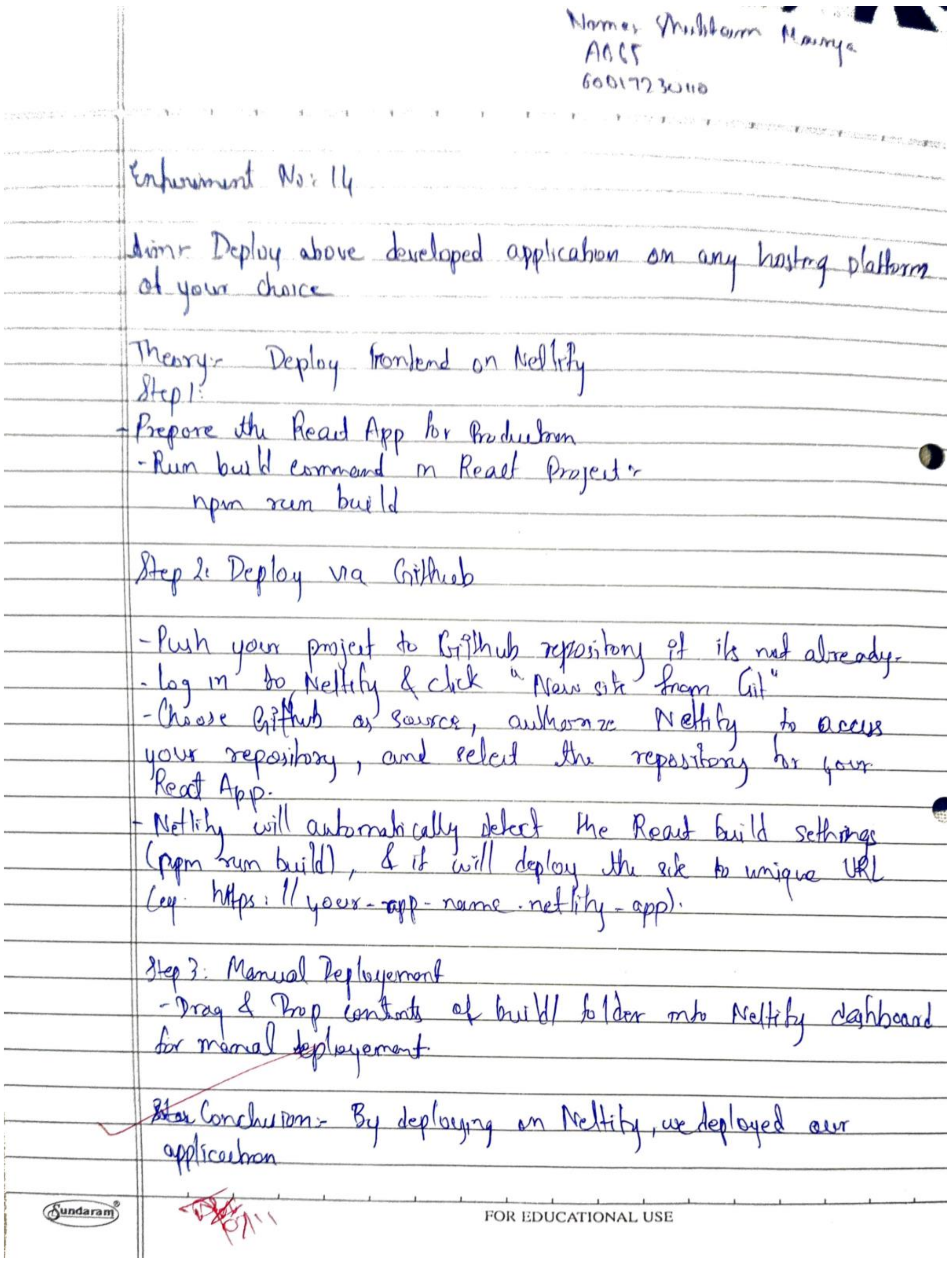
Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Artificial Intelligence and Machine Learning
B.Tech. Sem: V Subject: Full Stack Development Laboratory (DJS22AML504)

Experiment 14

Name: Shubham Mourya **SAP ID: 60017230110**



Name: Shubham Mourya
AIML
60017230110

Experiment No: 14

Aim: Deploy above developed application on any hosting platform of your choice

Theory: Deploy frontend on Netlify
Step 1:
- Prepare the React App for Production
- Run build command in React Project:
    npm run build

Step 2: Deploy via Github

- Push your project to Github repository if its not already.
- Log in to Netlify & click "New site from Git"
- Choose Github as source, authorize Netlify to access your repository, and select the repository for your React App.
- Netlify will automatically detect the React build settings (npm run build), & it will deploy the site to unique URL (eg: https://your-app-name.netlify.app).

