include other departments.

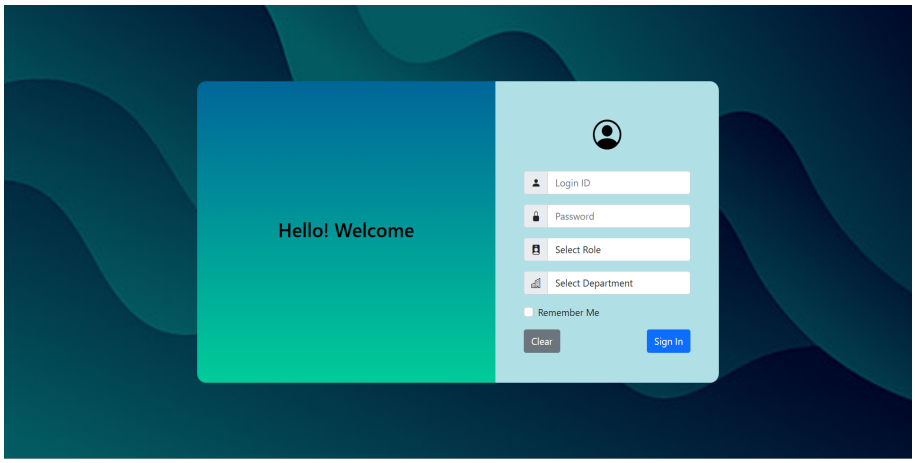

Figure 4.1.0: Login

4.2 Login with Invalid Credentials

The login page gives an error when user tries to login using invalid credentials.

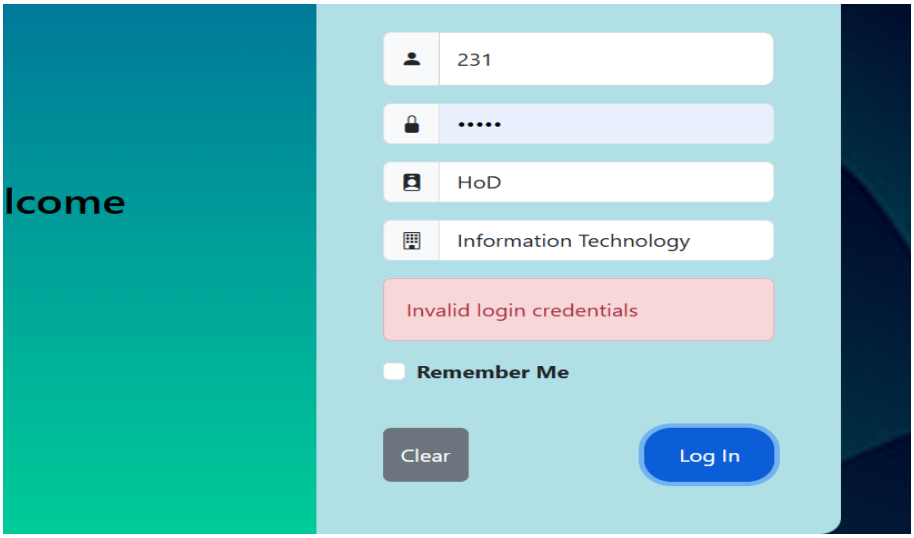

Figure 4.2.0: Login with Invalid Credentials

4.3 Faculty Dashboard

The faculty dashboard shows includes a side panel, nav bar and a window that shows cards containing faculty details and university details. It also includes a table showing faculty submission status with remarks.

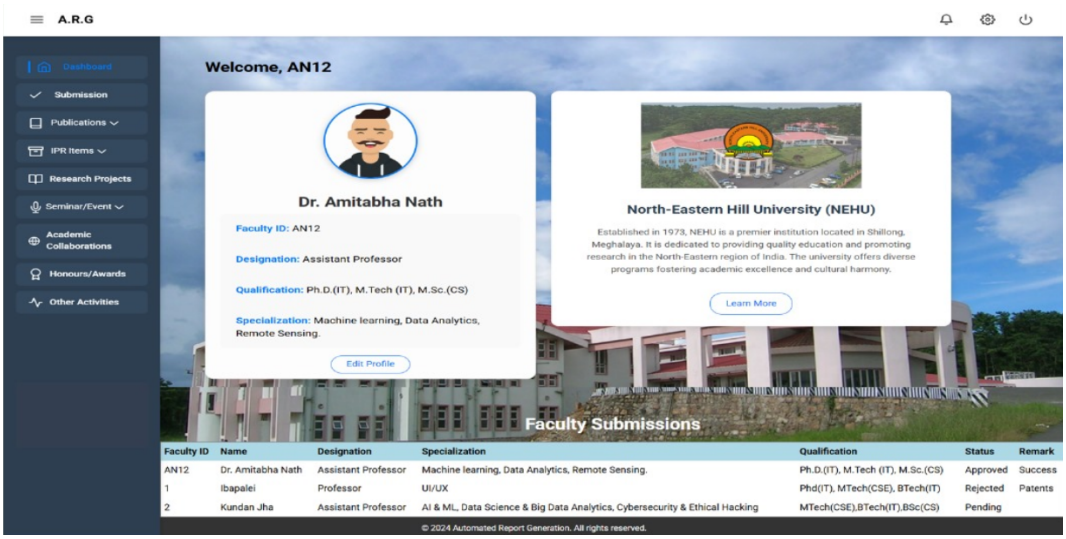

Figure 4.3.0: Faculty Dashboard

4.3.1 Settings

The settings button allows users to shift between light mode and dark mode.

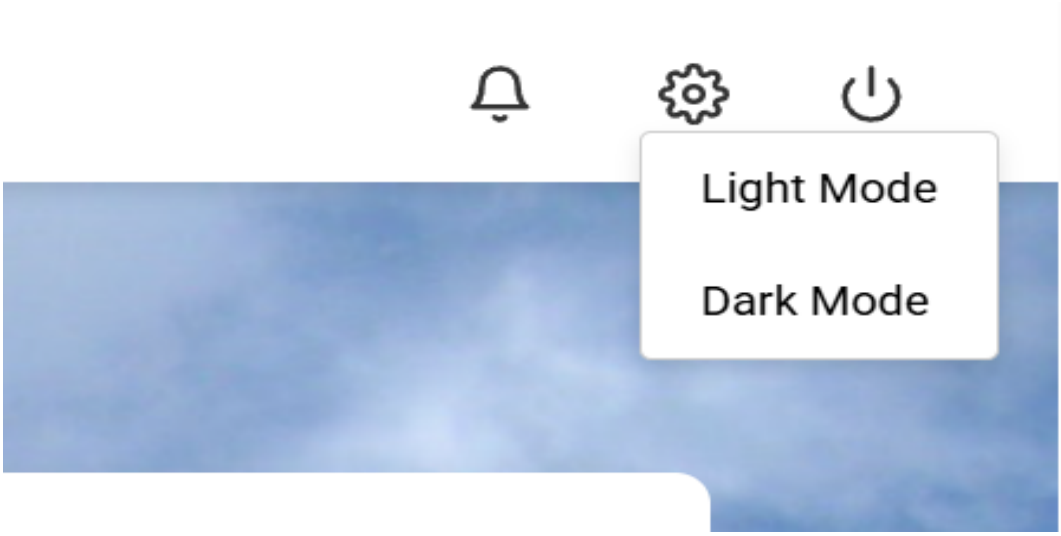

Figure 4.3.1: Modes

4.3.2 Logout

The dashboard also contains a power button which allows users to logout.

