



SRI KRISHNA COLLEGE OF TECHNOLOGY
An Autonomous Institution | Accredited by NAAC with 'A' Grade
Affiliated to Anna University | Approved by AICTE
KOVAIPUDUR, COIMBATORE 641042



SECURE AUTHENTICATION API

A PROJECT REPORT

Submitted by

SHERWIN V EVAN

727821TUCS224

*in partial fulfillment for the award of the
degree of*

**BACHELOR OF
ENGINEERING IN
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

June 2023



SRI KRISHNA COLLEGE OF TECHNOLOGY
An Autonomous Institution | Accredited by NAAC with 'A' Grade
Affiliated to Anna University | Approved by AICTE
KOVAIPUDUR, COIMBATORE 641042



BONAFIDE CERTIFICATE

Certified that this project report “**SECURE AUTHENTICATION API**” is the bonafide work of “**SHERWIN V EVAN**” who carried out the project work under my supervision.

SIGNATURE

Mrs.S.PRIYADARSHINI

SKILL DEVELOPMENT ENGINEER
IAMNEO
Coimbatore-641004

SIGNATURE

Dr.R.VIDHYA

HEAD OF THE DEPARTMENT

Associate Professor,
Department of Computer
Science and Engineering
Sri Krishna College of
Technology,
Coimbatore-641042.

Certified that the candidates were examined by us in the Project Work Viva Voce examination held on _____ at Sri Krishna College of Technology, Kovaipudur, Coimbatore -641042

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

First and foremost, we thank the **Almighty** for being our light and for showering his gracious blessings throughout the course of this project.

We are grateful to our beloved Principal **Dr.M.G. Sumithra M.E., Ph.D.** for her tireless and relentless support.

We extend our sincere thanks to our Head of the Department **Dr.R.Vidhya M.Tech., Ph.D.** for her encouragement and inspiration.

We are greatly indebted to our Industry Mentor **Mrs. C VIJITHA** for her valuable guidance and suggestions in all aspects that aided us to ameliorate our skills.

We are thankful to all those who have directly and indirectly extended their help to us in completing this project work successfully.

ABSTRACT

ABSTRACT

The objective of this project is to develop a comprehensive and secure Login/Signup and Authentication API using Spring Boot, MySQL, and ReactJS. The project focuses on creating a robust, scalable, and user-friendly system that enables users to register, log in, and authenticate their identities securely. The backend implementation relies on Spring Boot, a powerful framework known for its versatility in handling user management, authentication, and authorization. Spring Security is utilized to incorporate essential security measures, including password hashing, session management, and role-based access control. To ensure secure data storage and retrieval, MySQL, a reliable relational database management system, is employed. User information, including login credentials, undergoes encryption and other security measures to safeguard sensitive data. On the frontend, ReactJS is utilized to create an intuitive and interactive user interface. The signup and login forms are designed with user experience in mind, featuring real-time validation and error handling. Seamless communication and authentication processes are established through integration with the backend API. Throughout the project's development lifecycle, utmost attention is given to implementing industry-standard security practices, effectively mitigating common threats such as cross-site scripting (XSS), cross-site request forgery (CSRF), and SQL injection. The system is designed with scalability in mind, allowing for future enhancements and accommodating a growing user base. By implementing this project, users will benefit from a reliable, secure, and user-friendly authentication system. The project serves as a solid foundation, empowering developers to customize and tailor the solution according to specific application requirements.

TABLE OF CONTENTS

TABLE OF CONTENT

CHAPTER.NO	TITLE	PAGE NO.
	ABSTRACT	iv
	LIST OF FIGURES	vi
	LIST OF ABBREVIATIONS	vii
1	INTRODUCTION	1
	1.2 OVERVIEW	2
	1.3 OBJECTIVE	2
2	LITERATURE SURVEY	3
3	SYSTEM SPECIFICATIONS	5
4	PROPOSED SYSTEM	8
	4.1 PROPOSED SYSTEM	8
	4.2 ADVANTAGES	9
5	METHODOLOGIES	10
	5.1 LOGIN PAGE	11
	5.2 SIGN UP PAGE	12
	5.3 RESET PASSWORD	13
	5.4 ACCOUNT MANAGEMENT	13
6	IMPLEMENTATION AND RESULT	18
7	CONCLUSION AND FUTURE SCOPE	43
	7.1 CONCLUSION	44
	7.2 FUTURE SCOPE	45
8	REFERENCES	47

LIST OF FIGURES

LIST OF FIGURES

Figure No	TITLE	Page No
5.1	Process flow diagram	10
5.2	Login page flowchart	11
5.3	Signup page flowchart	12
5.4	Password page flowchart	15
6.1	Login page	15
6.2	Signup page	16
6.3	My Account page	17

LIST OF ABBREVIATIONS

LIST OF ABBREVIATIONS

ABBREVIATIONS

ACRONYMS

HTML

HYPertext MARKUP LANGUAGE

CSS

CASCADING STYLESHEET

JS

JAVASCRIPT

DOM

DOCUMENT OBJECT MODEL

UI

USER INTERFACE

INTRODUCTION

CHAPTER 1

INTRODUCTION

In today's digital landscape, user authentication and security are paramount concerns for any web application or system. Building a robust, scalable, and secure Login/Signup and Authentication API is crucial to safeguarding user data and ensuring a seamless user experience. This project aims to address these needs by utilizing the power of Spring Boot, MySQL, and ReactJS to create an advanced authentication system.

The project focuses on the development of a comprehensive API that enables users to securely register, log in, and authenticate their identities. By leveraging the capabilities of Spring Boot, a popular Java-based framework, the backend implementation provides a solid foundation for user management, authentication, and authorization. Spring Security, a key component of Spring Boot, ensures the implementation of essential security measures such as password hashing, session management, and role-based access control.

On the frontend, ReactJS is employed to create an intuitive and interactive user interface. The signup and login forms are designed to provide a seamless experience for users, with real-time validation and error handling. The integration of ReactJS with the backend API facilitates secure communication and authentication processes.

Throughout the development of this project, a strong emphasis is placed on implementing robust security practices. Measures are taken to address common vulnerabilities such as cross-site scripting (XSS), cross-site request forgery (CSRF), and SQL injection. By adhering to these security best practices, the system ensures the integrity and confidentiality of user data. Furthermore, the project is designed with scalability in mind, allowing for future enhancements and the accommodation of a growing user base. By providing a solid foundation for authentication, developers can build upon this project to suit their specific requirements.

.1 PROBLEM STATEMENT

Designing and implementing a secure and efficient Login/Signup and Authentication API that ensures user data protection and seamless user authentication experience.

1.2 OVERVIEW

This project aims to develop a comprehensive Login/Signup and Authentication API using Spring Boot, MySQL, and ReactJS. It focuses on creating a secure and scalable system for user registration, login, and authentication processes. The project emphasizes robust security measures, seamless user experience, and the integration of backend and frontend technologies.

1.3 OBJECTIVE

The objective of this project is to create a highly secure, scalable, and user-friendly Login/Signup and Authentication API using Spring Boot, MySQL, and ReactJS. The primary goal is to implement a robust authentication system that ensures the protection of user data while providing a seamless login and signup experience. Additionally, the project aims to incorporate industry-standard security measures, such as password hashing and encryption, to mitigate potential threats and vulnerabilities.

LITERATURE SURVEY

CHAPTER 2

LITERATURE SURVEY

2.1 METHDOLOGY

Design the architecture of the system, including the backend and frontend components. Define the data models, API endpoints, and integration points with Spring Boot, MySQL, and ReactJS. Incorporate industry-standard security practices and authentication mechanisms.

2.2 KEY FINDINGS

Authentication Methods: Identify the most suitable authentication methods for your project, such as OAuth, JSON Web Tokens (JWT), or Security Assertion Markup Language (SAML). Compare their security, ease of implementation, and compatibility with your chosen technologies.

Security Best Practices: Explore industry-standard security practices for user authentication and protection of sensitive data. This includes robust password hashing algorithms, secure session management, and measures to prevent common vulnerabilities like cross-site scripting (XSS) and SQL injection.

Scalability Considerations: Investigate techniques and approaches for building a scalable authentication system. Examine load balancing, caching mechanisms, and database optimization strategies to ensure the system can handle increasing user traffic and data volume.

User Experience: Analyze user experience considerations for login/signup forms, including usability, error handling, and validation. Explore design patterns and techniques to create a seamless and intuitive authentication process for users.