Step 3: Manual Deployement
- Drag & Drop contents of build folder into Netlify dashboard for manual deployement

~~Step~~ Conclusion:- By deploying on Netlify, we deployed our application

[illegible]

Sundaram
FOR EDUCATIONAL USE

Department of Artificial Intelligence and Machine Learning
B.Tech. Sem: V Subject: Full Stack Development Laboratory (DJS22AML504)

Aim	**Deploy the above developed application on any hosting platform of your choice**
Software	VS Code
Pre-requisite	Active internet connection
Theory	**Steps to Deploy a React App on Netlify using GitHub** **Step 1: Create a React Application** At the very first, you need a react application. To create a react application, there are some prerequisites for your local system. There must be **Node.js** and **npm** installed in your system. If Node.js is not installed in your system, you can download it and install it from **this link**. Also, to install npm, you can run the following command on your terminal – npm install npm@latest -g Now open your VS Code or some other IDE, Open the terminal, and run the following command to create a React App- npx create-react-app My-Project Navigate to your project directory, using the following command – cd My-Project And, start the React application by using the following command – npm start **Step 2: Install the Required Dependencies** First of all, open your code in Visual Studio Code or in any other IDE. Run the following command, in order to install the **Netlify CLI** package as a **development dependency** in your React project. npm install netlify-cli -g This allows users to deploy the react app directly from the terminal. (This is not a required step) **Step 3: Create a Build directory** Using the following command, we will now create a **build** for our production. npm run build This creates the main directory as '**build**'. This Build folder contains all the required files that we need to deploy our application. Now, we are ready to push this code into our GitHub repository. **Step 4: Create a GitHub repository** Now, You have a react app on your local system. Open your GitHub account and create a new repository for your project. Name the repository anything you want, for example, **My project**. Push the code to the repository using Git Bash. For this, you need to have a good understanding of version controls. Our project is uploaded on Github, We'll now move forward towards deploying our project: There are various development platforms like Heruku, Vercel, Netlify, Github pages, Digital Ocean, etc. But one of the most popular and easy-to-use platforms is Netlify. We'll be using Netlify to deploy our project in this article. **Step 5: Setting up Netlify**

Create a Netlify account (if you have not any). Now, connect your GitHub account with Netlify. Authorize Netlify to use GitHub to access your repositories.
Follow these steps further:

1. Go to the '**Sites**' section from the header of the Netlify website's dashboard.
2. Now, click on '**Add new site'.**
3. Select **'import an existing project**', This will redirect you to a page where it will ask you to connect a Git provider.
4. Select your Git provider, if our things are the same so far, choose **GitHub**.
5. This will take you to a page, where all your GitHub repositories are listed.
6. Choose your project repository.
7. Now, there will be some settings,
   a) Do not change the pre-filled details.
   b) Leave the **Base Directory** empty.
   c) In the **Build Command,** type "**npm run build**".
   d) In the Publish Directory, type "**build**".
8. And, finally, click on the Deploy site.
9. It'll redirect you to the deployment page, you can check the logs here, and see how things are working step by step.
10. If the deployment fails for some reason, you'll get the error message. Remove the error and try deploying it again.

By the time, The react app is deployed. Once the deployment gets completed, you get a public URL for your project with the domain '**netlify.com**'.

Step 6: Other ways to deploy

Using the GitHub repository is not the only way to deploy a React App on Netlify, this can also be done by –

- Drag and Drop method
- Using Netlify CLI (Command Line Interface)

Drag and Drop method is the easiest way to deploy your react app. All you need to do is, after **step 3,** Once you have to create a build directory, drag this '**build**' folder, and drop it to the dashboard of Netlify. And start deploying.

The other way is to use **Netlify CLI.** After performing **Step 2,** Navigate to your working directory and run the following command to deploy the application.

netlify deploy

These were all the ways, a React application can be deployed on Netlify.

Steps To Deploy MERN Full-Stack To Render

Deploying a MERN full-stack application to Render can be a straightforward process when following these steps. Aspiring professionals are suggested to join the **MERN Stack Course with Placement** for the best guidance. Render provides a simple and efficient platform for hosting various types of applications, including MERN stacks.

1. Set Up Your MERN Application Locally

To begin with, make sure your MERN application is fully developed and tested locally before proceeding with deployment. Ensure that your application is structured properly with separate backend (Node.js/Express) and frontend (React.js) directories.

2. Create a Render Account

For the next step, if you haven't already, sign up for a Render account at render.com, or use you GitHub account. Render offers a generous free tier for hosting your applications.

3. Prepare Your Application for Deployment

This step involves preparing your MERN application for deployment. For this, prepare the backend of the application first before proceeding to the frontend.

- **Backend (Node.js/Express):**

· Ensure that your backend code is configured to use environment variables for sensitive data such as database credentials.

· Next, create a ***"server.js"*** file that initializes your Express server.

· Update your ***"package.json"*** file to include a start script for running your server.

- **Frontend (React.js):**

· Now, build your React application for production. Run "npm run build" in your frontend directory to generate a production-ready build.

· Make sure your frontend is configured to fetch data from your backend server using the appropriate endpoints.

4. Set Up Your Database

Moving forward, if your MERN application relies on MongoDB, you can use MongoDB Atlas for cloud-hosted databases. Set up a MongoDB Atlas cluster and configure access credentials.

	5. Configure Environment Variables on Render Next, in your Render dashboard, navigate to your MERN service and configure environment variables for your application. Add the variables for MongoDB connection strings, API keys, and any other sensitive data your application requires. **6. Deploy Your Backend** Create a New Web Service: · Click on "New" in the Render dashboard and select "Web Service." · Then choose a name for your service and specify the GitHub repository where your backend code is hosted. • **Configure Build and Start Command:** · Next, in the settings for your web service, specify the build command (e.g., ***"npm install"***) and start command (e.g., ***"npm start"***) for your backend. • **Add Environment Variables:** · Add environment variables for your backend configuration, including MongoDB connection strings. • **Deploy Your Backend:** · Next, click on "Deploy" to deploy your backend to Render. **7. Deploy Your Frontend** • **Create a New Static Site:** · Now, click on "New" in the Render dashboard and select "Static Site." · Choose a name for your static site and specify the GitHub repository where your frontend code is hosted. • **Configure Build Command:** · Going ahead, in the settings for your static site, specify the build command (e.g., ***"npm run build"***) for your React frontend.