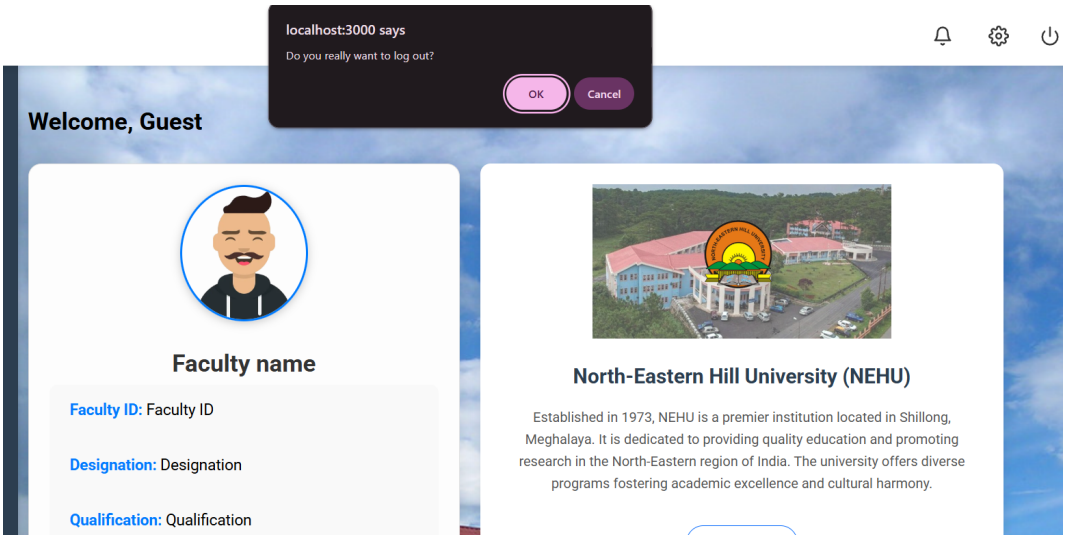

Figure 4.3.2: Logout

4.3.3 Uploaded Files

The notification button allows users to view files uploaded by the HoD.

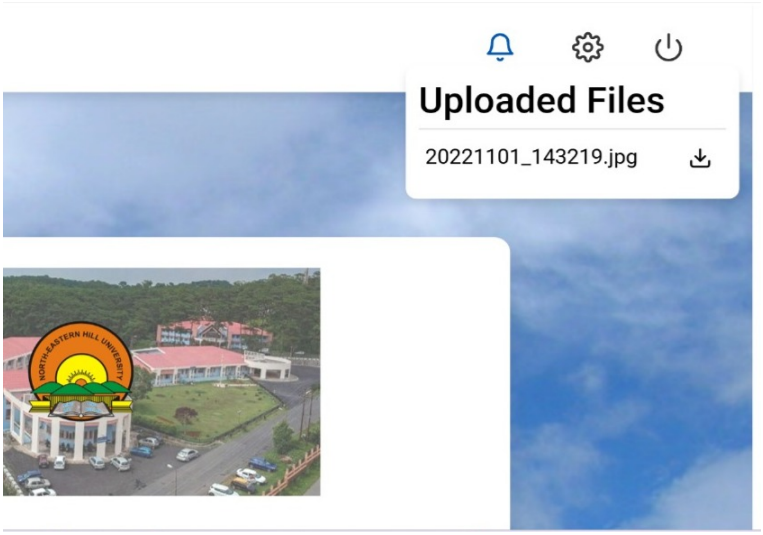

Figure 4.3.3: Uploaded Files

4.4 Faculty Submissions

This is the final submission form which sends all the data saved by the particular faculty to the HoD and the HoD can view all the details.

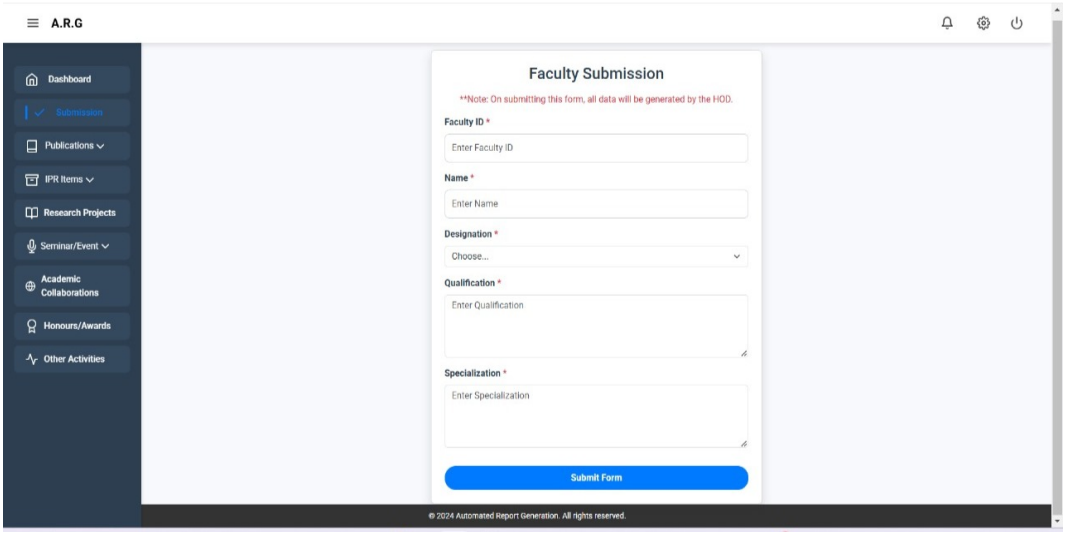

Figure 4.4.0: Faculty Submission

4.5 Forms

The faculty users are supposed to enter details in forms. The following figures show example forms.

