Project Report

Project Title:

SecureAuth Hub – Registration and Login System with Dashboard

---

1. Introduction

In modern web applications, **secure user authentication** and **real-time data display** are essential. This project implements a **full-stack registration and login system** with a **dashboard** that shows data from a PostgreSQL database. It demonstrates proper use of **DBMS concepts**, secure authentication practices, and frontend-backend integration.

Technology Stack:

- **Frontend:** React.js
- **Backend:** ASP.NET Core Web API
- **Database:** PostgreSQL

The system ensures **user registration**, **login/logout**, **session protection**, and **dashboard-based database visualization**.

---

2. Problem Statement

After user login, the system must securely retrieve data from the PostgreSQL database via the ASP.NET Core backend and display it dynamically in the React dashboard. Unauthorized users must be prevented from accessing the dashboard.

Challenges addressed:

- Secure password storage
- Session-based authorization
- Dynamic data rendering from the database
- Full-stack integration

---

3. Objectives

1. Implement **user registration** with validation and unique email constraint.
2. Implement **secure login/logout** with hashed password verification.
3. Create a **dashboard** to display database information after login.

4. Use **PostgreSQL** to store and retrieve user data.
5. Demonstrate **full-stack data flow** from database → backend → frontend.

Optional features:

- JWT-based authentication
- Protected API endpoints
- Clean, user-friendly interface

---

4. System Design

4.1 Database Design (PostgreSQL)

Table: users

Field	Type	Constraint
id	SERIAL	PRIMARY KEY
name	VARCHAR(100)	NOT NULL
email	VARCHAR(150)	UNIQUE, NOT NULL
password	TEXT	NOT NULL
phone	VARCHAR(20)	NOT NULL
gender	VARCHAR(10)	NOT NULL

DBMS Concepts Implemented:

- Primary key
- Unique constraints
- Data integrity and normalization

4.2 Backend Design (ASP.NET Core)

- **Controllers:** AuthController, DashboardController
- **Models:** User
- **Database Context:** AppDbContext (Entity Framework Core)
- **Authentication:** JWT + [Authorize] attributes

- **APIs:**
    - /api/auth/register → Register new users
    - /api/auth/login → Login and get token
    - /api/dashboard → Get user list and total count

4.3 Frontend Design (React)

- **Pages:** Register, Login, Dashboard
- **Routing:** React Router
- **API Service:** Axios with token interceptor
- **Dashboard Features:**
    - Total users count
    - List of all registered users
    - Logout button

---

5. Methodology

Step 1: Database Setup

- Create PostgreSQL database secureauth_db
- Create table users with necessary fields

Step 2: Backend Setup

- ASP.NET Core Web API project
- Entity Framework Core for database operations
- Password hashing with BCrypt
- JWT for authentication
- APIs for registration, login, and dashboard data retrieval

Step 3: Frontend Setup

- React project with pages: Register, Login, Dashboard
- Axios to call backend APIs
- Token stored in localStorage for session persistence

- Protected routes for dashboard

Step 4: Integration

- Register → Database insert
- Login → JWT token generation
- Dashboard → Fetch data from API and render

---

6. Results

1. **Registration Module**
    - Validates input fields
    - Ensures email uniqueness
    - Passwords hashed before storing
2. **Login Module**
    - Verifies credentials
    - Returns JWT token
    - Invalid login shows error message
3. **Dashboard Module**
    - Displays **total users**
    - Displays **list of users**
    - Protected access; unauthorized users cannot access

---

7. Conclusion

This project successfully demonstrates a **secure full-stack authentication system** with real-time **database connectivity**. The **React frontend** interacts with **ASP.NET Core backend APIs**, which fetch data from **PostgreSQL**. The system includes **registration, login/logout, session management**, and a **dashboard** that displays database data dynamically, satisfying all DBMS principles and project requirements.

---

8. Future Enhancements

- Add **search, filter, and pagination** in dashboard

- Implement **role-based access** (Admin/User)
- Add **password reset** functionality
- Improve **UI/UX with modern design frameworks**

If you want, I can also **create a fully formatted Word/PDF version** with **diagrams (ER diagram, system architecture)** and **screenshots**, ready to submit.