	• **Deploy Your Frontend:** · Next, click on "Deploy" to deploy your frontend to Render. **8. Access Your Application** Once both your backend and frontend are deployed, Render will provide you with URLs for accessing your services. You can access your MERN application through the provided URLs. Deploying a MERN full-stack application to Render involves configuring backend and frontend services, setting up environment variables, and deploying each component separately. By following these steps, you can efficiently host your MERN application on Render's platform, ensuring scalability, reliability, and ease of management.

Code	Frontend
	
	SignIn.jsx
	`import { BottomWarning } from "../components/BottomWarning";`
	`import { Button } from "../components/Button";`
	`import { Heading } from "../components/Heading";`
	`import { InputBox } from "../components/InputBox";`
	`import { SubHeading } from "../components/SubHeading";`
	`import { useState } from 'react';`
	`import axios from "axios";`
	`import { jwtDecode } from "jwt-decode";`
	
	`import { useNavigate } from 'react-router-dom'; // To redirect after sign-in`
	
	`export const Signin = () => {`
	`    const [email, setEmail] = useState("");`
	`    const [password, setPassword] = useState("");`
	`    const [error, setError] = useState(""); // State for error messages`
	`    const navigate = useNavigate(); // Hook for navigation`
	
	`    const handleSignIn = async () => {`
	`        try {`
	`            const response = await`
	`axios.post("http://localhost:3000/api/v1/user/signin", {`
	`                username: email, // Assuming 'username' is the same as 'email'`
	`                password`
	`            });`
	
	`            // Save the token (if needed) and redirect`
	`            localStorage.setItem("token", response.data.token);`
	`            localStorage.setItem("userId",jwtDecode(response.data.token).userId`
	`)`
	`            navigate("/dashboard"); // Redirect to the dashboard or wherever`

```
        } catch (err) {
            const errorMessage = err.response?.data?.message || "Sign in failed.
Please try again.";
            setError(errorMessage); // Set the error message to display
        }
    };

    return (
        <div className="bg-slate-300 h-screen flex justify-center">
            <div className="flex flex-col justify-center">
                <div className="rounded-lg bg-white w-80 text-center p-2 h-
max px-4">
                    <Heading label={"Sign in"} />
                    <SubHeading label={"Enter your credentials to access your
account"} />
                    <InputBox
                        placeholder="azlan@gmail.com"
                        label={"Email"}
                        onChange={(e) => setEmail(e.target.value)}
                    />
                    <InputBox
                        placeholder="123456"
                        label={"Password"}
                        type="password"
                        onChange={(e) => setPassword(e.target.value)}
                    />
                    {error && <div className="text-red-500">{error}</div>} {/*
Display error message */}
                    <div className="pt-4">
                        <Button label={"Sign in"} onClick={handleSignIn} />
                    </div>
                    <BottomWarning label={"Don't have an account?"}
buttonText={"Sign up"} to={"/signup"} />
                </div>
            </div>
        </div>
    );
}
```

SignUp.jsx

```
import { useState } from "react"
import { BottomWarning } from "../components/BottomWarning"
import { Button } from "../components/Button"
import { Heading } from "../components/Heading"
import { InputBox } from "../components/InputBox"
import { SubHeading } from "../components/SubHeading"
import axios from "axios";
import { useNavigate } from "react-router-dom"
import { jwtDecode } from "jwt-decode";

export const Signup = () => {
  const [firstName, setFirstName] = useState("");
  const [lastName, setLastName] = useState("");
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  const navigate = useNavigate();

  return <div className="bg-slate-300 h-screen flex justify-center">
    <div className="flex flex-col justify-center">
      <div className="rounded-lg bg-white w-80 text-center p-2 h-max px-
4">
        <Heading label={"Sign up"} />
        <SubHeading label={"Enter your infromation to create an account"} />
        <InputBox onChange={e => {
          setFirstName(e.target.value);
        }} placeholder="John" label={"First Name"} />
        <InputBox onChange={(e) => {
          setLastName(e.target.value);
        }} placeholder="Doe" label={"Last Name"} />
        <InputBox onChange={e => {
          setUsername(e.target.value);
        }} placeholder="azlan@gmail.com" label={"Email"} />
        <InputBox onChange={(e) => {
          setPassword(e.target.value)
        }} placeholder="123456" label={"Password"} />
        <div className="pt-4">
          <Button onClick={async () => {
            const response = await
axios.post("http://localhost:3000/api/v1/user/signup", {
              username,
              firstName,
              lastName,
              password
            });
            localStorage.setItem("token", response.data.token)
            localStorage.setItem("userId", jwtDecode(response.data.token))
            navigate("/dashboard")
          }} label={"Sign up"} />
        </div>
        <BottomWarning label={"Already have an account?"}
buttonText={"Sign in"} to={"/signin"} />
      </div>
    </div>
  </div>
}
```