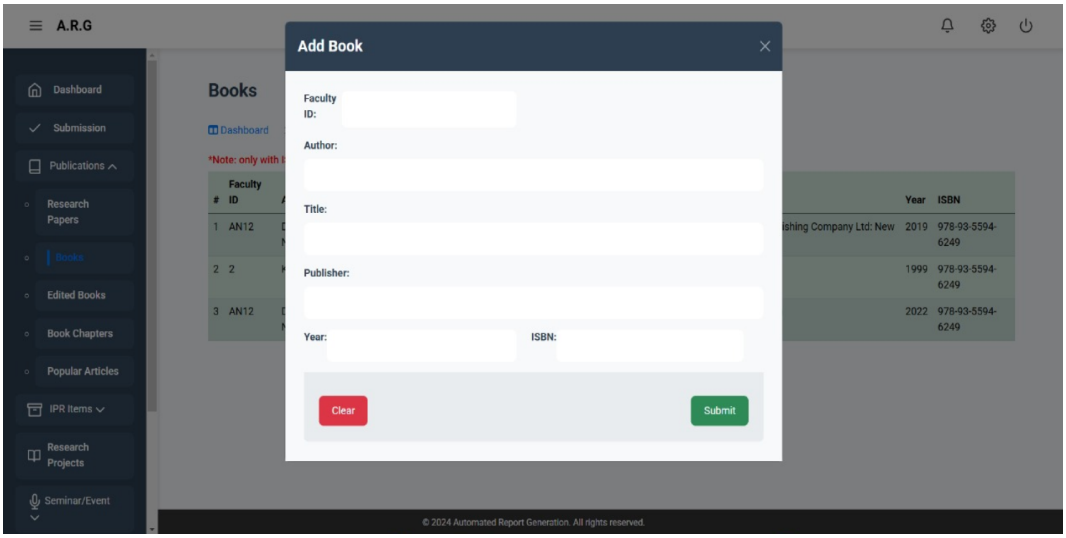

Figure 4.5.0: Form for Book details

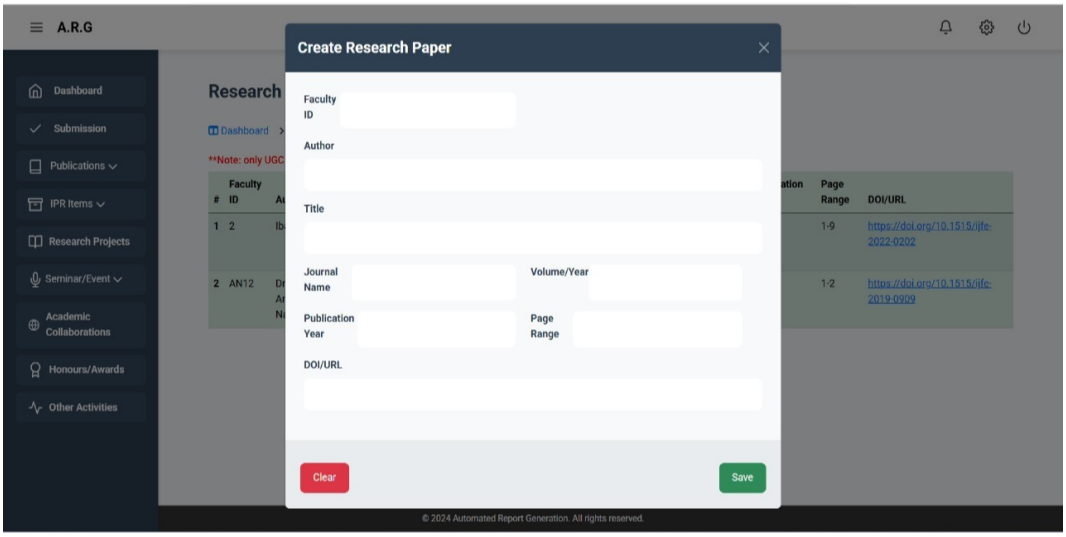

Figure 4.5.0: Form for Research Papers

4.6 Form Validation

Validations are added to all the forms to ensure users enter correct and properly formatted data.

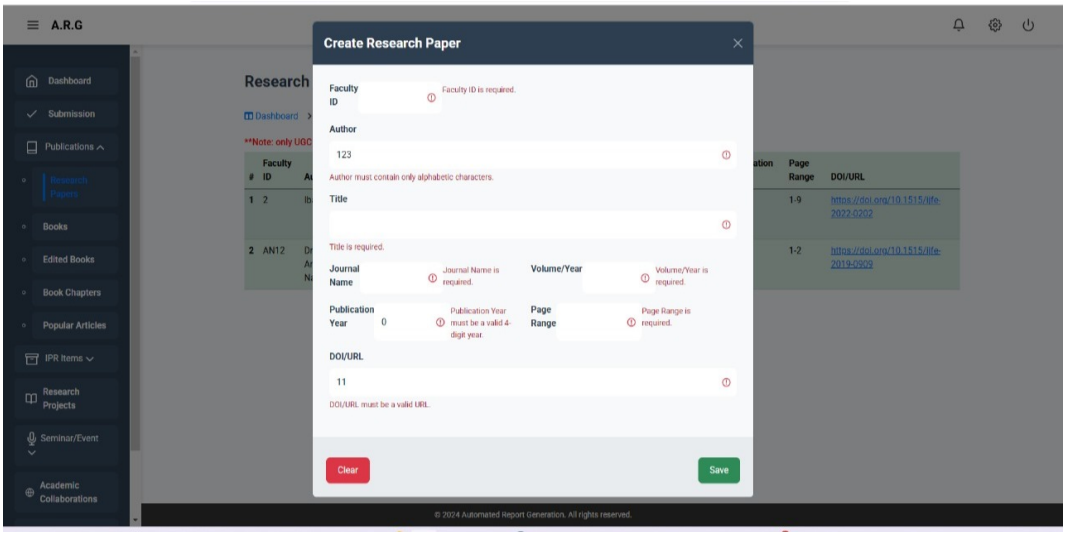

Figure 4.6.0: Example Form Validation

4.7 Form Tables

Form Tables display submitted details with Add and Delete options.

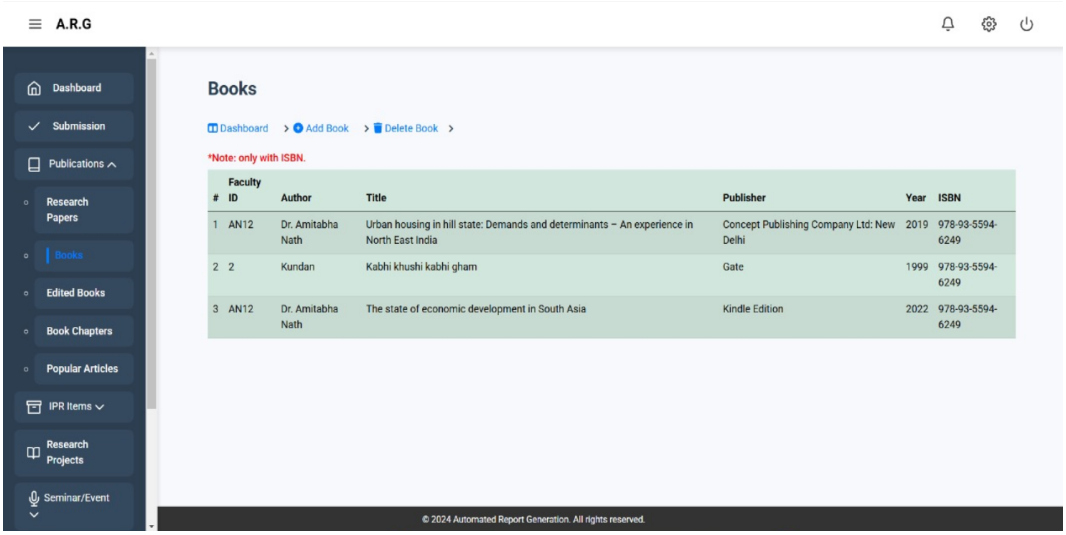

Figure 4.7.0: Form Tables

The add option allows users to add new details and the delete options allows them to delete records.