Performance Optimization: Look into performance optimization techniques to ensure fast

response times and efficient data retrieval. Consider database indexing, query optimization, and caching mechanisms to improve system performance.

Integration and Compatibility: Explore the integration aspects of your chosen technologies, such as Spring Boot, MySQL, and ReactJS. Investigate compatibility challenges, libraries or frameworks that facilitate integration, and best practices for establishing seamless communication between backend and frontend components.

2.3 SURVEY CONCLUSION

Firstly, it is crucial to implement robust security measures such as encryption, secure password hashing, and protection against common vulnerabilities like cross-site scripting (XSS) and SQL injection. Implementing authentication methods like OAuth, JSON Web Tokens (JWT), or Security Assertion Markup Language (SAML) can enhance security and user experience.

Secondly, scalability considerations should be taken into account by implementing load balancing, caching mechanisms, and optimizing database performance to handle increasing user traffic and data volume effectively.

Additionally, providing a seamless user experience through well-designed login/signup forms, efficient error handling, and validation is paramount for user satisfaction. Integration of technologies like Spring Boot, MySQL, and ReactJS should be carefully implemented to ensure compatibility and smooth communication between the backend and frontend components.

SYSTEM SPECIFICATION

SYSTEM SPECIFICATION

In this chapter, we are gonna see the softwares that we have used to build the website. This chapter gives you a small description about the softwares used in the project.

3.1 VS CODE

Visual Studio Code is a source code editor developed by Microsoft for Windows, Linux, and macOS. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is also customizable, so users can change the editor's theme, keyboard shortcuts, and preferences.

VS Code is an excellent code editor for React projects. It is lightweight, customizable, and has a wide range of features that make it ideal for React development. It has built-in support for JavaScript, JSX, and TypeScript, and enables developers to quickly move between files and view detailed type definitions. It also has a built-in terminal for running tasks. Additionally, VS Code has an extensive library of extensions that allow developers to quickly add features like code snippets, debugging tools, and linting support to their projects.

3.2.REACT

React is a JavaScript library created by Facebook for building user interfaces. It is a component-based, declarative, and highly efficient library that is used to develop interactive UIs (user interfaces) for single page web applications. React uses a virtual DOM (Document Object Model) that makes it faster and easier to manipulate the DOM

elements. It also provides declarative components that allow developers to write code that is easy to read and maintain. React also offers an extensive library of tools and components that make it easier to develop complex user interfaces.

3.3 ROUTERS IN REACT

Routers are important components in React applications. They provide the ability to navigate between different views or components of the application. React Router is the most popular library to handle routing in React applications. It provides the ability to define routes, set up links, and render components based on the current route. It also provides features like data fetching, code-splitting, and server-side rendering.

3.4 HTTP-ONLY COOKIES

HTTP-only cookies are a crucial security feature used in web applications to mitigate the risk of cross-site scripting (XSS) attacks and enhance the protection of user session data.

HTTP-only cookies are cookies that can only be accessed and modified by the server through HTTP or HTTPS requests. They are designed to prevent client-side scripts, such as JavaScript, from accessing the cookie values. By restricting access to cookies, HTTP-only cookies effectively mitigate the risk of malicious scripts attempting to steal sensitive information, such as session identifiers or authentication tokens.

The main advantage of using HTTP-only cookies is that they provide an additional layer of protection against XSS attacks. XSS attacks occur when an attacker injects malicious scripts into a web application, which are then executed by unsuspecting users' browsers. If an attacker manages to inject a script that attempts to read or manipulate cookies, the presence of HTTP-only cookies will prevent the attacker from accessing the

sensitive cookie data.

By setting the HTTP-only flag in the cookie's attributes, web developers ensure that the cookie is inaccessible to client-side scripts. This flag is enforced by modern web browsers, preventing JavaScript code from accessing or modifying the cookie values. However, the browser still includes the HTTP-only cookies in subsequent HTTP or HTTPS requests, allowing the server to maintain session information and authenticate the user.

HTTP-only cookies are widely used in combination with secure communication protocols like HTTPS to enhance the overall security of web applications. When used alongside secure cookies (cookies that are also transmitted over HTTPS), HTTP-only cookies significantly reduce the risk of session hijacking and other security vulnerabilities.

In conclusion, HTTP-only cookies play a vital role in web application security by protecting sensitive cookie data from being accessed or manipulated by client-side scripts. By using HTTP-only cookies, web developers can mitigate the risk of cross-site scripting attacks and ensure the integrity and confidentiality of user session information.

PROPOSED SYSTEM

CHAPTER 4

PROPOSED SYSTEM

This chapter gives a small description about the proposed idea behind the development of our website

4.1 PROPOSED SYSTEM

The system will leverage the Spring Boot framework, known for its ease of development and powerful capabilities, to handle user management, authentication, and authorization. Spring Security, a key component of Spring Boot, will be utilized to implement essential security measures such as password hashing, session management, and role-based access control. This will ensure that user data is protected and only accessible to authorized individuals.

To ensure secure storage and retrieval of user information, the system will employ MySQL, a reliable and widely used relational database management system. User credentials and other sensitive data will be encrypted to safeguard against unauthorized access.

4.2 ADVANTAGES

Enhanced Security: The project prioritizes security by incorporating industry-standard practices such as password hashing, encryption, and protection against common vulnerabilities. The use of Spring Security ensures robust authentication and authorization mechanisms, reducing the risk of unauthorized access and data breaches.

Scalability and Flexibility: The project is designed to be scalable, allowing it to handle increasing user traffic and growing data volume. With the use of Spring Boot and ReactJS, developers have the flexibility to add new features, optimize performance, and adapt the system to evolving business needs.

User-Friendly Experience: The integration of ReactJS on the frontend enables the creation of a modern and intuitive user interface. Real-time validation, error handling, and seamless communication with the backend API contribute to a user-friendly login/signup experience.

Seamless Integration: The combination of Spring Boot, MySQL, and ReactJS ensures seamless integration between the backend and frontend components. This streamlines the development process and enables efficient communication, reducing potential compatibility issues.

Industry-Standard Technologies: Spring Boot, MySQL, and ReactJS are widely adopted and supported technologies with active developer communities. This means that developers can leverage extensive documentation, tutorials, and resources to facilitate development, troubleshooting, and future maintenance.

Robust Documentation and Community Support: The Spring Boot, MySQL, and ReactJS ecosystems offer comprehensive documentation and active communities. Developers can benefit from extensive resources, forums, and community support to address challenges, seek guidance, and stay updated with best practices.

METHODOLOGIES

CHAPTER 5

METHODOLOGIES

This chapter gives a small description about how our system works.