Transactions.jsx

```
import { useState, useEffect} from "react"
import { Navigate } from "react-router-dom"
import axios from "axios"
import { TransactionCard } from "../components/TransactionCard"

export const Transactions = ()=>{
    const token = localStorage.getItem("token")
    const [transactions, setTransactions] = useState([]);
    const [loading, setLoading] = useState(true); // Loading state

    if(!token){
        return(<Navigate to = '/signin'/>)
    }

    useEffect(() => {
        const fetchTransactions = async () => {
            try {
                const response = await
axios.get('http://localhost:3000/api/v1/account/history', {
                    headers: {
                        Authorization: "Bearer " + token,
                    }
                });
                setTransactions(response.data); // Store the transaction data
                setLoading(false);
            } catch (error) {
                console.error("Error fetching transactions", error);
                setLoading(false);
            }
        };

        fetchTransactions();
    }, []); // Empty dependency array ensures this runs once on component
mount

    if (loading) {
        return <div>Loading transactions...</div>; // Show loading message
while data is being fetched
    }

    return(
        <div className="m-8">
            <h2 className="text-2xl font-bold mb-4">Transaction
History</h2>
            <div className="space-y-4">
                {transactions.length > 0 ? transactions.map((transaction) => (
                    <TransactionCard key={transaction._id}
transaction={transaction} />
                )) : <div>No transactions found.</div>}
            </div>
        </div>
    )
}

UpdateInfo.jsx
import axios from 'axios'
import { useState, Navigate} from 'react'
import { useNavigate } from "react-router-dom"

export const UpdateInfo = () => {
    const [message,setMessage] = useState('')
    const [firstName, setFirstName] = useState('')
    const [lastName, setLastname] = useState('')
    const [password, setPassword] = useState('')

    const navigate = useNavigate();

    const handleSubmit = async(event)=>{
        setMessage('')
        event.preventDefault();
        try{
            const response = axios.put('http://localhost:3000/api/v1/user', {
                password: password,
                firstName: firstName,
                lastName: lastName
            }, {
                headers: {
                    Authorization: "Bearer " + localStorage.getItem("token"),
                },
            })
            setMessage('Successfull!')
        }
        catch(err){
            throw(e)
            setMessage('Error occured!')
        }
    }
    return (
        <div className='flex justify-center h-screen bg-gray-100'>
            <div className="h-full w-full max-w-5xl flex flex-col justify-center
p-10">
                <form className="border rounded-lg border-zinc-400 w-full
max-w-4xl p-12 bg-white">
```

```
          <h2 className="text-3xl font-bold text-center mb-10">Update
Information</h2>
          <div className="flex flex-wrap -mx-3 mb-6">
            <div className="w-full md:w-1/2 px-3 mb-6 md:mb-0">
              <label className="block uppercase tracking-wide text-
gray-700 text-lg font-bold mb-2" htmlFor="grid-first-name">
                First Name
              </label>
              <input
                className="appearance-none block w-full bg-gray-200
text-gray-700 border rounded py-4 px-5 mb-3 leading-tight focus:outline-
none focus:bg-white text-base"
                id="grid-first-name"
                type="text"
                placeholder="Jane"
                value={firstName}
                onChange={(e)=>{setFirstName(e.target.value)}}
              />

            </div>
            <div className="w-full md:w-1/2 px-3">
              <label className="block uppercase tracking-wide text-
gray-700 text-lg font-bold mb-2" htmlFor="grid-last-name">
                Last Name
              </label>
              <input
                className="appearance-none block w-full bg-gray-200
text-gray-700 border border-gray-200 rounded py-4 px-5 leading-tight
focus:outline-none focus:bg-white text-base"
                id="grid-last-name"
                type="text"
                placeholder="Doe"
                value={lastName}
                onChange={(e)=>{setLastname(e.target.value)}}
              />
            </div>
          </div>
          <div className="flex flex-wrap -mx-3 mb-6">
            <div className="w-full px-3">
              <label className="block uppercase tracking-wide text-
gray-700 text-lg font-bold mb-2" htmlFor="grid-password">
                Password
              </label>
              <input
                className="appearance-none block w-full bg-gray-200
text-gray-700 border border-gray-200 rounded py-4 px-5 mb-3 leading-tight
focus:outline-none focus:bg-white text-base"
                id="grid-password"
```

```
                        type="password"
                        placeholder="*****************"
                        value={password}
                        onChange={(e)=>{setPassword(e.target.value)}}
                    />
                    <p className="text-gray-600 text-sm italic">Make it as
long and as crazy as you'd like</p>
                </div>
            </div>
            <div className="flex justify-around">
                <button
                    type="submit"
                    onClick = {handleSubmit}
                    className="bg-blue-500 text-white font-bold py-2 px-6
rounded hover:bg-blue-600">
                    Submit
                </button>
                <button onClick={()=>{navigate("/dashboard")}}
className="bg-blue-500 text-white font-bold py-2 px-6 rounded hover:bg-
blue-600">Dashboard</button>
            </div>
            <div className='flex justify-center text-lg font-semibold'>
                <h2 className=' text-red  selection:font-bold py-2 px-6
rounded'>{message}</h2>
            </div>
        </form>
    </div>
  </div>
 )
}
```