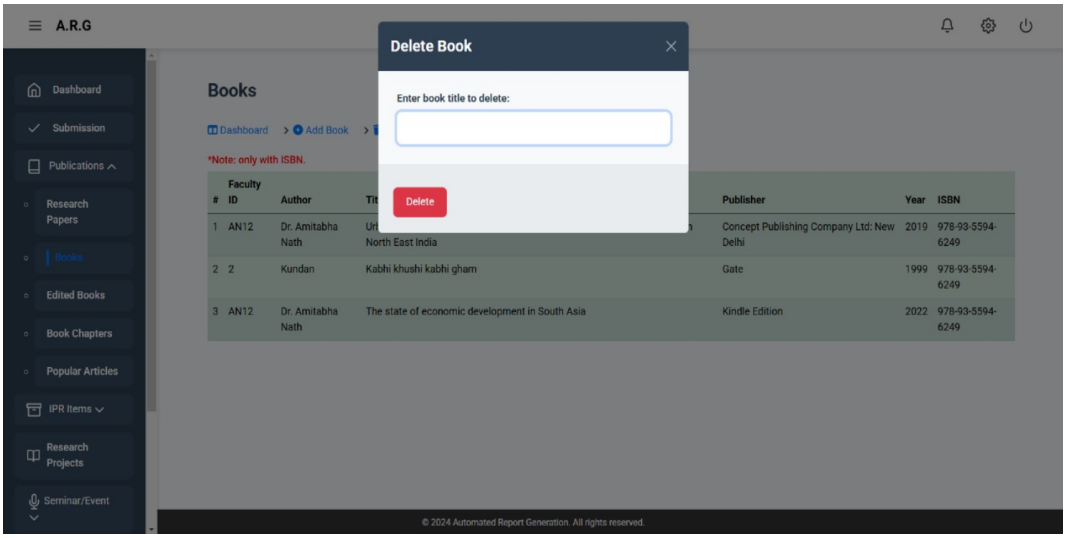

Figure 4.7.0: Deletion

4.8 HoD Dashboard

The HoD Dashboard contains a side panel, nav bar, cards with HoD and University details.

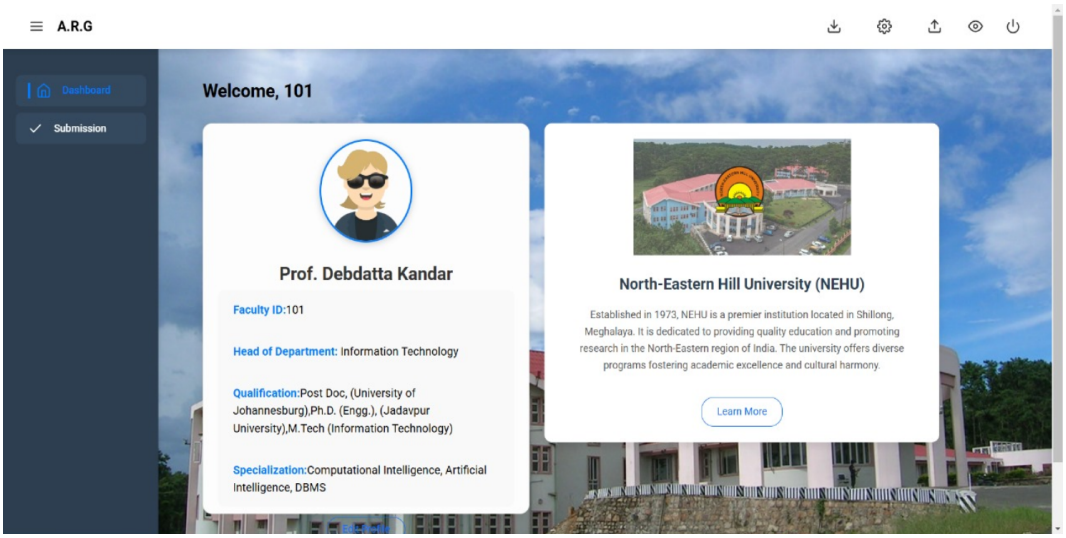

Figure 4.8.0: HoD Dashboard

4.8.1 Settings

The settings button allows users to shift between light mode and dark mode similar to Faculty Dashboard.

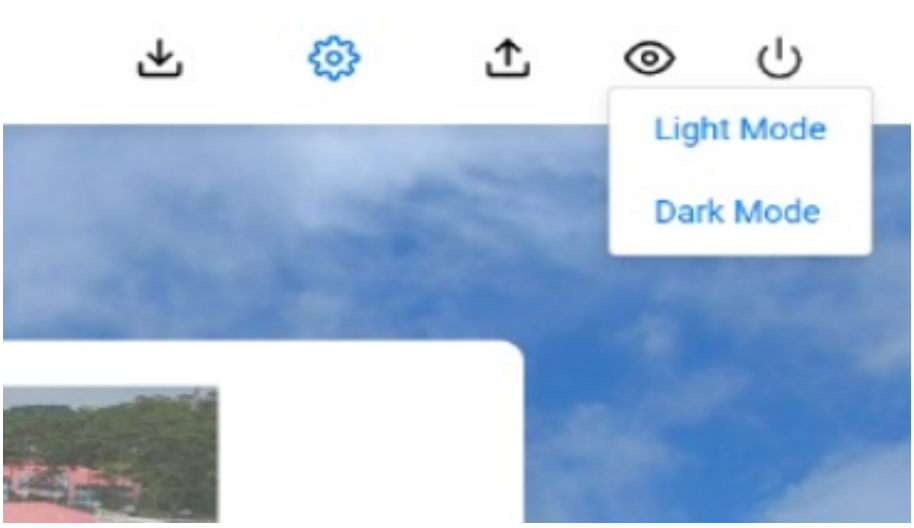

Figure 4.8.1: Modes(HoD Dashboard)

4.8.2 Functionalities

The HoD dashboard has some additional functions like uploading files and viewing them.

