Project-AnimeFlixx

Website made with React.js as frontend and Node.js and Express as backend and MongoDB database

# Problem Statement

Design a responsive website using MERN Stack to stream different anime which will be fetched from the backend. The user will sign up /login before he /she can see any anime. Use jwt tokens to store user login credentials in the local storage. Also make an admin page to upload animes to backend

# Technologies used

<ul>

  <li>React.js</li>

  <li>Node.js</li>

  <li>material-ui</li>

  <li>Express.js</li>

  <li>MongoDb</li>

</ul>

Client/src/actions/anime.js

import \* as api from "../api/index.js";

export const getAnime = () => async (dispatch) => {

  try {

    const { data } = await api.getAnime();

    dispatch({ type: "FETCH\_ALL", payload: data });

  } catch (error) {

    console.log(error.message);

  }

};

Client/src/actions/auth.js

import \* as api from "../api/index.js";

export const signin = (formData, router) => async (dispatch) => {

  try {

    const { data } = await api.signIn(formData);

    dispatch({ type: "AUTH", data });

    router.push("/");

  } catch (error) {

    console.log(error);

  }

};

export const signuphere = (formData, router) => async (dispatch) => {

  try {

    const { data } = await api.signUp(formData);

    dispatch({ type: "AUTH", data });

    router.push("/");

  } catch (error) {

    console.log(error);

  }

};

Client/src/api/index.js

import axios from "axios";

const API = axios.create({ baseURL: "http://localhost:5000" });

export const getAnime = () => API.get("/anime");

export const signIn = (formData) => API.post("/user/signin", formData);

export const signUp = (formData) => API.post("/user/signup", formData);

Client/src/layouts/layoutsDefault.js

import React from 'react';

import Header from '../components/layout/Header';

import Footer from '../components/layout/Footer';

const LayoutDefault = ({ children }) => (

  <>

    <Header navPosition="right" className="reveal-from-bottom" />

    <main className="site-content">

      {children}

    </main>

    <Footer />

  </>

);

export default LayoutDefault;

Client/src/reducers/anime.js

export default (animes = [], action) => {

  switch (action.type) {

    case "FETCH\_ALL":

      return action.payload;

    default:

      return animes;

  }

};

Client/src/reducers/auth.js

const authReducer = (state = { authData: null }, action) => {

  switch (action.type) {

    case "AUTH":

      localStorage.setItem("profile", JSON.stringify({ ...action?.data }));

      return { ...state, authData: action.data, loading: false, errors: null };

    case "LOGOUT":

      localStorage.clear();

      return { ...state, authData: null, loading: false, errors: null };

    default:

      return state;

  }

};

export default authReducer;

Client/src/reducers/index.js

import { combineReducers } from 'redux';

import auth from './auth'

export const reducers = combineReducers({ auth });

Client/src/utlis/AppRoute.js

import React from 'react';

import { Route } from 'react-router-dom';

const AppRoute = ({

  component: Component,

  layout: Layout,

  ...rest

}) => {

  Layout = (Layout === undefined) ? props => (<>{props.children}</>) : Layout;

  return (

    <Route

      {...rest}

      render={props => (

        <Layout>

          <Component {...props} />

        </Layout>

      )} />

  );

}

export default AppRoute;

Client/src/utlis/ScrollReveal.js

import React, { useState, useEffect, useImperativeHandle } from 'react';

import PropTypes from 'prop-types';

import { throttle } from 'lodash';

const ScrollReveal = React.forwardRef((props, ref) => {

  const [viewportHeight, setViewportheight] = useState(window.innerHeight);

  const [revealEl, setRevealel] = useState([]);

  const checkComplete = () => {

    return revealEl.length <= document.querySelectorAll('[class\*=reveal-].is-revealed').length;

  };

  const elementIsVisible = (el, offset) => {

    return (el.getBoundingClientRect().top <= viewportHeight - offset);

  };

  const revealElements = () => {

    if (checkComplete()) return;

    for (let i = 0; i < revealEl.length; i++) {

      let el = revealEl[i];

      let revealDelay = el.getAttribute('data-reveal-delay');

      let revealOffset = (el.getAttribute('data-reveal-offset') ? el.getAttribute('data-reveal-offset') : '200');

      let listenedEl = (el.getAttribute('data-reveal-container') ? el.closest(el.getAttribute('data-reveal-container')) : el);

      if (elementIsVisible(listenedEl, revealOffset) && !el.classList.contains('is-revealed')) {

        if (revealDelay && revealDelay !== 0) {

          setTimeout(function () {

            el.classList.add('is-revealed');