Backend

```
// backend/routes/account.js
const express = require('express');
const { authMiddleware } = require('../middleware');
const { Account, Transaction } = require('../db');
const { default: mongoose } = require('mongoose');

const router = express.Router();

router.get("/balance", authMiddleware, async (req, res) => {
    const account = await Account.findOne({
        userId: req.userId
    });

    res.json({
        balance: account.balance
    })
});

router.post("/transfer", authMiddleware, async (req, res) => {
    const session = await mongoose.startSession();

    session.startTransaction();
    const { amount, to } = req.body;
    const from = req.userId
    // Fetch the accounts within the transaction
    const fromAccount = await Account.findOne({ userId: req.userId
}).session(session);
    const toAccount = await Account.findOne({ userId: to
}).session(session);

    if (!fromAccount || fromAccount.balance < amount) {
        await session.abortTransaction();
        return res.status(400).json({
            message: "Insufficient balance"
        });
    }

    if (!toAccount) {
        await session.abortTransaction();
        return res.status(400).json({
            message: "Invalid account"
        });
    }

    // Perform the transfer
    await Account.updateOne({ userId: req.userId }, { $inc: { balance: -
amount } }).session(session);
    await Account.updateOne({ userId: to }, { $inc: { balance: amount }
}).session(session);

    // Log the transaction in the Transaction collection
    const trans = new Transaction({
        from: from,
        to: to,
        amount: amount,
        type: 'outgoing'
    });

    //console.log('transaction:', outgoingTransaction)

    // Save the transactions
    await trans.save({ session });
    // Commit the transaction
    await session.commitTransaction();
    res.json({
        message: "Transfer successful"
    });
});
router.get('/history',authMiddleware, async(req,res)=>{
    try{
        const userId = req.userId

        const transactions = await Transaction.find({
            $or: [{from:userId}, {to:userId}] //This is an array of two conditions.
MongoDB will return documents where either condition is true.
        }).sort({date:-1});

        res.json(transactions)

    }catch(error){
        res.status(500).json({msg:"Failed"})
    }
})
module.exports = router;

// backend/user/index.js
const express = require('express');
const userRouter = require("./user");
const accountRouter = require("./account");

const router = express.Router();

router.use("/user", userRouter);
router.use("/account", accountRouter);

module.exports = router;

user.js
// backend/routes/user.js
const express = require('express');

const router = express.Router();
const zod = require("zod");
const { User, Account } = require("../db");
const jwt = require("jsonwebtoken");
const { JWT_SECRET } = require("../config");
const  { authMiddleware } = require("../middleware");

const signupBody = zod.object({
    username: zod.string().email(),
    firstName: zod.string(),
    lastName: zod.string(),
    password: zod.string()
})

router.post("/signup", async (req, res) => {
    const { success } = signupBody.safeParse(req.body)
    if (!success) {
        return res.status(411).json({
            message: "Email already taken / Incorrect inputs"
        })
    }

    const existingUser = await User.findOne({
        username: req.body.username
    })

    if (existingUser) {
        return res.status(411).json({
            message: "Email already taken/Incorrect inputs"
        })
    }

    const user = await User.create({
        username: req.body.username,
        password: req.body.password,
        firstName: req.body.firstName,
        lastName: req.body.lastName,
    })
    const userId = user._id;

    await Account.create({
        userId,
        balance: 10000
    })

    const token = jwt.sign({
        userId
    }, JWT_SECRET);

    res.json({
        message: "User created successfully",
        token: token
    })
})

const signinBody = zod.object({
    username: zod.string().email(),
    password: zod.string()
})

router.post("/signin", async (req, res) => {
    const { success } = signinBody.safeParse(req.body)
    if (!success) {
        return res.status(411).json({
            message: "Email already taken / Incorrect inputs"
        })
    }

    const user = await User.findOne({
        username: req.body.username,
        password: req.body.password
    });

    if (user) {
        const token = jwt.sign({
            userId: user._id
        }, JWT_SECRET);

        res.json({
            token: token
        })
        return;
    }


    res.status(411).json({
        message: "Error while logging in"
    })
})

const updateBody = zod.object({
    password: zod.string().optional(),
    firstName: zod.string().optional(),
    lastName: zod.string().optional(),
})

router.put("/", authMiddleware, async (req, res) => {

    const { success } = updateBody.safeParse(req.body)

    if (!success) {
        return res.status(411).json({
            message: "Unsuccessfull zod parse"
        })
    }
    await User.updateOne(
        {_id: req.userId},
        req.body
    )

    res.json({
        msg: "Updated successfully"
    })
})

router.get("/bulk", async (req, res) => {
    const filter = req.query.filter || "";

    const users = await User.find({
        $or: [{
            firstName: {
                "$regex": filter
            }
        }, {
            lastName: {
                "$regex": filter
            }
        }]
    })

    res.json({
        user: users.map(user => ({
            username: user.username,
            firstName: user.firstName,
            lastName: user.lastName,
            _id: user._id
        }))
    })
})

module.exports = router;
```