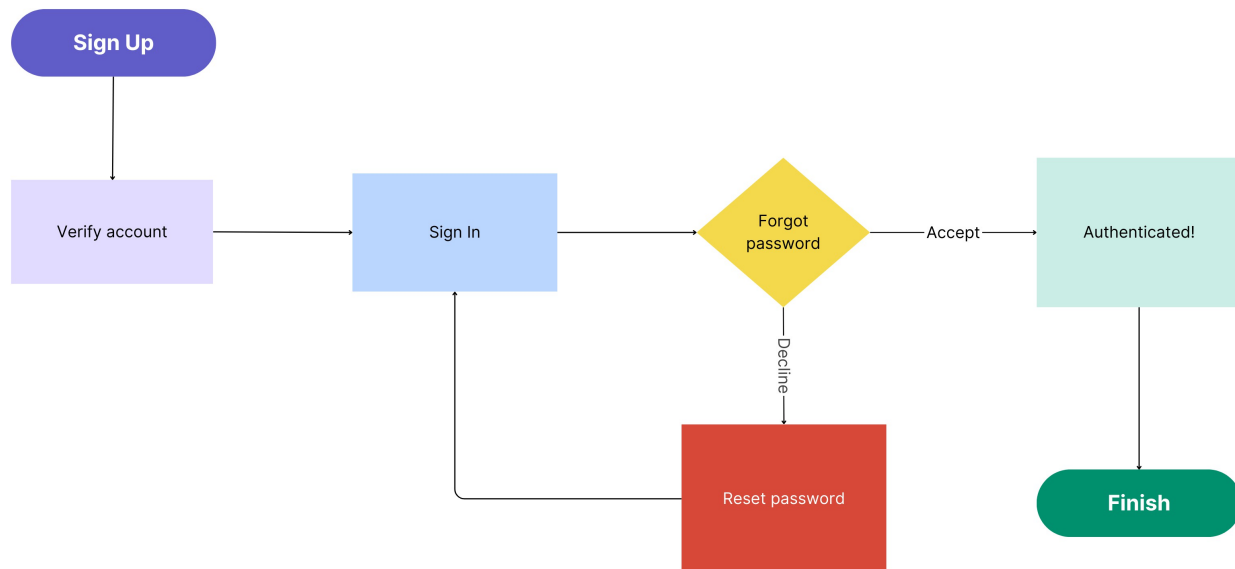


Fig 5.1 PROCESS FLOW DIAGRAM

5.1.LOGIN

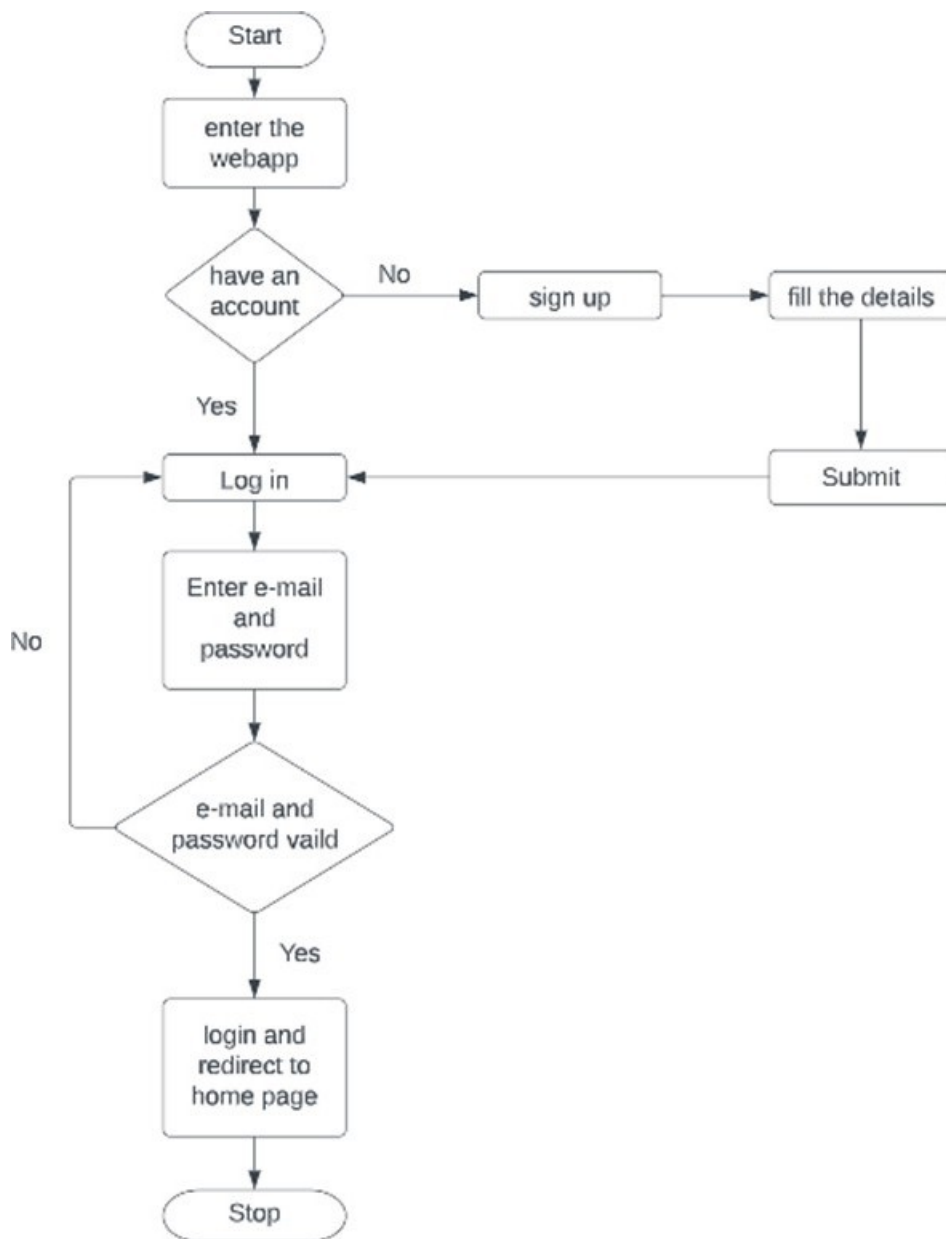


Fig 5.2. LOGIN PAGE FLOWCHART

In this page we will be asking about the username and password of the user. Firstly, the website validates the user inputs. It verifies the username and password by checking it with the usernames and passwords stored in the local storage when the user creates an account in the website.

5.2 SIGNUP PAGE

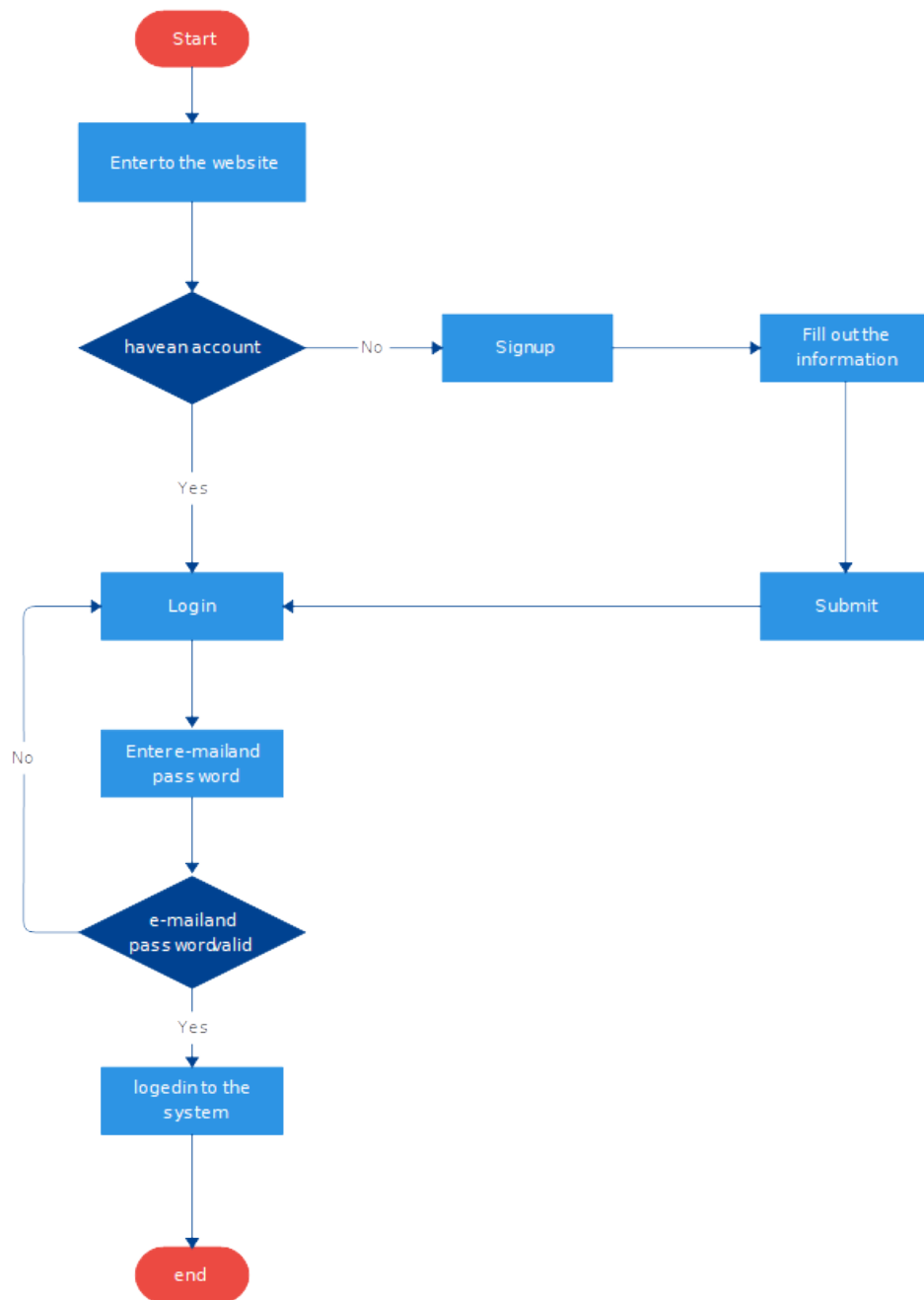


Fig 5.3 SIGNUP PAGE FLOWCHART

This page asks users about the basic details of the user to create an account. This page asks for details like username, password, email id, phone number. After the user enters the details, these details are then validated by our code. If all details are correct then the user is then directed to the login page.