          }, revealDelay);

        } else {

          el.classList.add('is-revealed');

        }

      }

    }

  };

  useImperativeHandle(ref, () => ({

    init() {

      setRevealel(document.querySelectorAll('[class\*=reveal-]'));

    }

  }));

  useEffect(() => {

    if (typeof revealEl !== 'undefined' && revealEl.length > 0) {

      if (!checkComplete()) {

        window.addEventListener('scroll', handleScroll);

        window.addEventListener('resize', handleResize);

      }

      revealElements();

    }

  }, [revealEl]);

  const handleListeners = () => {

    if (!checkComplete()) return;

    window.removeEventListener('scroll', handleScroll);

    window.removeEventListener('resize', handleResize);

  };

  const handleScroll = throttle(() => {

    handleListeners();

    revealElements();

  }, 30);

  const handleResize = throttle(() => {

    setViewportheight(window.innerHeight);

  }, 30);

  useEffect(() => {

    handleListeners();

    revealElements();

  }, [viewportHeight]);

  return (

    <>

      {props.children()}

    </>

  );

});

ScrollReveal.propTypes = {

  children: PropTypes.func.isRequired

};

export default ScrollReveal;

Client/src/utlis/SectionProps.js

import PropTypes from 'prop-types';

const SectionShared = {

  types: {

    topOuterDivider: PropTypes.bool,

    bottomOuterDivider: PropTypes.bool,

    topDivider: PropTypes.bool,

    bottomDivider: PropTypes.bool,

    hasBgColor: PropTypes.bool,

    invertColor: PropTypes.bool

  },

  defaults: {

    topOuterDivider: false,

    bottomOuterDivider: false,

    topDivider: false,

    bottomDivider: false,

    hasBgColor: false,

    invertColor: false

  }

}

export const SectionProps = {

  types: {

    ...SectionShared.types

  },

  defaults: {

    ...SectionShared.defaults

  }

}

export const SectionSplitProps = {

  types: {

    ...SectionShared.types,

    invertMobile: PropTypes.bool,

    invertDesktop: PropTypes.bool,

    alignTop: PropTypes.bool,

    imageFill: PropTypes.bool

  },

  defaults: {

    ...SectionShared.defaults,

    invertMobile: false,

    invertDesktop: false,

    alignTop: false,

    imageFill: false

  }

}

export const SectionTilesProps = {

  types: {

    ...SectionShared.types,

    pushLeft: PropTypes.bool

  },

  defaults: {

    ...SectionShared.defaults,

    pushLeft: false

  }

}

Client/src/views/Auth/Auth.js

import { React, useState } from 'react'

import { Avatar, Button, Paper, Grid, Typography, Container, TextField } from '@material-ui/core';

import useStyles from './styles';

import Input from './Input';

import LockOutlinedIcon from '@material-ui/icons/LockOutlined';

import { GoogleLogin } from 'react-google-login';

import Icon from './icon';

import { useDispatch } from 'react-redux';

import { useHistory } from 'react-router-dom';

import { signin, signuphere } from '../../actions/auth';

const initialState = { firstName: '', lastName: '', email: '', password: '', confirmPassword: '' };

function Auth() {

  const classes = useStyles();

  const [signup, setIsSignup] = useState(false);

  const [showPassword, setshowPassword] = useState(false);

  const [form, setForm] = useState(initialState);

  const dispatch = useDispatch();

  const history = useHistory();

  const handleSubmit = (e) => {

    e.preventDefault();

    console.log(form);

    if (signup) {

      console.log(form);

      dispatch(signuphere(form, history));

    } else {

      dispatch(signin(form, history));

    }

  }

  const handleChange = (e) =>  setForm({ ...form, [e.target.name]: e.target.value });

  const handleShowPassword = () => {

    setshowPassword(!showPassword)

  }

  const switchMode = () => {

    setIsSignup(!signup)

  }

  const googleSuccess = async (res) => {

    const result = res?.profileObj;

    const token = res?.tokenId;

    try {

      dispatch({ type: 'AUTH', data: { result, token } });

      history('/');

    } catch (error) {

      console.log(error);

    }

  }

  const googleError = (error) => {

    console.log(error)

    alert('Google Sign In was unsuccessful. Try again later');

