Addwise Tech Login System:

What I Built

I developed a full-stack authentication system for Addwise Tech with a secure, modern UI and robust backend logic. The system supports user registration, login, password reset via OTP, Google authentication, and protected page access using tokens.

Technologies Used

Frontend:

- React (with Vite)
- Bootstrap 5 (for styling)
- React Router DOM (for routing)
- React Context API (for authentication state)
- @react-oauth/google (for Google login)
- jwt-decode (to decode Google token)

Backend:

- Node.js
- Express.js
- MongoDB (with Mongoose ODM)
- JSON Web Token (JWT) for authentication
- Bcrypt (for hashing passwords)
- Nodemailer (for sending OTPs via email)

Tools & Environment:

- Vite (for fast frontend development)
- Axios (for HTTP requests)

Features Implemented

- ➢ User Registration with hashed password storage (bcrypt)
- ➢ Login with form validation
- ➢ Dashboard after login displaying user info (protected route)

- Google Login using OAuth and token decoding
- Password Reset with OTP emailed to user
- Protected Routes using JWT middleware
- Auth Context for global login/logout state
- Clean Bootstrap UI for all pages

Project Implementation Procedure

1. Backend Development

Step 1: Created an Express server and connected it to MongoDB using Mongoose.

Step 2: Set up middleware including cors, express.json(), and route handling.

Step 3: Created a User model with name, email, password, and role fields.

Step 4: On user registration:

Used bcrypt to hash passwords before saving them to MongoDB.

Step 5: On login:

Compared entered password with hashed one using bcrypt.compare().

If valid, generated a JWT token and sent it to the client.

Step 6: Created JWT middleware to protect routes.

Step 7: Set up Nodemailer to send OTPs to users for password reset.

Step 8: Created routes for:

- Register
- Login
- Get current user details
- Forgot password (send OTP)
- Reset password
- Update user profile

2. Frontend Development

Step 1: Created a new Vite + React project in Cursor AI.

Step 2: Installed packages:

bootstrap, axios, react-router-dom, react-toastify, @react-oauth/google, jwt-decode

Step 3: Set up Bootstrap by importing it in main.jsx.

Step 4: Created project folder structure: pages/, components/, context/, etc.

Step 5: Implemented AuthContext.jsx using React Context API to manage login/logout and store JWT in localStorage.

Step 6: Created a ProtectedRoute.jsx component to restrict page access for unauthenticated users.

Step 7: Set up React Router with routes for:

/ → Login

/signup → Signup

/dashboard → Dashboard (protected)

/edit-profile → Edit profile (protected)

/forgot-password → Forgot password

/reset-password/:token → Reset password

Step 8: Built individual pages using Bootstrap:

- LoginPage.jsx: with email/password and Google login using @react-oauth/google

- Register.jsx: collects name, email, and password

- Dashboard.jsx: displays user info after login

- EditProfile.jsx: allows updating user info
- ForgotPassword.jsx: sends OTP to email
- ResetPassword.jsx: sets new password using token

3. Authentication Logic

- Used JWT token for route protection (stored in localStorage)
- Used axios interceptors (optional) for attaching token to requests
- Displayed success/error messages using react-toastify

4. Testing & Validation

- Tested full user flow:
- Register → Login → Dashboard → Logout
- Forgot Password → Email OTP → Reset Password
- Google Login
- Profile Update
- Verified JWT protection works by blocking access to protected pages without a token.

Project Flow

1. Register/Login with email & password (passwords are hashed before storing).
2. On login, a JWT is issued and stored in localStorage.
3. The user is redirected to a protected Dashboard page showing their information.
4. If password is forgotten, the user can request an OTP to their email.

5. The user uses the OTP link to access the Reset Password page securely.

6. Google Login is available as an alternative to email login.

7. All protected pages use a custom ProtectedRoute component to check JWT validity.

Authentication Flowchart

This flowchart represents a **basic user authentication process** used in most login systems. It outlines the logical steps involved in verifying a user's identity based on their username and password.

How It Works:

1. **Start:**
   The process begins when a user attempts to log in to a system or application.

2. **Login Screen:**
   The user is presented with a login screen where they must enter their **username** and **password**.

3. **Click Submit:**
   Once the user fills in the credentials, they click on the submit/login button to proceed with authentication.

4. **Username Validation:**
   The system first checks whether the entered username exists in the database.

   - If the username is **invalid**, the system displays an **"Invalid Username"** error message, and the process ends there.
   - If the username is **valid**, the system proceeds to password verification.

5. **Password Validation:**
   The system then checks whether the provided password matches the one stored for that username.