5.3 FORGOT PASSWORD

If an existing user forgets the password he can access a component named forgot password where he will be directed for a page that's requesting their email id, after entering the existing email id the user will be redirected to the email OTP verification, then the user can change their password.

5.4 RESET PASSWORD PAGE

Request for Password Reset: The user initiates the password reset process by requesting a password reset. This can be done by clicking on a "Forgot Password" link or button on the login page. The system verifies the user's identity through an email address or username.

Verification and Security Checks: To ensure the security of the password reset process, the system may implement additional verification steps. This may include sending a verification code or link to the user's registered email address or utilizing multi-factor authentication (MFA) to confirm the user's identity.

Password Reset Token Generation: Once the user's identity is verified, the system generates a unique password reset token. This token is typically a random string or a hashed value associated with the user's account. It serves as a secure mechanism for identifying and validating the password reset request.

Sending the Password Reset Link: The system sends an email containing the password reset link or token to the user's registered email address. The email should include clear instructions and a link/button to proceed with the password reset.

User Confirmation and Password Reset Form: Upon receiving the password reset email, the user clicks on the provided link to access the password reset form. The link should include

the password reset token to validate the user's request. The system verifies the token and presents the user with a form to enter a new password.

New Password Submission: The user enters a new password in the password reset form and submits it. The system validates and securely stores the new password, ensuring it meets the required security criteria such as complexity and length.

Password Reset Confirmation: Once the new password is successfully submitted, the system confirms the password reset and notifies the user through a confirmation message. The user can then proceed to log in using the newly set password.

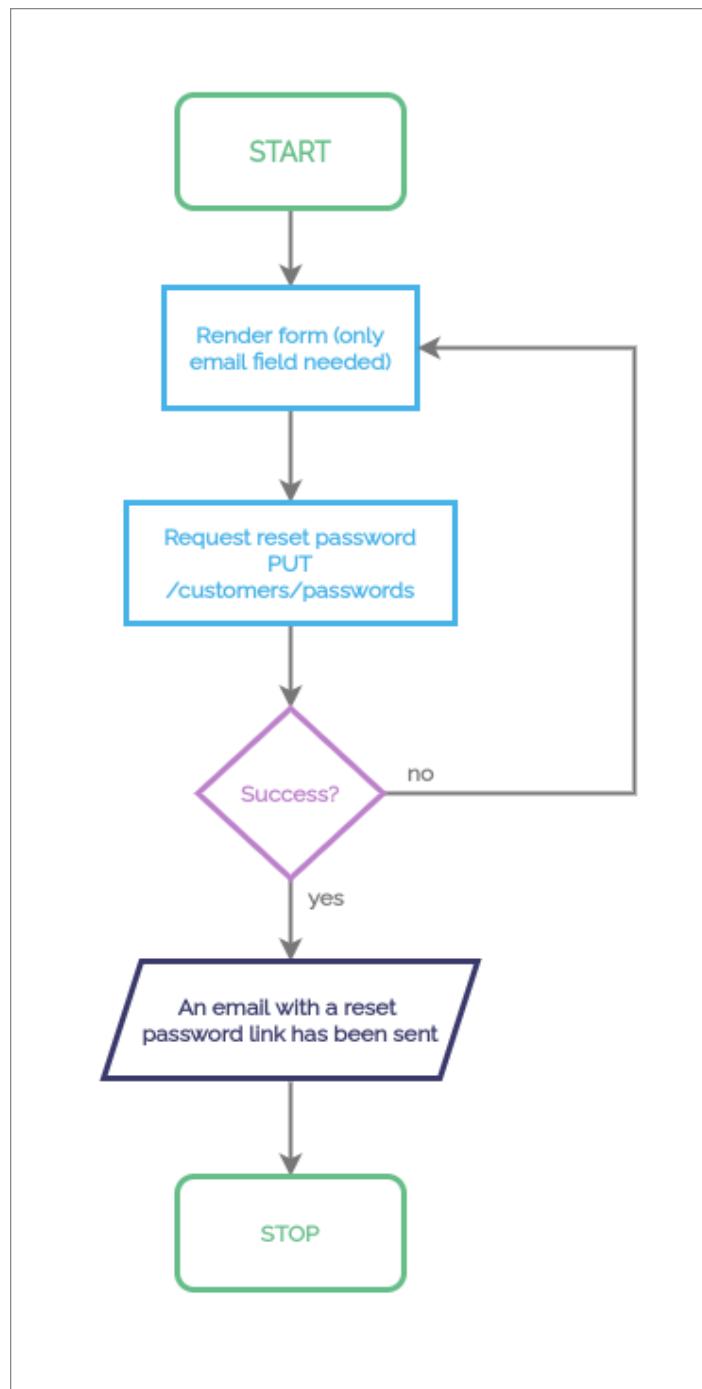


Fig 5.4 PASSWORD PAGE FLOWCHART

5.5 ACCOUNT MANAGEMENT PAGE

An account management page is a central hub within a web application where users can view and manage their account settings and preferences. It provides a convenient and user-friendly interface for users to have control over their account-related information. Here are some key aspects and functionalities typically found in an account management page:

1. **Personal Information:** The account management page displays the user's personal information, such as name, email address, and profile picture. Users can update and modify their personal details, ensuring that their account information is accurate and up to date.

2. **Password Management:** This section allows users to change their account password. It typically includes fields for the current password, new password, and password confirmation. Users can update their passwords periodically or in case of security concerns, ensuring the integrity and confidentiality of their account.

3. **Email Preferences:** Users can customize their email notification preferences through the account management page. They can choose to receive notifications for various activities, such as account updates, promotions, or newsletters. This allows users to tailor their email preferences according to their interests and requirements.

4. **Privacy and Security Settings:** The account management page includes options for managing privacy and security settings. Users can configure settings related to account visibility, two-factor authentication (2FA), account recovery options, and other security measures. This empowers users to maintain control over their account's privacy

and security aspects.

5. **Linked Accounts and Social Media Integration:** In cases where the web application allows social media integration, users may have the option to connect their accounts with social media platforms. The account management page provides the ability to manage and disconnect linked social media accounts if needed.

6. **Billing and Subscription Management:** If the web application offers paid services or subscriptions, the account management page may include a section for managing billing and subscription details. Users can view their payment history, update payment methods, and manage their subscription plans from this section.

7. **Account Deactivation or Closure:** Some account management pages include options for users to deactivate or close their accounts. This involves providing clear instructions and confirmation prompts to ensure that users are aware of the consequences of their actions.

The account management page plays a vital role in providing users with control over their account-related settings and preferences. By offering a comprehensive and user-friendly interface, it enhances the overall user experience, promotes user engagement, and fosters a sense of ownership and customization within the web application.

IMPLEMENTATION AND RESULT

CHAPTER 6

IMPLEMENTATION AND RESULT

This chapter gives a description about the output that we produced by developing the website of our idea.

6.1 LOGIN

When User enters our website he will be asked about his login details like email id and password. The login details will be verified with the details given while the user creates an account.