  }

  return (

    <div className="gradient-bg-welcome">

    <Container component="main" maxWidth="xs">

      <Paper className={classes.paper} elevation={3}>

        <Avatar className={classes.avatar}>

          <LockOutlinedIcon/>

        </Avatar>

        <Typography variant="h5">{signup ? 'Sign Up' : 'Sign In'}</Typography>

        <form className={classes.form} onSubmit={handleSubmit}>

          <Grid container spacing={2}>

            {

              signup && (

                <>

                  <Input name="firstName" label="First Name" handleChange={handleChange} autoFocus half />

                  <Input name="lastName" label="Last Name" handleChange={handleChange} half />

                </>

              )}

            <Input name="email" label="Email Address" handleChange={handleChange} type="email" />

            <Input name="password" label="Password" handleChange={handleChange} type={showPassword ? 'text' : 'password'} handleShowPassword={handleShowPassword} />

            {signup && <Input name="confirmPassword" label="Repeat Password" handleChange={handleChange} type="password" />}

          </Grid>

          <Button type="submit" fullWidth variant="contained" color='#38b2f9' className={classes.submit}>

            {signup ? 'Sign Up' : 'Sign In'}

          </Button>

          <GoogleLogin

            clientId='983873906158-moqbtetckfo6ev76t591i5sagb40jr1a.apps.googleusercontent.com'

            render={(renderProps) => (

              <Button className={classes.googleButton} fullWidth onClick={renderProps.onClick} disabled={renderProps.disabled} startIcon={<Icon />} variant="contained">

                Google Sign In

              </Button>

            )}

            onSuccess={googleSuccess}

            onFailure={googleError}

            cookiePolicy="single\_host\_origin"/>

          <Grid container justify="flex-end">

            <Grid item>

              <Button onClick={switchMode}>

                {signup ? 'Already have an account? Sign in' : "Don't have an account? Sign Up"}

              </Button>

            </Grid>

          </Grid>

        </form>

      </Paper>

    </Container>

    </div>)

}

export default Auth

Client/src/views/Auth/icon.js

import React from 'react';

const icon = () => (

  <svg style={{ width: '20px', height: '20px' }} viewBox="0 0 24 24">

    <path

      fill="currentColor"

      d="M21.35,11.1H12.18V13.83H18.69C18.36,17.64 15.19,19.27 12.19,19.27C8.36,19.27 5,16.25 5,12C5,7.9 8.2,4.73 12.2,4.73C15.29,4.73 17.1,6.7 17.1,6.7L19,4.72C19,4.72 16.56,2 12.1,2C6.42,2 2.03,6.8 2.03,12C2.03,17.05 6.16,22 12.25,22C17.6,22 21.5,18.33 21.5,12.91C21.5,11.76 21.35,11.1 21.35,11.1V11.1Z"/></svg>

);

export default icon;

Client/src/views/Auth/input.js

import React from 'react';

import { TextField, Grid, InputAdornment, IconButton } from '@material-ui/core';

import Visibility from '@material-ui/icons/Visibility';

import VisibilityOff from '@material-ui/icons/VisibilityOff';

const Input = ({ name, handleChange, label, half, autoFocus, type, handleShowPassword }) => (

  <Grid item xs={12} sm={half ? 6 : 12}>

    <TextField

      name={name}

      onChange={handleChange}

      variant="outlined"

      required

      fullWidth

      label={label}

      autoFocus={autoFocus}

      type={type}

      InputProps={name === 'password' ? {

        endAdornment: (

          <InputAdornment position="end">

            <IconButton onClick={handleShowPassword}>

              {type === 'password' ? <Visibility /> : <VisibilityOff />}

            </IconButton>

          </InputAdornment>

        ),

      } : null}/></Grid>

);

export default Input;

Client/src/views/Auth/styles.js

import { makeStyles } from '@material-ui/core/styles';

export default makeStyles((theme) => ({

  paper: {

    marginTop: theme.spacing(10),

    display: 'flex',

    flexDirection: 'column',

    alignItems: 'center',

    padding: theme.spacing(2),

  },

  root: {

    '& .MuiTextField-root': {

      margin: theme.spacing(1),

    },

  },

  avatar: {

    margin: theme.spacing(1),

    backgroundColor: theme.palette.secondary.main,

  },

  form: {

    width: '100%',

    marginTop: theme.spacing(3),

  },

  submit: {

    margin: theme.spacing(3, 0, 2),

    backgroundColor: '#2b89bf',

  },

  googleButton: {

    marginBottom: theme.spacing(2),

    backgroundColor: '#2b89bf',

  },

}));