1. Upload Files: Allows the HoD to upload files from his device.

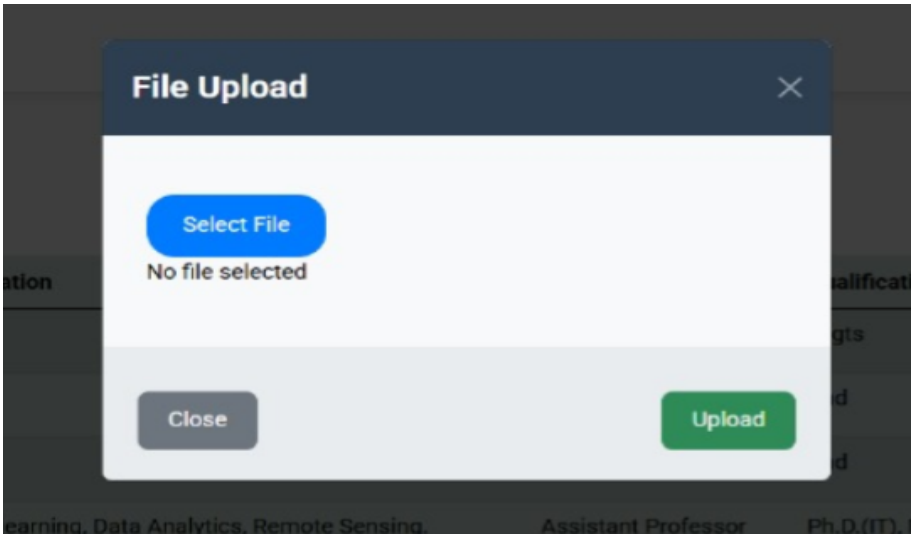

Figure 4.8.2: File Upload

2. View Files: Allows HoD to view and delete the files that he/she has uploaded.

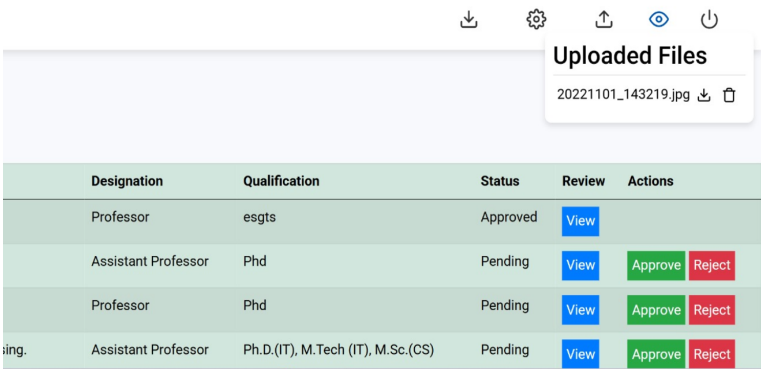

Figure 4.8.2: View Files

4.8.3 Confirmation Table

The confirmation show fetches all submitted data and allows HoD to approve/reject and put remarks. At first, the page shows an empty table with a Fetch Data button.

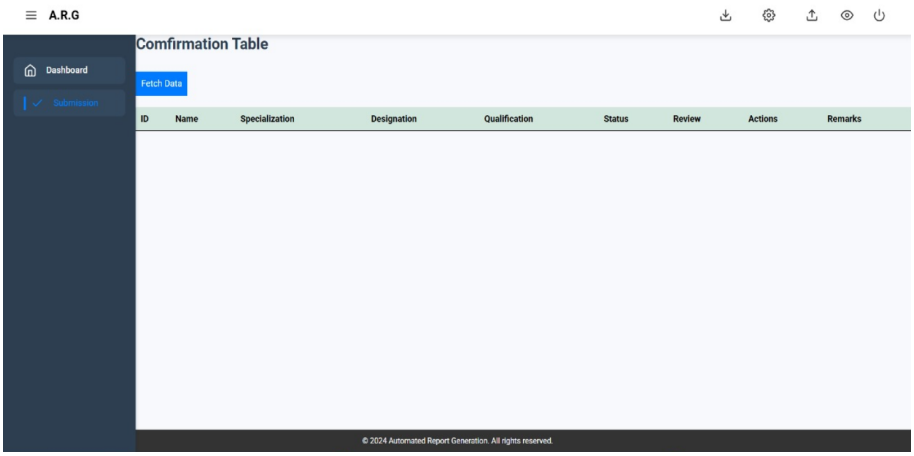

Figure 4.8.3: Confirmation Table

When the Fetch Data button is clicked, the table shows data submitted by different faculties for approval. The table also contains a view button for each faculty.

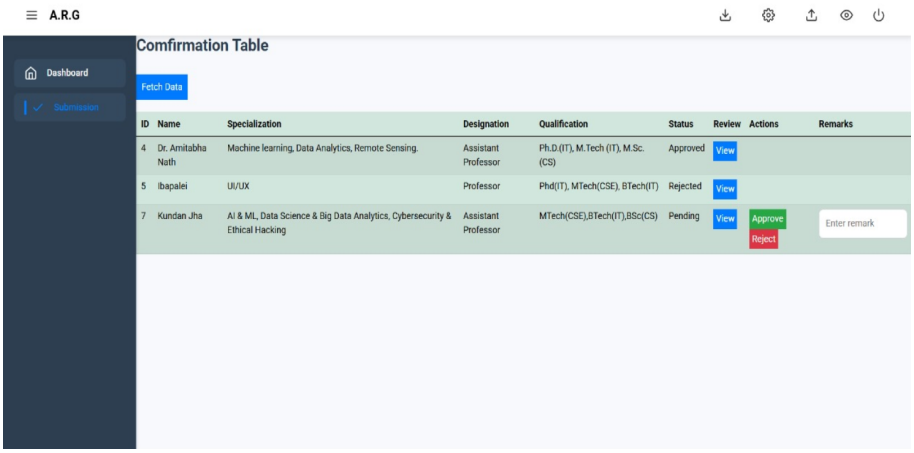

Figure 4.8.3: Fetched Data

The view button shows all the details submitted by a particular faculty.

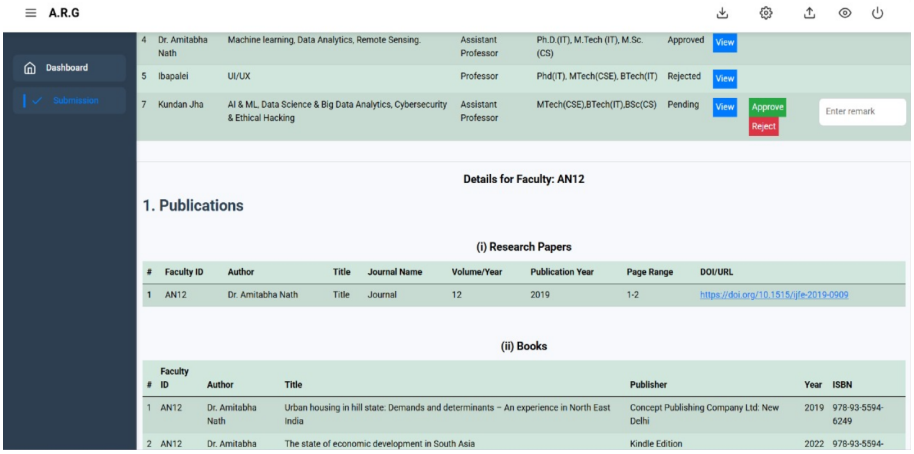

Figure 4.8.3: View Data

The HoD then verifies all the submitted data and approves if everything is correct; otherwise he rejects with a remark which informs the faculty about the area where he needs to make corrections. This is reflected in the Faculty Dashboard which displays the status and remarks in the Faculty Submissions table.

4.9 Report Generation

The Final report is generated by the HoD using the download button as a PDF File.

The following figure shows the download button on the HoD Dashboard that allows the HoD to generate the annual report and save it to his device.

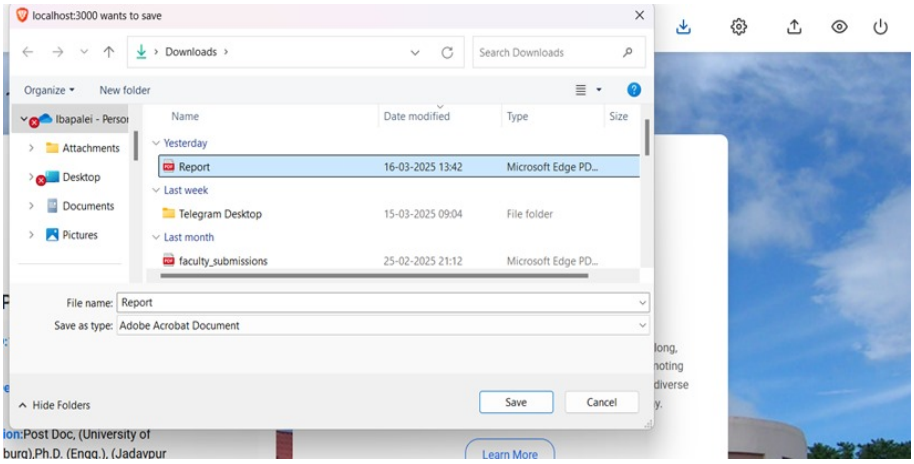

Figure 4.9.0: Generate Report

The following figure shows the generated final report:

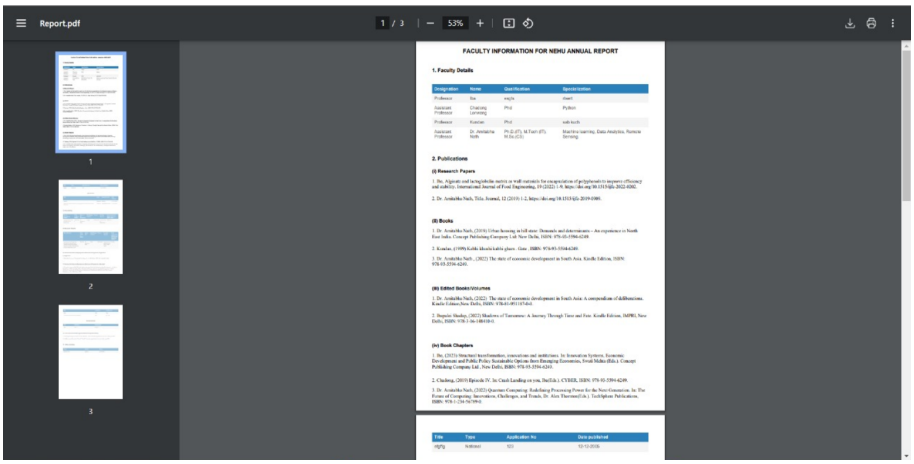

Figure 4.9.0: Report

5 Implementation

The implementation phase focuses on developing the system components, integrating them, and ensuring smooth functionality.

5.1 Technology Stack Used

Component	Technology
Frontend	React.js, HTML, CSS, Bootstrap
Backend	Node.js
Database	MySQL
Storage	Local server

5.2 System Modules

1. User Role Management
    - Faculty Login – Faculty can log in and update their records.
    - HoD Login – Can review faculty data and generate reports.
2. Faculty Data Management
    - Faculty members can add, edit, and delete details such as Personal Information, Research Papers, Books, Book Chapters, Edited Books, Patents, Research Projects, Events, Academic Collaborations, Awards.
    - Data is validated before saving to the database.
3. Automated Report Generation
    - HoD can generate customized department reports for any academic year.
    - The system automatically aggregates faculty data for structured reports.

4. File Upload System
    - HoD can upload important files (e.g., circulars, templates).
    - The uploaded files are available for downloading at both ends, HoD and Faculty.
5. Database Design
    - A well-structured relational database ensures efficient data retrieval.

5.3 Code Implementations

This section shows some code segments and logic behind certain functionalities.

5.3.1 Frontend Logic

1. **Frontend Routing Logic**

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <LoginComponent />,
  },
  {
    path: "/layout",
    element: <LayoutComponent />,
    children: [
      {
        path: "page\_name",
        element: <PageComponent />,
      },
      {
```

```
        path: "nested\_page\_name",
        element: <NestedPageComponent />,
      },
    ],
  },
]);
const App = () => {
  return <RouterProvider router={router} />;
};
```

2. Code Implementation for Forms

```
function FormComponent({title,formFields,validationSchema,
onSubmit}){
  const initialState = formFields.reduce((acc, field) => {
    acc[field.name] = "";
    return acc;
  }, {});

  const [formData, setFormData] = useState(initialState);
  const [formErrors, setFormErrors] = useState({});

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData((prev) => ({ ...prev, [name]: value }));
```

```
  setFormErrors((prev) => ({ ...prev, [name]: "" }));
};

const validateForm = () => {
  let errors = {};
  let isValid = true;

  formFields.forEach((field) => {
    if (validationSchema[field.name]) {
      const error = validationSchema[field.name]
      (formData[field.name]);
      if (error) {
        errors[field.name] = error;
        isValid = false;
      }
    }
  });

  setFormErrors(errors);
  return isValid;
};

const handleSubmit = (e) => {
  e.preventDefault();
```

```
  if (validateForm()) {
    onSubmit(formData);
  }
};

return (
  <div className="modal-content">
    <div className="modal-header">
      <h5 className="modal-title">{title}</h5>
      <button type="button" className="btn-close"
      data-bs-dismiss="modal"></button>
    </div>
    <div className="modal-body">
      <form onSubmit={handleSubmit}>
        {formFields.map((field) => (
          <div key={field.name} className="mb-3">
            <label htmlFor={field.name} className="form-label">
              {field.label}
            </label>
            <input
              type={field.type}
              id={field.name}
              name={field.name}
              value={formData[field.name]}
```

```
                onChange={handleChange}
                className={`form-control ${formErrors[field.name]
                ? "is-invalid" : ""}`}
              />
              {formErrors[field.name] && (
                <div className="invalid-feedback">
                {formErrors[field.name]}</div>
              )}
            </div>))}
          <div className="modal-footer">
            <button type="button" className="btn btn-danger"
            onClick={() => setFormData(initialState)}>
              Clear
            </button>
            <button type="submit" className="btn btn-primary">
              Submit
            </button>
          </div>
        </form>
      </div>
    </div>
  );}
```

3. Code Implementation for HoD Layout

```
export default function Hodlay() {
```

```
const [data, setData] = useState({ faculty: [] });
const [file, setFile] = useState(null);
const [isFilesOpen, setIsFilesOpen] = useState(false);
const fileRef = useRef(null);

useEffect(() => {
  fetchData("faculty");
}, []);

const fetchData = async (type) => {
  try {
    const response = await axios.get
    (`http://localhost:5000/${type}`);
    setData((prev) => ({ ...prev, [type]: response.data }));
  } catch (error) {
    console.error(`Error fetching ${type}:`, error);
  }
};

const handleFileChange = (e) => setFile(e.target.files[0]);

const uploadFile = async () => {
  if (!file) return alert("Select a file first");
  const formData = new FormData();
```

```
  formData.append("uploaded_file", file);
  try {
    await axios.post("http://localhost:5000/upload",formData,{
      headers: { "Content-Type": "multipart/form-data" },
    });
    setFile(null);
    fetchData("files");
  } catch (error) {
    console.error("Upload failed:", error);
  }
};

const handleDownload = async (fileUrl) => {
  try {
    const response = await axios.get
    (`http://localhost:5000${fileUrl}`, {
      responseType: "blob",
    });
    const url = window.URL.createObjectURL(new Blob
    ([response.data]));
    const link = document.createElement("a");
    link.href = url;
    link.setAttribute("download", fileUrl.split("/").pop());
    document.body.appendChild(link);
```

```
    link.click();
    link.remove();
    window.URL.revokeObjectURL(url);
  } catch (error) {
    console.error("Download error:", error);
  }
};

const handlePrint = () => {
  const doc = new jsPDF();
  doc.text("Annual Report", 105, 10, { align: "center" });
  if (data.faculty.length)
    doc.autoTable({ head: [["ID", "Name",
    "Specialization"]], body: data.faculty.map
    (f => [f.id, f.faculty_name, f.specialization]) });
};
return (
  <div className="app-container">
    <nav className="navbar">
      <div className="navbar-left">
        <button className="menu-btn"><Menu /></button>
        <span className="navbar-title">A.R.G</span>
      </div>
      <div className="navbar-right">
```