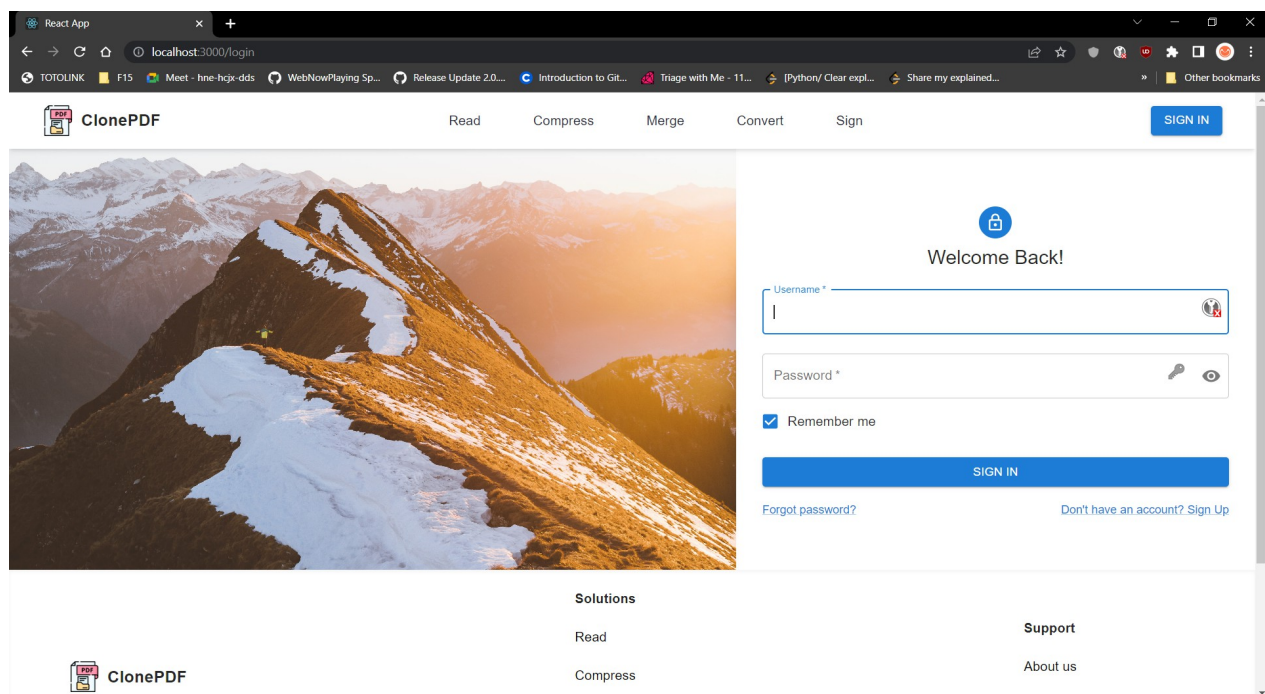
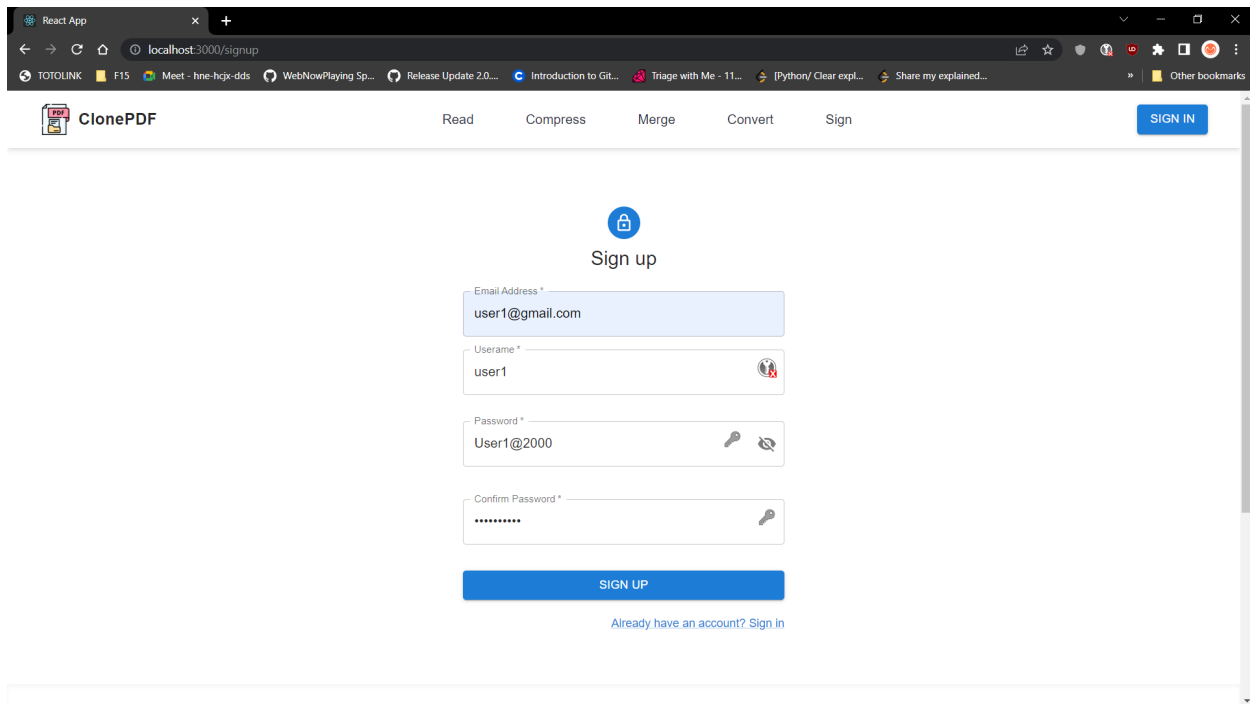


Fig 6.1 LOGIN PAGE

6.2 SIGN UP

If a user doesn't have an account on the website, User can use a component named create new account available in the login page. When the user clicks on that he will be redirected to the signup page. In sign up he should fill up his email id, password and phone number. These inputs will be validated.



The screenshot displays a web browser window with the address bar showing 'localhost:3000/signup'. The page features a dark header with the 'ClonePDF' logo and navigation links: 'Read', 'Compress', 'Merge', 'Convert', 'Sign', and a blue 'SIGN IN' button. The main content area is titled 'Sign up' with a blue lock icon. It contains four input fields: 'Email Address *' with the value 'user1@gmail.com', 'Username *' with the value 'user1', 'Password *' with the value 'User1@2000', and 'Confirm Password *' with masked characters. Below the fields is a blue 'SIGN UP' button and a link 'Already have an account? Sign in'.

Fig 6.2 SIGN UP PAGE

6.3 ACCOUNT MANAGEMENT PAGE

The account management page provides authenticated users with a centralized platform to manage their account settings and preferences. Upon authentication, users are presented with essential account information such as their username and email address, ensuring they have a clear understanding of their account's identity.