index.js

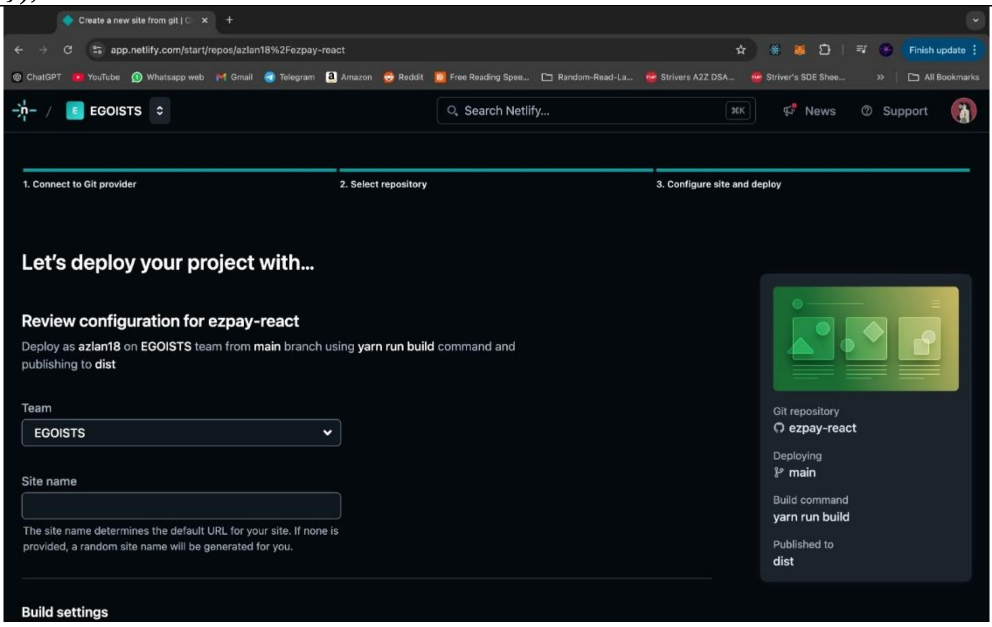
```
// backend/index.js
const express = require('express');
const cors = require("cors");
const rootRouter = require("./routes/index");

const app = express();

app.use(cors());
app.use(express.json());
app.use("/api/v1", rootRouter);

app.listen(3000, ()=>{
    console.log('Listening on port 3000')
});
```

Result


Create a new site from git | C
app.netlify.com/start/repos/azlan18%2Fezpay-react
ChatGPT YouTube Whatsapp web Gmail Telegram Amazon Reddit Free Reading Spee... Random-Read-La... Strivers A2Z DSA... Striver's SDE Shee... All Bookmarks
EGOISTS Search Netlify... News Support Finish update

1. Connect to Git provider 2. Select repository 3. Configure site and deploy

Let's deploy your project with...

Review configuration for ezpay-react

Deploy as azlan18 on EGOISTS team from main branch using yarn run build command and publishing to dist

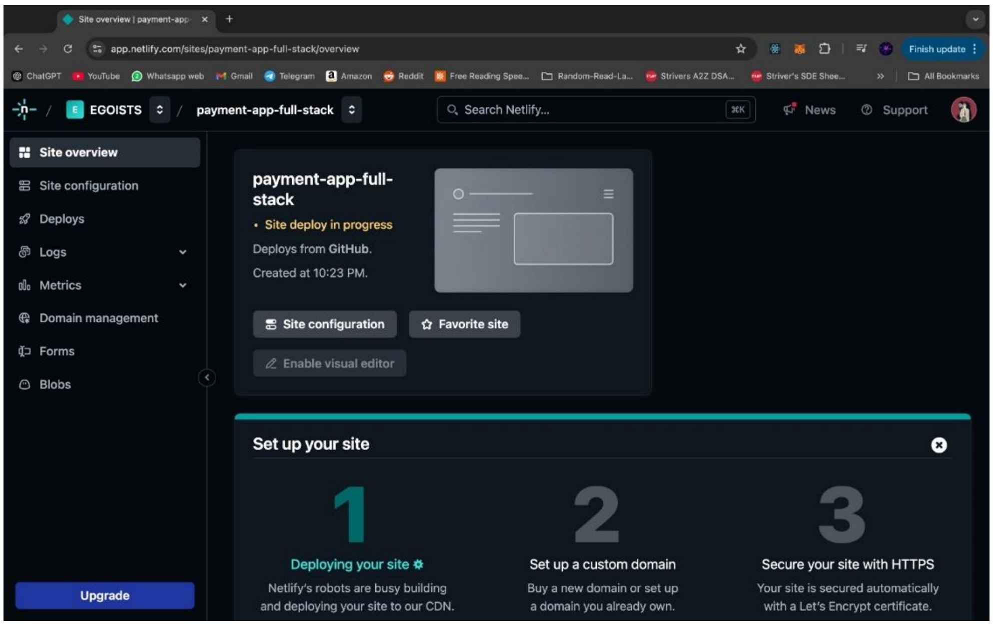
Team: EGOISTS

Site name:

The site name determines the default URL for your site. If none is provided, a random site name will be generated for you.

Git repository: ezpay-react
Deploying: main
Build command: yarn run build
Published to: dist

Build settings


Site overview | payment-app-
app.netlify.com/sites/payment-app-full-stack/overview
EGOISTS / payment-app-full-stack Search Netlify... News Support Finish update

Site overview
Site configuration
Deploys
Logs
Metrics
Domain management
Forms
Blobs
Upgrade

payment-app-full-stack
• Site deploy in progress
Deploys from GitHub.
Created at 10:23 PM.