```
    <button className="nav-icon" onClick={handlePrint}>
    <Printer /></button>
    <button className="nav-icon" data-bs-toggle="modal"
    data-bs-target="#uploadModal"><Upload /></button>
    <button className="nav-icon" onClick={()
    => setIsFilesOpen(!isFilesOpen)}><Eye /></button>
    <Power className="nav-icon" />
  </div>
</nav>

{isFilesOpen && (
  <div className="notifications-dropdown">
    <h3>Uploaded Files</h3>
    <ul>
      {data.files.length ? data.files.map((file, index)
      => (
        <li key={index} className="file-item">
          <span>{file.name}</span>
          <button onClick={() => handleDownload
          (file.url)} className="download-btn">
          <Download size={18} /></button>
        </li>
      )) : <li>No files uploaded</li>}
    </ul>
```

```
  </div>
)}

<aside className="sidebar">
  <ul className="sidebar-menu">
    <li><Link to="/hodlay/dashhod"><Home /> Dashboard
    </Link></li>
    <li><Link to="/hodlay/submission">Submission</Link>
    </li>
  </ul>
</aside>

<div className="modal fade" id="uploadModal"
tabIndex="-1" role="dialog">
  <div className="modal-dialog" role="document">
    <div className="modal-content">
      <div className="modal-header">
        <h5 className="modal-title">File Upload</h5>
        <button type="button" className="btn-close"
        data-bs-dismiss="modal"></button>
      </div>
      <div className="modal-body">
        <button className="btn btn-primary" onClick={()
        => fileRef.current.click()}>Select File</button>
```

```
                <input type="file" ref={fileRef} style=
                {{ display:"none"}} onChange={handleFileChange}/>
                <p>{file ? `Selected: ${file.name}` : "No file
                selected"}</p>
              </div>
              <div className="modal-footer">
                <button className="btn btn-secondary" data-bs-
                dismiss="modal">Close</button>
                <button className="btn btn-primary" onClick=
                {uploadFile}>Upload</button>
              </div>
            </div>
          </div>
        </div>
        <div className="content">
          <Outlet />
        </div>
      </div>);}
```

4. Code Implementation for Faculty Dashboard Layout

```
export default function Layout() {
  const navigate = useNavigate();
  const [theme, setTheme] = useState("light");
```

```
const [menuState, setMenuState] = useState
({ publications: false, ipr: false, seminar: false });
const [files, setFiles] = useState([]);
const [isNotificationsOpen, setIsNotificationsOpen]
= useState(false);
const [notificationCount, setNotificationCount]
= useState(0);
const notificationRef = useRef(null);


useEffect(() => {
  setTheme(localStorage.getItem("theme") || "light");
  document.body.className = theme;
  fetchFiles();
}, [theme]);


const fetchFiles = async () => {
  try {
    const { data } = await axios.get("http://localhost:
    5000/files");
    setFiles(data);
  } catch (error) {
    console.error("Error fetching files:", error);
  }
};
```

```
const toggleMenu = (key) => setMenuState((prev) =>
({ ...prev, [key]: !prev[key] }));
const toggleNotifications = async () => {
  setIsNotificationsOpen(!isNotificationsOpen);
  if (!isNotificationsOpen) await resetNotifications();
};
const resetNotifications = async () => {
  try {
    await axios.post("http://localhost:5000/reset-
    notifications");
    setNotificationCount(0);
  } catch (error) {
    console.error("Failed to reset notifications", error);
  }
};
const handleThemeChange = (selectedTheme) => {
  setTheme(selectedTheme);
  localStorage.setItem("theme", selectedTheme);
  document.body.className = selectedTheme;
};
const handleLogout = () => window.confirm("Do you really
want to log out?") && navigate("/");
const handleDownload = async (fileUrl) => {
```

```
  try {
    const { data } = await axios.get(`http://localhost:
    5000${fileUrl}`, { responseType: "blob" });
    const url = window.URL.createObjectURL(new Blob([data]));
    const link = document.createElement("a");
    link.href = url;
    link.setAttribute("download", fileUrl.split("/").pop());
    document.body.appendChild(link);
    link.click();
    link.remove();
  } catch (error) {
    console.error("Download error:", error);
  }
};

return (
  <div className={`app-container ${theme}`}>
    <nav className="navbar">
      <div className="navbar-left">
        <button className="menu-btn"><Menu /></button>
        <span className="navbar-title">A.R.G</span>
      </div>
      <div className="navbar-right">
        <button className="nav-icon" onClick=
```

```
{toggleNotifications}><Bell /></button>
{isNotificationsOpen && (
  <div className="notifications-dropdown" ref=
  {notificationRef}>
    <h3>Uploaded Files</h3>
    <ul>{files.length ? files.map((file, index) => (
      <li key={index} className="file-item">
        <span>{file.name.split("_").slice(1).join
        ("_")}</span>
        <button onClick={() => handleDownload
        (file.url)} className="download-btn">
        <Download size={18} className="download-icon"/>
        </button>
      </li>
    )) : <li>No files uploaded</li>}</ul>
  </div>
)}
<button className="nav-icon" onClick={() =>
toggleMenu("settings")}><Settings /></button>
{menuState.settings && (
  <div className="settings-dropdown">
    <div className="dropdown-item" onClick={() =>
    handleThemeChange("light")}>Light Mode</div>
    <div className="dropdown-item" onClick={() =>
```

```
          handleThemeChange("dark")}>Dark Mode</div>
        </div>
      )}
      <div className="nav-icon" onClick={handleLogout}>
      <Power /></div>
    </div>
  </nav>

  <aside className="sidebar">
    <ul className="sidebar-menu">
      {[{ to: "/dashboard", label: "Dashboard" }, { to:
      "/faculty", label: "Submission" }].map((item) => (
        <li key={item.to}><NavLink to={item.to}>
        {item.label}</NavLink></li>
      ))}
      {[["publications", "Publications"],
      ["ipr", "IPR Items"], ["seminar", "Seminar/Event"]].map
      (([key, label]) => (
        <li key={key} onClick={() => toggleMenu(key)}
        className="menu-item">
          {label} {menuState[key] ? <ChevronUp /> :
          <ChevronDown />}
          {menuState[key] && <ul className="submenu">
            {["Option 1", "Option 2", "Option 3"].map
```

```
                ((sub, i) => (
                  <li key={i}><NavLink to={`/${key}${i}`}>{sub}
                  </NavLink></li>
                ))}
              </ul>}
            </li>
          ))}
        </ul>
      </aside>
      <div className="content"><Outlet /></div>
      <footer className="footer"><p>&copy; 2024
      Automated Report Generation. All rights reserved.</p>
      </footer>
    </div>
    );
}
```

5. Logic behind Form Submission

```
const Submission = () => {
  const [data, setData] = useState([]);
  const [selectedFaculty, setSelectedFaculty] = useState(null);
  const [relatedData, setRelatedData] = useState([]);

  const fetchData = async () => {
    try {
      const response = await axios.get("http://localhost:
      5000/faculty-submissions");
      setData(response.data);
    } catch (error) {
      console.error("Error fetching data:", error);
    }
  };

  const handleView = async (faculty_id) => {
    setSelectedFaculty(faculty_id === selectedFaculty ? null
    : faculty_id);
    if (faculty_id !== selectedFaculty) {
      const response = await axios.get(`http://localhost:
      5000/faculty-details/${faculty_id}`);
```