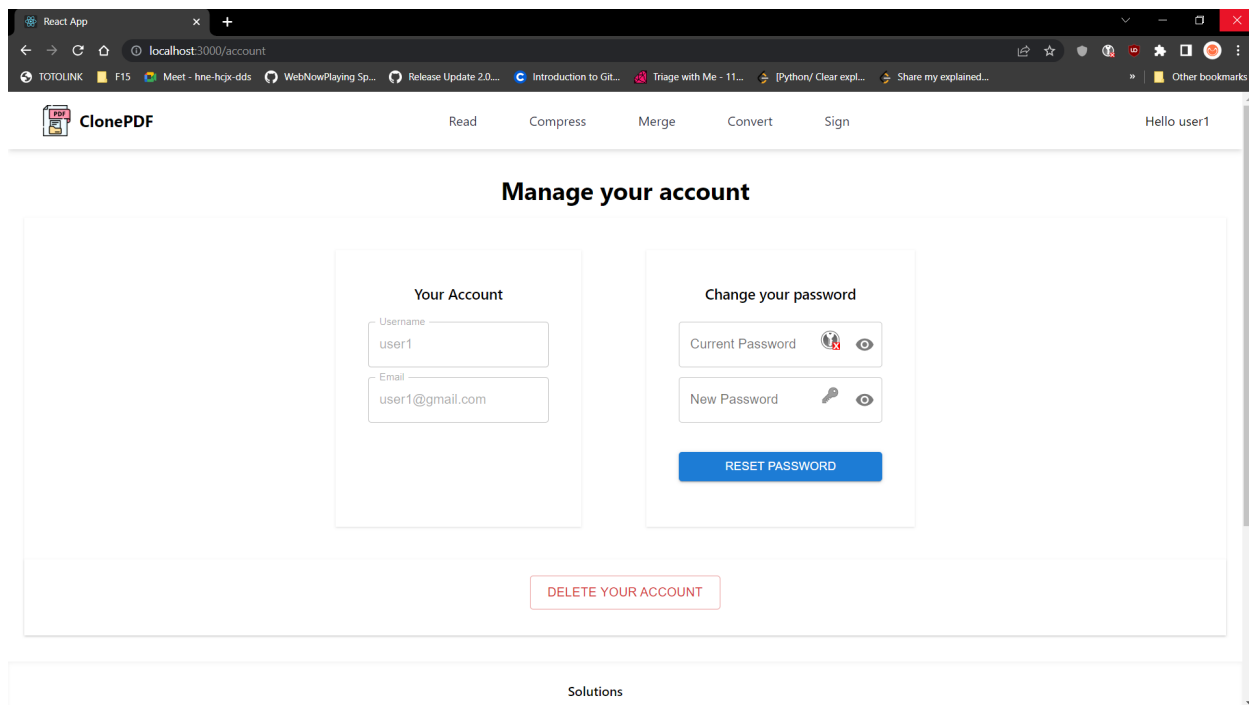


Fig 6.3 MY ACCOUNT PAGE

CODING

Login:

```
export default function Login() {
  React.useEffect(DeleteSessionCookie);

  const { handleSubmit, register } = useForm();

  const [showPassword, setShowPassword] = React.useState(false);

  const handleTogglePassword = () => {
    setShowPassword(!showPassword);
  };

  const onSubmit = async (data) => {
    console.log(data);

    if (
      !/^[a-zA-Z][a-zA-Z0-9_]{3,15}$/.test(data.userName) ||
      !/^(?=.*[A-Z])(?=.*[a-z])(?=.*[!@#$%^&*()_+\-=\[\]{};':"\\|,.<\/?])(?=.*\d){8,127}$/.test(
        data.password
      )
    ) {
      toast.error(<div>Invalid Username/Password!</div>, {
        position: "top-center",
        autoClose: 2500,
        hideProgressBar: false,
        closeOnClick: true,
        pauseOnHover: true,
        draggable: true,
        progress: undefined,
        theme: "light",
      });
    } else {
      try {
        const response = await axios.post("login/", data);
        await toast.promise(Promise.resolve(response), {
          pending: "Authenticating...",
          success: "Login successful!",
          error: "Invalid Username/Password!",
          position: "top-center",
        });
      } catch {
        toast.error("Invalid Username/Password!");
      }
    }
  };
}
```



```

        autoClose: 2750,
        hideProgressBar: false,
        closeOnClick: true,
        pauseOnHover: true,
        draggable: true,
        progress: undefined,
        theme: "light",
    });

    if (response.status === 200) {
        saveRememberMePreference(data.remember);
        setTimeout(() => (window.location = "/"), 3000);
    }
} catch (error) {
    console.log(error);
    toast.error(
        <div>
            Server error! <br /> Please try again later.
        </div>,
        {
            position: "top-center",
            autoClose: 3000,
            hideProgressBar: false,
            closeOnClick: true,
            pauseOnHover: true,
            draggable: true,
            progress: undefined,
            theme: "light",
        }
    );
}
}
};

return (
    <
        <NavBar />
        <ThemeProvider theme={theme}>
        <Grid container component="main" sx={{ height: "100%" }}>
        <CssBaseline />
        <Grid
            item
            xs={false}
            sm={4}

```

```

md={7}
sx={{
  backgroundImage: `url(${loginBG})`,
  backgroundRepeat: "no-repeat",
  backgroundColor: (t) =>
    t.palette.mode === "light"
      ? t.palette.grey[50]
      : t.palette.grey[900],
  backgroundSize: "cover",
  backgroundPosition: "center",
}}
/>
<Grid
  item
  xs={12}
  sm={8}
  md={5}
  component={Paper}
  elevation={0}
  square
>
  <Box
    sx={{
      my: 8,
      mx: 4,
      display: "flex",
      flexDirection: "column",
      alignItems: "center",
    }}
  >
    <Avatar sx={{ m: 1, bgcolor: "primary.main" }}>
      <LockOutlinedIcon />
    </Avatar>
    <Typography component="h1" variant="h5">
      Welcome Back!
    </Typography>
    <Box
      component="form"
      noValidate
      onSubmit={handleSubmit(onSubmit)}
      sx={{ mt: 1 }}
    >
      <TextField
        margin="normal"

```

```

    required
    fullWidth
    id="userName"
    label="Username"
    name="userName"
    autoComplete="userName"
    {...register("userName")}
    autoFocus
  />
<TextField
  margin="normal"
  required
  fullWidth
  name="password"
  label="Password"
  type={showPassword ? "text" : "password"}
  id="password"
  autoComplete="current-password"
  {...register("password")}
  InputProps={{
    endAdornment: (
      <IconButton onClick={handleTogglePassword} edge="end">
        {showPassword ? (
          <VisibilityOffIcon />
        ) : (
          <VisibilityIcon />
        )}
      </IconButton>
    ),
  }}
/>

<FormControlLabel
  control={
    <Checkbox
      {...register("remember")}
      color="primary"
      defaultChecked={true}
    />
  }
  label="Remember me"
/>
<Button
  type="submit"

```

```

        fullWidth
        variant="contained"
        sx={{ mt: 3, mb: 2 }}
    >
        Sign In
    </Button>
    <Grid container>
        <Grid item xs>
            <Link href="/forgotPassword" variant="body2">
                Forgot password?
            </Link>
        </Grid>
        <Grid item>
            <Link href="/signup" variant="body2">
                {"Don't have an account? Sign Up"}
            </Link>
        </Grid>
    </Grid>
</Box>
</Box>
</Grid>
</Grid>
</ThemeProvider>
<Footer />
</>
);
}

```

Sign up:

```

export default function SignUp() {
  const {
    control,
    handleSubmit,
    formState: { errors },
    getValues,
  } = useForm();

  React.useEffect(DeleteSessionCookie);

  const [showPassword, setShowPassword] = React.useState(false);

  const handleTogglePassword = () => {
    setShowPassword(!showPassword);
  };

  const validatePassword = (value) => {
    const uppercaseRegex = /[A-Z]/;
    const lowercaseRegex = /[a-z]/;
    const specialCharactersRegex = /[!@#$%^&*()_+\-=\[\]{};':"\"|,.<>/?]+/;
    const digitRegex = /\d/;

    if (!uppercaseRegex.test(value)) {
      return "At least one uppercase character is required.";
    }

    if (!specialCharactersRegex.test(value)) {
      return "At least one special character is required.";
    }

    if (!lowercaseRegex.test(value)) {
      return "At least one lowercase character is required.";
    }

    if (!digitRegex.test(value)) {
      return "At least one digit is required.";
    }
  };

```

```

    return true;
};

const onSubmit = (data) => {
  console.log(data);

  axios
    .post("auth/signup", data)
    .then((res) => {
      if (res.status === 200) {
        toast.success(
          <div>
            Signed up successfully! <br /> Please login to continue.
          </div>,
          {
            position: "top-center",
            autoClose: 3000,
            hideProgressBar: false,
            closeOnClick: true,
            pauseOnHover: true,
            draggable: true,
            progress: undefined,
            theme: "light",
          }
        );

        setTimeout(() => (window.location = "/login"), 3000);
      }
    })
    .catch((error) => {
      toast.error(
        <div>
          Server error! <br /> Please try again later.
        </div>,
        {
          position: "top-center",
          autoClose: 3000,
          hideProgressBar: false,
          closeOnClick: true,
          pauseOnHover: true,
          draggable: true,
          progress: undefined,
          theme: "light",
        }
      );
    });

```

```

    );
  });
};

return (
  <>
    <NavBar />
    <div className="outerbox">
      <ThemeProvider theme={theme}>
        <Container component="main" maxWidth="xs">
          <CssBaseline />
          <Box
            className="innerBox"
            sx={{
              marginTop: 8,
              marginBottom: 8,
              display: "flex",
              flexDirection: "column",
              alignItems: "center",
            }}
          >
            <Avatar sx={{ m: 1, bgcolor: "primary.main" }}>
              <LockOutlinedIcon />
            </Avatar>
            <Typography component="h1" variant="h5">
              Sign up
            </Typography>
            <Box
              component="form"
              noValidate
              onSubmit={handleSubmit(onSubmit)}
              sx={{ mt: 3 }}
            >
              <Grid container spacing={2}>
                <Grid item xs={12}>
                  <Controller
                    name="email"
                    control={control}
                    rules={{
                      required: "Email is required.",
                      pattern: {
                        value:
                          //RFC2822 Email regex
                          /[a-z0-9!#$%&'*/+=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*/+=?^_`{|}~-]+)*@(?:[a-