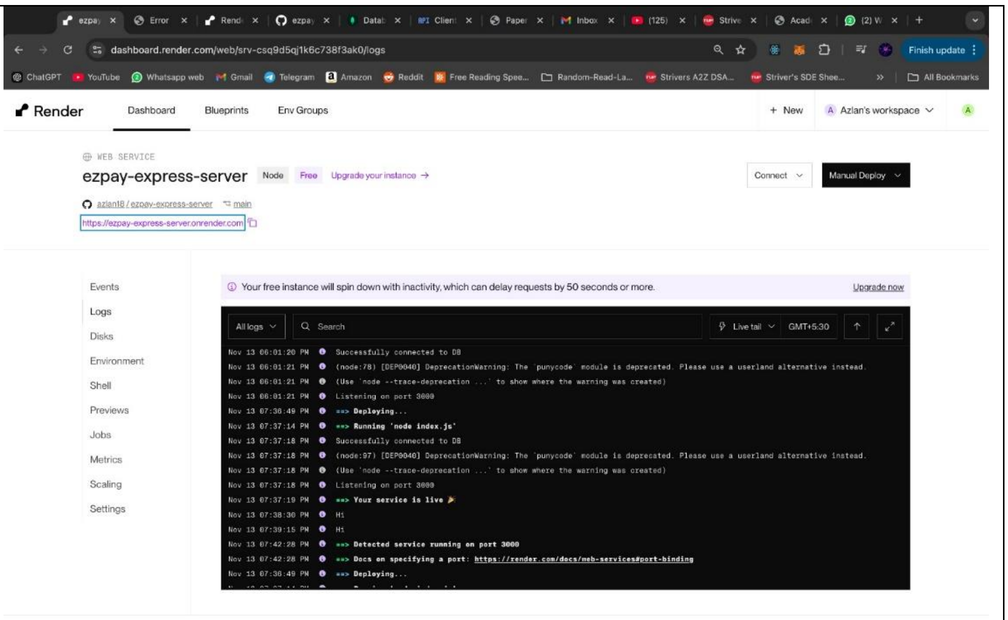
Site configuration Favorite site
Enable visual editor

Set up your site

1 Deploying your site
Netlify's robots are busy building and deploying your site to our CDN.

2 Set up a custom domain
Buy a new domain or set up a domain you already own.

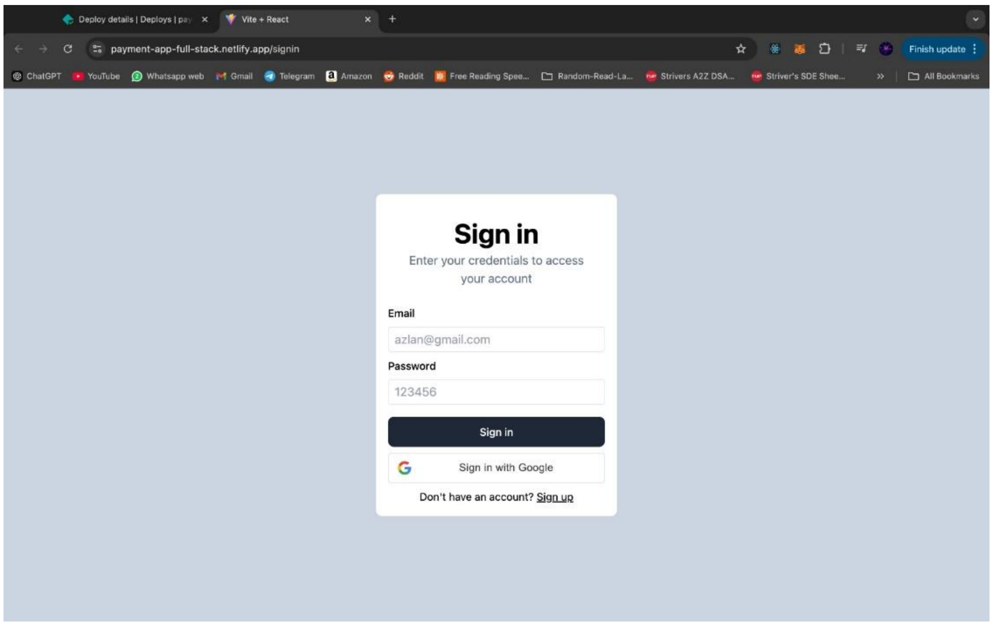
3 Secure your site with HTTPS
Your site is secured automatically with a Let's Encrypt certificate.


dashboard.render.com/web/srv-csq9d5qj1k6c738f3ak0/logs

WEB SERVICE
ezpay-express-server Node Free Upgrade your instance →
azlan18 / ezpay-express-server main
https://ezpay-express-server.onrender.com

Your free instance will spin down with inactivity, which can delay requests by 50 seconds or more. Upgrade now

```
Nov 13 06:01:20 PM  Successfully connected to DB
Nov 13 06:01:21 PM  (node:78) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
Nov 13 06:01:21 PM  (Use `node --trace-deprecation ...` to show where the warning was created)
Nov 13 06:01:21 PM  Listening on port 3000
Nov 13 07:36:49 PM  ==> Deploying...
Nov 13 07:37:14 PM  ==> Running 'node index.js'
Nov 13 07:37:18 PM  Successfully connected to DB
Nov 13 07:37:18 PM  (node:97) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
Nov 13 07:37:18 PM  (Use `node --trace-deprecation ...` to show where the warning was created)
Nov 13 07:37:18 PM  Listening on port 3000
Nov 13 07:37:19 PM  ==> Your service is live 🎉
Nov 13 07:38:30 PM  Hi
Nov 13 07:39:15 PM  Hi
Nov 13 07:42:28 PM  ==> Detected service running on port 3000
Nov 13 07:42:28 PM  ==> Docs on specifying a port: https://render.com/docs/web-services#port-binding
Nov 13 07:36:49 PM  ==> Deploying...
```