Do you want me to do that?

not problem report. i need my project report

Got it 👍
You do **NOT** want only a "problem report".
You want a FULL PROJECT REPORT for your completed project, ready to **submit**.

Below is a complete Project Report, written in simple, formal academic English, exactly matching what teachers expect.

You can **copy–paste this** into Word / PDF.

PROJECT REPORT

Project Title

Registration and Login System with Dashboard Using React, ASP.NET Core, and PostgreSQL

---

1. Introduction

In modern web-based systems, user authentication and secure data management are essential components. This project focuses on designing and implementing a **Registration and Login System** integrated with a **dashboard** that displays data retrieved from a relational database.

The system is developed using **React.js** for the frontend, **ASP.NET Core Web API** for the backend, and **PostgreSQL** as the database. It demonstrates proper use of **Database Management System (DBMS) concepts**, including database connectivity, data storage, querying, and secure access control.

---

2. Project Objectives

The main objectives of this project are:

1. To design and implement a secure **user registration system**.
2. To authenticate users using **email and password**.
3. To store user data securely in a **PostgreSQL database**.
4. To restrict dashboard access only to **authenticated users**.
5. To retrieve and display database data dynamically on the dashboard.
6. To demonstrate **frontend–backend–database integration**.

---

3. Tools and Technologies Used

Frontend

- React.js
- JSX
- Axios
- React Router

Backend

- ASP.NET Core Web API
- Entity Framework Core
- JWT (JSON Web Token)
- BCrypt for password hashing

Database

- PostgreSQL

---

4. System Architecture

The system follows a **three-tier architecture**:

1. **Presentation Layer (React Frontend)**
   Handles user interface, form input, and data display.
2. **Application Layer (ASP.NET Core Backend)**
   Handles business logic, authentication, authorization, and database queries.
3. **Data Layer (PostgreSQL Database)**
   Stores user information securely with proper constraints.

---

5. Database Design

Users Table

Field Name	Data Type	Description
id	SERIAL	Primary key
name	VARCHAR(100)	User full name
email	VARCHAR(150)	Unique email address
password	TEXT	Encrypted password
phone	VARCHAR(20)	Phone number
gender	VARCHAR(10)	Gender

Database Constraints Used:

- Primary Key
- Unique constraint on email
- NOT NULL constraints

These constraints ensure **data integrity and normalization**.

---

6. Functional Modules

6.1 Registration Module

The registration module allows new users to create an account by entering:

- Full Name
- Email Address
- Password
- Phone Number
- Gender

Features:

- Input validation
- Unique email verification

- Password encryption using BCrypt
- Secure storage in PostgreSQL database

6.2 Login Module

The login module authenticates registered users using:

- Email Address
- Password

Features:

- Credential verification from database
- Encrypted password matching
- JWT token generation
- Error message for invalid login
- Session handling using token

6.3 Dashboard Module

After successful login:

- The user is redirected to the dashboard
- The dashboard fetches data from the backend API
- Database data is displayed dynamically

Displayed Data:

- Total number of registered users
- List of registered users (Name, Email, Phone, Gender)

The dashboard is protected and cannot be accessed without login.

---

7. Security Implementation

- Passwords are stored in **hashed format**
- JWT-based authentication is used
- Backend APIs are protected using [Authorize]

- Dashboard access is restricted
- Logout clears authentication token

---

8. Working Procedure

1. User registers → data stored in PostgreSQL
2. User logs in → backend verifies credentials
3. JWT token generated and sent to frontend
4. Frontend stores token and accesses dashboard
5. Dashboard requests data from backend
6. Backend retrieves data using database queries
7. Data is displayed in the dashboard

---

9. Results and Output

- Registration works successfully
- Login authentication works correctly
- Dashboard displays real-time database data
- Unauthorized access is blocked
- System demonstrates proper DBMS usage

---

10. Conclusion

This project successfully implements a **secure full-stack registration and login system** with a **dashboard** using React, ASP.NET Core, and PostgreSQL. It demonstrates real-world application of **DBMS concepts**, secure authentication, and dynamic data retrieval. The system meets all the functional and security requirements defined at the beginning of the project.

---

11. Future Enhancements

- Role-based access (Admin/User)
- Search and pagination in dashboard

- Password reset functionality
- Improved UI design
- Audit logs and activity tracking