```

```

z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?/,
    message: "Enter a valid email address.",
  },
}}
render=(({ field }) => (
  <TextField
    {...field}
    required
    fullWidth
    autoComplete="email"
    name="email"
    id="email"
    label="Email Address"
    error={!errors.email}
    helperText={errors.email?.message}
  />
))
/>
</Grid>
<Grid item xs={12}>
  <Controller
    name="userName"
    control={control}
    rules={{
      required: "Username is required.",
      pattern: {
        value: /^[a-zA-Z][a-zA-Z0-9_]{3,15}$/ ,
        message:
          "Enter a valid Username (Alphanumeric and _ character(s) are allowed).",
      },
      minLength: {
        value: 4,
        message: "Username must be at least 4 characters.",
      },
      maxLength: {
        value: 16,
        message: "Username can be at atmost 16 characters.",
      },
    }}
  render=(({ field }) => (
    <TextField
      {...field}
      required
      fullWidth

```



```

      autoComplete="given-name"
      name="userName"
      id="userName"
      label="Username"
      error={!errors.userName}
      helperText={errors.userName?.message}
    />
  ))
/>
</Grid>
<Grid item xs={12}>
  <Controller
    name="password"
    control={control}
    rules={{
      required: "Password is required.",
      minLength: {
        value: 8,
        message: "Password must be at least 8 characters.",
      },
      validate: validatePassword,
    }}
    render={({ field }) => (
      <TextField
        {...field}
        margin="normal"
        required
        fullWidth
        name="password"
        label="Password"
        id="password"
        autoComplete="new-password"
        type={showPassword ? "text" : "password"}
        error={!errors.password}
        helperText={errors.password?.message}
        InputProps={{
          endAdornment: (
            <IconButton
              onClick={handleTogglePassword}
              edge="end"
            >
              {showPassword ? (
                <VisibilityOffIcon />
              ) : (

```

```

        <VisibilityIcon />
    })
</IconButton>

),
}}
/>
))
/>
</Grid>
<Grid item xs={12}>
    <Controller
        name="confirmPassword"
        control={control}
        rules={{
            required: "Confirm Password is required.",
            validate: {
                matchesPassword: (value) =>
                    value === getValues().password ||
                    "Passwords do not match.",
            },
        }}
        render={({ field }) => (
            <TextField
                {...field}
                margin="normal"
                required
                fullWidth
                name="confirmPassword"
                label="Confirm Password"
                id="confirmPassword"
                autoComplete="new-password"
                type="password"
                error={!!errors.confirmPassword}
                helperText={errors.confirmPassword?.message}
            />
        ))
    />
</Grid>
</Grid>
<Button
    type="submit"
    fullWidth
    variant="contained"
    sx={{ mt: 3, mb: 2 }}

```

```

    >
      Sign Up
    </Button>
    <Grid container justifyContent="flex-end">
      <Grid item>
        <Link href="/Login" variant="body2">
          Already have an account? Sign in
        </Link>
      </Grid>
    </Grid>
  </Box>
</Box>
</Container>
</ThemeProvider>
</div>
<Footer />
</>
);
}

```

My account page:

```

export default function MyAccount() {

  useRememberMe();

  const {
    handleSubmit: handleResetPasswordSubmit,
    formState: { errors: resetPasswordErrors },
    getValues: getResetPasswordValues,
    reset: resetResetPasswordForm,
    control: resetPasswordControl,
  } = useForm();

  const {
    handleSubmit: handleDeleteAccountSubmit,
    formState: { errors: deleteAccountErrors },
    reset: resetDeleteAccountForm,
    control: deleteAccountControl,
  } = useForm();

  const [currUser, setCurrUser] = useState("Guest");

```

```

const [email, setEmail] = useState("NoEmail");
const [showCurrPassword, setShowCurrPassword] = useState(false);
const [showNewPassword, setShowNewPassword] = useState(false);
const [modalOpen, setModalOpen] = useState(false);

```

```

const handleModalOpen = () => setModalOpen(true);
const handleModalClose = () => setModalOpen(false);

```

```

const modalStyle = {
  position: "absolute",
  top: "40%",
  left: "50%",
  transform: "translate(-50%, -50%)",
  width: 400,
  bgcolor: "background.paper",
  border: "2px solid #000",
  boxShadow: 24,
  p: 4,
};

```

```

useEffect(() => {
  async function checkUserStatus() {
    const user = await GetUser();
    setCurrUser(user);
    getEmail();
    if (user === "Guest" || user === "Error") {
      toast.error(<div>You are not logged in!</div>, {
        position: "top-center",
        autoClose: 2500,
        hideProgressBar: false,
        closeOnClick: true,
        pauseOnHover: true,
        draggable: true,
        progress: undefined,
        theme: "light",
      });
      setTimeout(() => (window.location = "/login"), 2250);
    }
  }
  checkUserStatus();
}, []);

```

```

const getEmail = async () => {
  const email = await axios.get("login/email");

```

```

    setEmail(email.data);
  };

  const handleToggleCurrPassword = () => {
    setShowCurrPassword(!showCurrPassword);
  };

  const handleToggleNewPassword = () => {
    setShowNewPassword(!showNewPassword);
  };

  const validatePassword = (value) => {
    const uppercaseRegex = /[A-Z]/;
    const lowercaseRegex = /[a-z]/;
    const specialCharactersRegex = /[!@#$%^&*()_+!-=[]{};':"\\|,.<>/?]+/;
    const digitRegex = /\d/;

    if (!uppercaseRegex.test(value)) {
      return "At least one uppercase character is required.";
    }

    if (!specialCharactersRegex.test(value)) {
      return "At least one special character is required.";
    }

    if (!lowercaseRegex.test(value)) {
      return "At least one lowercase character is required.";
    }

    if (!digitRegex.test(value)) {
      return "At least one digit is required.";
    }

    if (getResetPasswordValues("currentPassword") === value) {
      return "New password must be different.";
    }

    return true;
  };

  const onResetPasswordSubmit = async (data) => {
    console.log(data);

    if (

```

```

    !/^(?=.*[A-Z])(?=.*[a-z])(?=.*[!@#$%^&*()_+\\-=[]{};':"\\|,.< >/?])(?=.*\d).
{8,127}$/.test(
    data.currentPassword
)
) {
    toast.error(<div>Invalid current password!</div>, {
        position: "top-center",
        autoClose: 2500,
        hideProgressBar: false,
        closeOnClick: true,
        pauseOnHover: true,
        draggable: true,
        progress: undefined,
        theme: "light",
    });
} else {
    try {
        const formData = {
            userName: currUser,
            oldPassword: getResetPasswordValues("currentPassword"),
            newPassword: getResetPasswordValues("newPassword"),
        };

        console.log(formData);

        const changePasswordResponse = await axios.post(
            "/auth/changePassword",
            formData
        );
        await toast.promise(Promise.resolve(changePasswordResponse), {
            pending: "Authenticating...",
            success: changePasswordResponse.data,
            error: changePasswordResponse.data,
            position: "top-center",
            autoClose: 2750,
            hideProgressBar: false,
            closeOnClick: true,
            pauseOnHover: true,
            draggable: true,
            progress: undefined,
            theme: "light",
        });
    } catch (error) {
        console.log(error);
    }
}

```

```

    toast.error(
    <div>
      Server error! <br /> Please try again later.
    </div>,
    {
      position: "top-center",
      autoClose: 3000,
      hideProgressBar: false,
      closeOnClick: true,
      pauseOnHover: true,
      draggable: true,
      progress: undefined,
      theme: "light",
    }
  );
} finally {
  resetResetPasswordForm({
    currentPassword: "",
    newPassword: "",
  });
}
}
};

const validateUsername = (value) => {
  if (value === currUser) return true;
  return "Incorrect username.";
};

const onDeleteAccountSubmit = async (data) => {
  if (data.deleteUsername !== currUser) {
    return;
  }
  try {
    const deleteData = {
      user
    }

```

```

@RestController
@RequestMapping("/login")
public class LoginController {

    private final TokenService tokenService;

    private final UserRepo userRepo;

    private final AuthenticationManager
authenticationManager;

    public LoginController(TokenService tokenService,
AuthenticationManager authenticationManager, UserRepo
userRepo,
        PasswordEncoder passwordEncoder) {
        this.tokenService = tokenService;
        this.authenticationManager =
authenticationManager;
        this.userRepo = userRepo;
    }

    @PostMapping("/")
    public ResponseEntity<?> token(@RequestBody
UserModel loginUser, HttpServletResponse response) throws
Exception {

        UserEntity userEntity =
userRepo.findByUserName(loginUser.getUserName());
        if (userEntity.getUserName() != null) {
            if (!userEntity.isEnabled())
                return ResponseEntity.ok("Activate
your account!");

            Authentication authentication =
authenticationManager.authenticate(
                new
UsernamePasswordAuthenticationToken(userEntity.getUserNa
me(), userEntity.getPassword()));

            String accessToken =
tokenService.generateToken(authentication);

            Cookie cookie = new
Cookie("access_token", accessToken);