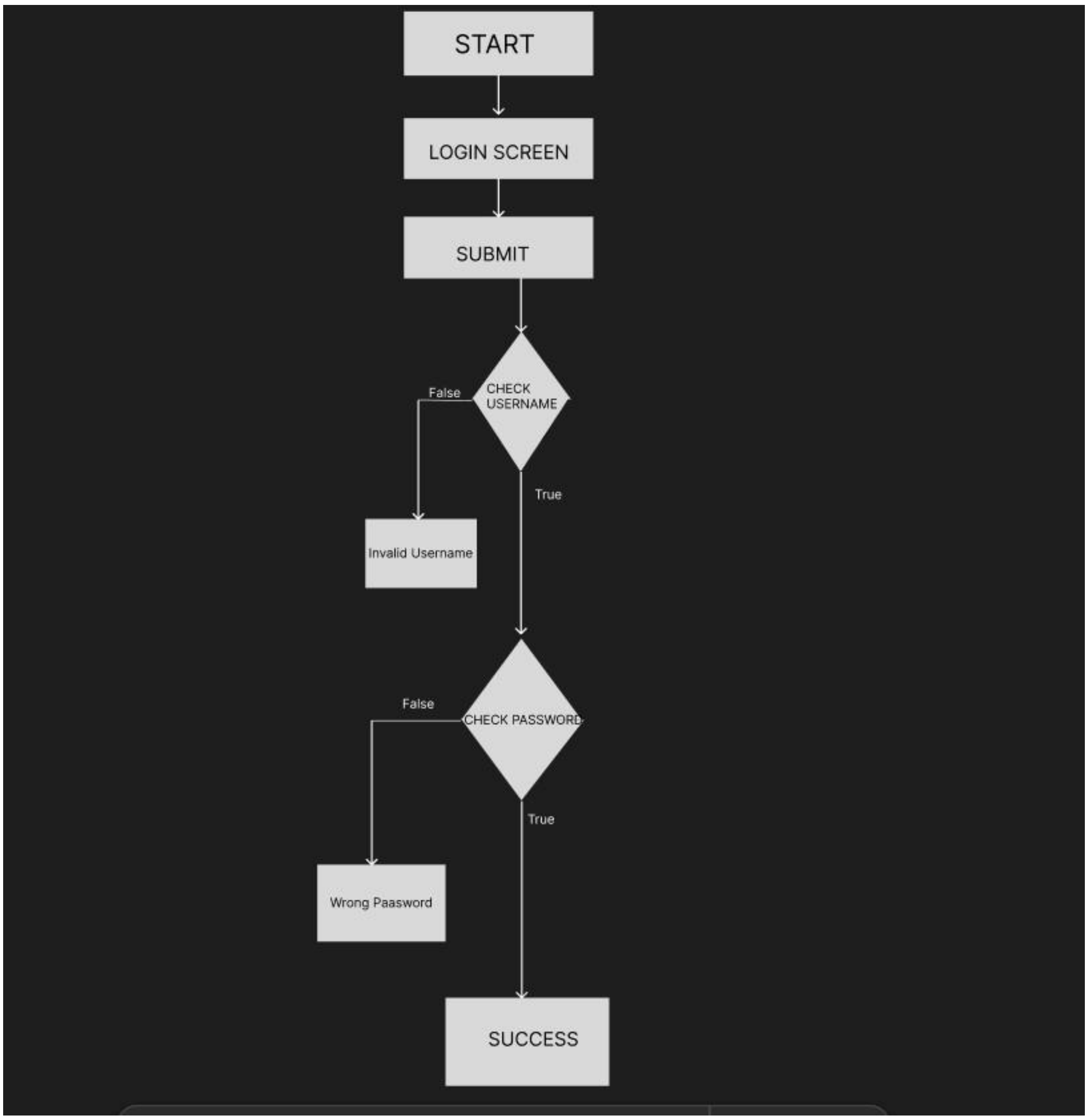
   - If the password is **incorrect**, an **"Incorrect Password"** error message is shown.
   - If the password is **correct**, the user is successfully authenticated.

6. **Login Successful:**

Upon successful username and password validation, the user gains access to the system or is redirected to their dashboard or homepage.

Outcome:

The flowchart ensures a secure, step-by-step validation process that protects user data by checking both the username and password before granting access.



Conclusion

This project successfully demonstrates the development of a secure and user-friendly authentication system using the MERN (MongoDB, Express, React, Node.js) stack, enhanced with Bootstrap for responsive UI and Cursor AI for accelerated development.

All major functionalities were implemented, including user registration, login, secure JWT-based route protection, Google sign-in, password reset via OTP, and a post-login dashboard displaying user information. The system ensures both usability and security, with hashed password storage and token-based authentication.

This project lays a strong foundation for real-world applications that require user authentication. Future enhancements like role-based access control, admin dashboard, and activity logs can further expand its capabilities.