payment-app-full-stack.netlify.app/signin

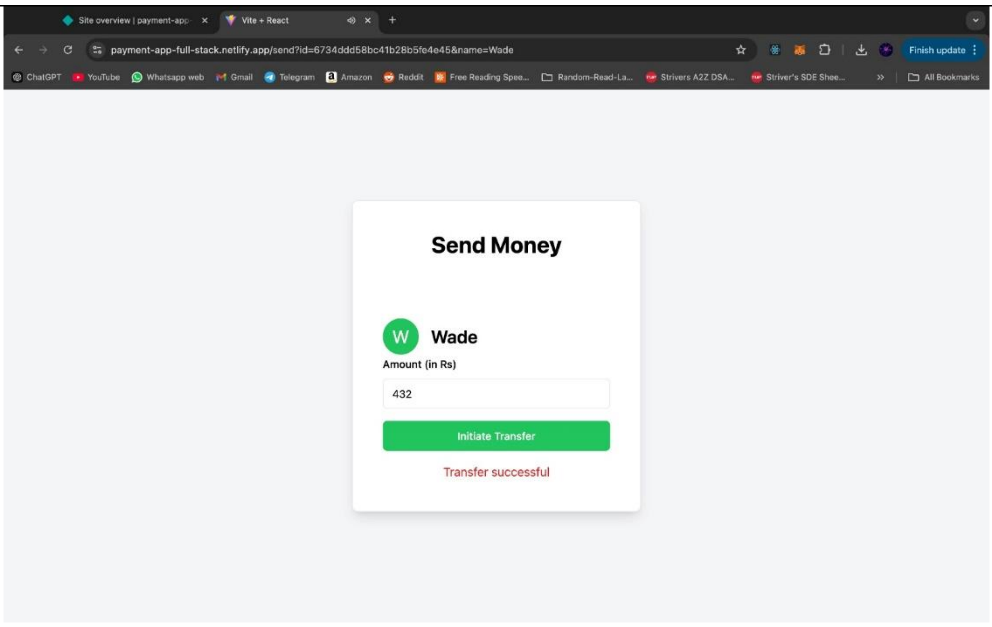
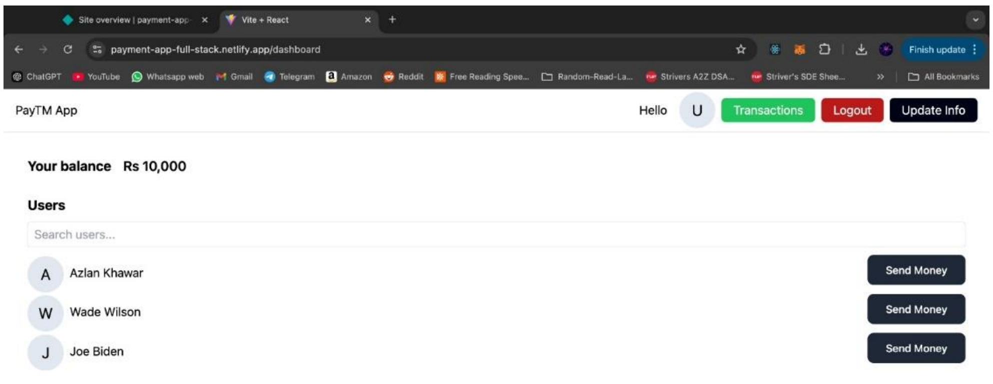
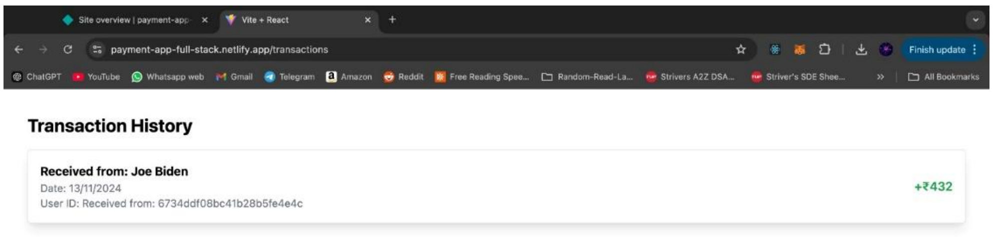
Sign in

Enter your credentials to access your account

Email: azlan@gmail.com

Password: 123456

Deploy details | Deploys | pay
Vite + React
Sign in
Sign in with Google
Don't have an account? Sign up

	 payment-app-full-stack.netlify.app/send?id=6734ddd58bc41b28b5fe4e45&name=Wade **Send Money** W Wade Amount (in Rs) 432 Initiate Transfer Transfer successful  payment-app-full-stack.netlify.app/dashboard PayTM App — Hello U — Transactions — Logout — Update Info **Your balance** Rs 10,000 **Users** Search users... A Azlan Khawar — Send Money W Wade Wilson — Send Money J Joe Biden — Send Money  payment-app-full-stack.netlify.app/transactions **Transaction History** **Received from: Joe Biden** Date: 13/11/2024 User ID: Received from: 6734ddf08bc41b28b5fe4e4c +₹432
Conclusion	Thus we have successfully deployed our MERN application using Render and Netlify