```



```

        cookie.setMaxAge(864000); // Set cookie to
expire in 1 Day
        cookie.setPath("/");
        cookie.setHttpOnly(true);
        response.addCookie(cookie);

        return ResponseEntity.ok("Login
successful");
    }
    return ResponseEntity.ok("Invalid Credentials!");
}

```

```

@GetMapping("/")
public String home(Principal principal) {

    Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
    if (authentication != null &&
authentication.isAuthenticated()) {
        // User is authenticated
        log.info("Authenticated!");
        String currentUser_name =
authentication.getName();
        return currentUser_name;
    } else {
        // User is not authenticated
        log.info("Not Authenticated!");
    }

    return "Not logged in!";
}

```

```

@GetMapping("/email")
public String email(Principal principal) {

    Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
    if (authentication != null &&
authentication.isAuthenticated()) {
        // User is authenticated
        log.info("Authenticated!");
        String currentUser_name =
authentication.getName();

```

```
        return
userRepo.findEmailByuserName(currentUserName);
    } else {
        // User is not authenticated
        log.info("Not Authenticated!");
    }

    return "Not logged in!";
}
}
```

Security Config:

```

@Configuration
@EnableWebSecurity
public class WebSecurityConfig {

    /*@Autowired
    private UserDetailsService userDetailsService;
    */

    private static RSAKey rsaKeys;

    private static final String[] WHITE_LIST_URL = {
        "/auth/**",
        "/login/**",
        "/logout/**"
    };

    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        return http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth
                .requestMatchers(WHITE_LIST_URL).permitAll()
                .anyRequest().authenticated()
            )
            .sessionManagement(session ->
                session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .exceptionHandling(ex -> ex
                .authenticationEntryPoint(new BearerTokenAuthenticationEntryPoint())
                .accessDeniedHandler(new BearerTokenAccessDeniedHandler())
            )
            .addFilterAfter(new CookieAuthorizationFilter(jwtDecoder(), new
                JwtAuthenticationConverter()), SecurityContextPersistenceFilter.class)
            .build();
    }

    @Bean
    PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder(11);
    }

    @Bean
    JWKSSource<SecurityContext> jwkSource() {

```

```

    rsaKeys = Jwks.generateRsa();
    JWKSet jwkSet = new JWKSet(rsaKeys);
    return (jwkSelector, securityContext) -> jwkSelector.select(jwkSet);
}

@Bean
JwtDecoder jwtDecoder() throws JOSEException {
return NimbusJwtDecoder.withPublicKey(rsaKeys.toRSAPublicKey()).build();
}

@Bean
JwtEncoder jwtEncoder(JWKSource<SecurityContext> jwks) {
return new NimbusJwtEncoder(jwks);
}

@Bean
AuthenticationManager authManager(UserDetailsService userDetailsService) {
    var authProvider = new DaoAuthenticationProvider();
    authProvider.setUserDetailsService(userDetailsService);
    return new ProviderManager(authProvider);
}

/*
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(userDetailsService).passwordEncoder(new
BCryptPasswordEncoder(11));
    }*/

@Bean
UserDetailsService user(UserRepo userRepository) {
    return username -> {
        Optional<UserEntity> userEntity =
Optional.ofNullable(userRepository.findByUserName(username));
        if (userEntity.isPresent()) {
            UserEntity user = userEntity.get();
            List<GrantedAuthority> authorities = Collections.singletonList(new
SimpleGrantedAuthority("ROLE_USER"));
            return User.builder()
                .username(user.getUserName())
                .password("{noop}" + user.getPassword())
                .authorities(authorities)
                .build();
        } else {
            throw new UsernameNotFoundException("UserEntity not found: " + username);
        }
    };
}
}

```

CONCLUSION

CHAPTER 7

CONCLUSION

This chapter tells about the conclusion that anyone can drive from the project and the learning we learnt by taking over this project.

7.1 CONCLUSION

In conclusion, the project report outlines the development of a Login/Signup and Authentication API built with Spring Boot, MySQL, and ReactJS. The project aims to provide a secure, scalable, and user-friendly solution for managing user authentication and account-related functionalities.

Through the utilization of Spring Boot, the project ensures robust backend functionality, including secure password hashing, session management, and role-based access control. Integration with MySQL enables secure storage and retrieval of user data, while ReactJS facilitates the creation of a modern and intuitive user interface.

The project's advantages lie in its emphasis on security, scalability, and customization. By incorporating industry-standard practices and technologies, it mitigates risks associated with vulnerabilities and provides a reliable authentication system. The system's scalability ensures it can handle increasing user traffic, while customization options enable developers to tailor it to specific application requirements.

In summary, this project report presents a comprehensive solution for implementing a Login/Signup and Authentication API. It not only ensures the security and integrity of user data but also provides a user-friendly experience. The report serves as a foundation for further development and customization, enabling the creation of robust and secure web applications.

7.2 FUTURE SCOPE

Two-Factor Authentication (2FA): Implementing 2FA would add an extra layer of security to the authentication process. Users could opt for additional verification methods such as SMS-based verification codes, biometrics, or authenticator apps to further protect their accounts.

1. Social Media Integration: Integrating the authentication system with popular social media platforms would provide users with the option to sign up or log in using their social media accounts. This feature would enhance user convenience and streamline the registration process.
2. Single Sign-On (SSO): Incorporating SSO capabilities would allow users to log in to multiple related applications or services using the same set of credentials. This would improve user experience by eliminating the need for multiple logins and enhancing interoperability between different systems.
3. Account Recovery and Password Reset Improvements: Enhancements to the account recovery process, such as alternate account verification methods (e.g., security questions, recovery email), would facilitate easier account recovery in case of forgotten passwords or compromised accounts.
4. Role-Based Access Control: Implementing a more granular access control system based on user roles would allow for differentiated permissions and access levels within the application. This would enable administrators to define specific privileges for different user roles, enhancing security and data protection.
5. Audit Logs and Security Monitoring: Incorporating an audit logging system and security monitoring features would enable the tracking and analysis of user activities and security events. This would provide valuable insights for detecting and preventing potential security breaches or suspicious activities.
6. Multi-factor Authentication (MFA): Going beyond 2FA, integrating additional authenti-

cation factors like fingerprint scanning, facial recognition, or hardware tokens could further enhance the security of user accounts.

7. Integration with External Identity Providers: Enabling integration with external identity providers (e.g., OAuth providers, OpenID Connect) would allow users to authenticate through third-party services, expanding the authentication options available to users.

Localization and Internationalization: Adding support for multiple languages and cultural preferences would make the application more accessible and user-friendly for a global user base.

8. Continuous Security Auditing and Updates: Regular security audits and updates to keep pace with evolving security threats and best practices are crucial for maintaining the project's security and ensuring the protection of user data.

By exploring these future scope areas, the project can evolve into a more comprehensive and feature-rich solution, catering to the evolving needs of users and enhancing the overall security and user experience.

REFERENCES

REFERENCES

- 1) W3Schools (www.w3schools.com): W3Schools is a widely used online learning platform that offers comprehensive HTML and CSS tutorials, references, and examples. It provides a structured curriculum, hands-on exercises, and a "Try it Yourself" feature that allows you to experiment with code directly in your browser.
- 2) WordPress (wordpress.com): A popular content management system that allows users to create their own blogs or websites. It provides customizable templates, writing tools, and the ability to share and engage with readers.
- 3) RESTful API Design: The website "RESTful API Design" (<https://restfulapi.net/>) provides a comprehensive guide to designing and implementing RESTful APIs.
- 4) FreeCodeCamp (www.freecodecamp.org): FreeCodeCamp offers a free and interactive learning platform with a dedicated section for HTML and CSS. Their curriculum includes hands-on coding exercises and projects to help you practice and apply what you learn.
- 5) Codecademy (www.codecademy.com): Codecademy provides interactive HTML and CSS courses, ranging from beginner to advanced levels. Their step-by-step lessons, quizzes, and projects enable you to gain practical experience while learning the fundamentals of HTML and CSS.