```
    setRelatedData(response.data || []);
  }
};

const handleStatusChange = async (id, newStatus) => {
  try {
    await axios.put(`http://localhost:5000
    /faculty-form/${id}/status`, { status: newStatus });
    fetchData();
  } catch (error) {
    console.error("Error updating status:", error);
  }
};

return (
  <div>
    <h2>Confirmation Table</h2>
    <button onClick={fetchData}>Fetch Data</button>
    <table>
      <thead>...</thead>
      <tbody>
        {data.map(({ id, faculty_name, specialization,... })
        => (
          <tr key={id}>
```

```
            <td>{id}</td><td>{faculty_name}</td>...
            <td><button onClick={() => handleView(faculty_id)}>
            View</button></td>
            <td>
              {status === "Pending" && (
                <>
                  <button onClick={() => handleStatusChange
                  (id, "Approved")}>Approve</button>
                  <button onClick={() => handleStatusChange
                  (id, "Rejected")}>Reject</button>
                </>
              )}
            </td>
          </tr>
        ))}
      </tbody>
    </table>
    {selectedFaculty && (
      <div>
        <h3>Details for Faculty: {selectedFaculty}</h3>
        <table>
          <thead>
            <tr><th>#</th><th>Faculty ID</th><th>Author</th>...
          </thead>
```

```
            <tbody>
              {relatedData.length ? relatedData.map((book,
              index) => (
                <tr key={index}>
                  <td>{index + 1}</td><td>{book.faculty_id}
                  </td>...
                </tr>
              )) : <tr><td colSpan="7">No Records Found</td>
              </tr>}
            </tbody>
          </table>
        </div>
      )}
    </div>
  );
};
```

5.3.2 Backend Logic

1. **API Calls**

```
const express = require("express");

const cors = require("cors");

const bodyParser = require("body-parser");


const app = express();

const PORT = 5000;


// Import all route modules

const PatentsBackend = require("./PatentsBackend");

const LoginBackend = require("./LoginBackend");


// Middleware

app.use(cors());

app.use(bodyParser.json());


// Register all routes dynamically

app.use(require(`./${route}`)));


// Start server

app.listen(PORT, () => console.log(`Server running on

port ${PORT}`));
```

2. Form Submission

```
//Example Form for Published Books
//Submit Form
router.post("/books", async (req, res) => {
  const { faculty_id, author, title, publisher, year, isbn }
  = req.body;

  try {
    const [result] = await pool.query(
      "INSERT INTO books (faculty_id, author, title, publisher,
      year, isbn) VALUES (?, ?, ?, ?, ?, ?)",
      [faculty_id, author, title, publisher, year, isbn]
    );
    res.status(201).json({ message: "Book created", id:
    result.insertId });
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Failed to create book" });
  }
});

// Delete a book by title
router.delete("/books/delete", async (req, res) => {
  const { title } = req.body;
```

```
  try {
    const [result] = await pool.query("DELETE FROM
    books WHERE title = ?", [
      title,
    ]);
    res.status(result.affectedRows > 0 ? 200 : 404).json({
      message: result.affectedRows > 0 ? "Book deleted" :
      "Book not found",
    });
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Failed to delete book" });
  }
});

// Get all books to display
router.get("/books", async (req, res) => {
  try {
    const [books] = await pool.query("SELECT * FROM books");
    res.status(200).json(books);
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Failed to fetch books" });
  }
```

```
});
```

3. Collection of Faculty Submissions

```
router.get("/faculty-submissions", async (req, res) => {
  try {
    const [rows] = await pool.query("SELECT * FROM
    faculty_submissions");
    res.status(200).json(rows);
  } catch (error) {
    console.error("Error fetching faculty data:", error);
    res.status(500).json({ error: "Failed to fetch data" });
  }
});

router.put("/faculty-form/:id/status", async (req, res) => {
  try {
    const { id } = req.params;
    const { status } = req.body;

    if (!["Pending", "Approved", "Rejected"].includes(status)) {
      return res.status(400).json({ error: "Invalid status
      value" });
    }
```

```
    const [result] = await pool.query(
      "UPDATE faculty_submissions SET status = ? WHERE id = ?",
      [status, id]
    );

    if (result.affectedRows === 0) {
      return res.status(404).json({ error: "Faculty submission
      not found" });
    }

    res.status(200).json({ message: "Status updated
    successfully!" });
  } catch (error) {
    console.error("Error updating faculty status:", error);
    res.status(500).json({ error: "Failed to update status" });
  }
});
```

4. Report Printing Logic

```
// Route to fetch all faculty submissions
router.get("/faculty", async (req, res) => {
  try {
    const [rows] = await pool.query("SELECT * FROM faculty_
    submissions");
    res.status(200).json(rows);
```

```
  } catch (error) {
    console.error("Error fetching faculty submissions:", error);
    res.status(500).json({ error: "Failed to fetch faculty
    submissions" });
  }});
```

5. Final Submission Logic

```
router.get("/faculty-details/:faculty_id",async(req, res)=>{
  try {
    const { faculty_id } = req.params;
    const result = await pool.query(
      "SELECT * FROM books WHERE faculty_id = ?",
      [faculty_id]
    );
    if (result.length === 1) {
      return res.status(404).json({error:"No records found"});
    }

    return res.status(200).json(result[0]);
  } catch (error) {
    console.error("Error fetching faculty data:", error);
    return res.status(500).json({error:"Failed to fetch data"});
  }
});
```

5.4 Testing & Deployment

- API Testing: Postman used to test API endpoints.

6 Conclusion and Future scope

6.1 Conclusion

The Automated Department Annual Report System successfully addresses the challenges associated with manual report generation, data entry errors, and time-consuming compilation processes. By integrating a centralized database, secure faculty login system, and automated report generation, the system ensures accuracy, efficiency, and accessibility.

The project has significantly reduced the time and effort required for compiling faculty-related information, allowing for real-time updates and structured report formatting. With its user-friendly web interface, faculty members can easily upload and manage their academic records, while department administrators can generate customized, print-ready reports in minutes.

Overall, the system has successfully achieved its objectives and can be further scaled and enhanced to accommodate additional features based on institutional needs.

6.2 Future Scope

Although the system has successfully streamlined the report generation process, several enhancements can be made to improve its functionality and usability:

- Department Expansion
  - Can incorporate different departments.
  - Role Management at university level.
- Authentication
  - Can Add Google Authentication for login authentication.
- Storage Expansion
  - Can use cloud storage instead of local storage.

- Automated Notifications & Reminders
  - Enable email/SMS notifications to remind faculty about report submission deadlines.
  - Provide real-time updates on newly uploaded documents.
- Data Visualization & Analytics
  - Integrate charts, graphs, and statistical insights to analyze faculty contributions.
- Advanced Report Customization
  - Allow users to generate custom reports based on selected filters (e.g., research projects, patents, publications).
  - Provide multi-format export options (Word, Excel, PDF).

7 References

- Bootstrap: https://getbootstrap.com/
- Youtube: https://www.youtube.com/
- MySQL: https://www.mysql.com/
- React: https://react.dev/