Client/src/views/Home.js

import React from 'react';

import Hero from '../components/sections/Hero';

import FeaturesTiles from '../components/sections/FeaturesTiles';

import FeaturesSplit from '../components/sections/FeaturesSplit';

import Testimonial from '../components/sections/Testimonial';

const Home = () => {

  return (

    <>

      <Hero className="illustration-section-01" />

      <FeaturesTiles />

      <FeaturesSplit invertMobile topDivider imageFill className="illustration-section-02" />

      <Testimonial topDivider /></>

  );

}

export default Home;

Client/src/views/Main.js

import React from "react";

import AllShows from "../components/sections/AllShows";

import Banner from "../components/sections/Banner/Banner";

import Modal from "../components/elements/Modal"

const Main = () => {

  return (

    <div>

      <Banner />

      <AllShows />

    </div>

  );

};

export default Main;

Client/src/views/Pricing.js

import React from "react";

const Pricing = () => {

  return (

    <div>

      <img

        style={{

          display: "block",

          marginLeft: "auto",

          marginRight: "auto",

        }}

        src={require("../assets/images/pricing.png")}

        alt="Open"

      />

    </div>

  );

};

export default Pricing;

Client/src/views/Privacy.js

import React from "react";

const Privacy = () => {

  return (

    <div style={{margin:"10px",marginTop:"100px"}}>

      <h1 class="kb-title">Privacy Statement</h1>

      <div>

        <p>

          This Privacy Statement explains our practices, including your choices,

          regarding the collection, use, and disclosure of certain information,

          including your personal information in connection with the AnimeFlixx

          service.

        </p>

        <h3>Contacting Us</h3>

        <p>

          If you have general questions about your account or how to contact

          customer service for assistance, please visit our online help center

          at <a href="https://help.AnimeFlixx.com">help.AnimeFlixx.com</a>. For

          questions specifically about this Privacy Statement, or our use of

          your personal information, cookies or similar technologies, please

          contact our Data Protection Officer/Privacy Office by email at{" "}

          <a href="mailto:privacy@AnimeFlixx.com">privacy@AnimeFlixx.com</a>.{" "}

        </p>

        <p>

          The data controller of your personal information is{" "}

          <a href="https://help.AnimeFlixx.com/legal/corpinfo">

            AnimeFlixx Entertainment Services India LLP

          </a>

          . Please note that if you contact us to assist you, for your safety

          and ours we may need to authenticate your identity before fulfilling

          your request.

        </p>

        <h3>Collection of Information</h3>

        <p>We receive and store information about you such as:</p>

Client/src/App.js

import React, { useRef, useEffect,useState } from "react";

import { useLocation, Switch } from "react-router-dom";

import AppRoute from "./utils/AppRoute";

import ScrollReveal from "./utils/ScrollReveal";

import ReactGA from "react-ga";

import LayoutDefault from "./layouts/LayoutDefault";

import Home from "./views/Home";

import Auth from "./views/Auth/Auth";

import Main from "./views/Main";

import Modal from "./components/elements/Modal";

import Privacy from "./views/Privacy";

import Pricing from "./views/Pricing";

ReactGA.initialize(process.env.REACT\_APP\_GA\_CODE);

const trackPage = (page) => {

  ReactGA.set({ page });

  ReactGA.pageview(page);

};

const App = () => {

  const childRef = useRef();

  let location = useLocation();

  const [user, setUser] = useState(JSON.parse(localStorage.getItem("profile")));

  useEffect(() => {

    const page = location.pathname;

    document.body.classList.add("is-loaded");

    childRef.current.init();

    trackPage(page);

    setUser(JSON.parse(localStorage.getItem("profile")));

  }, [location]);

  return (

    <>

      <ScrollReveal

        ref={childRef}

        children={() => (

          <Switch>

            {user ? (

              <AppRoute

                exact

                path="/"

                component={Main}

                layout={LayoutDefault}

              />

            ) : (

              <AppRoute exact path="/" component={Home} layout={LayoutDefault}/>

            )}

            {/\* <AppRoute exact path="/" component={Home} layout={LayoutDefault} /> \*/}

            <AppRoute

              exact

              path="/auth"

              component={Auth}

              layout={LayoutDefault}

            />

            {/\* <AppRoute

              exact

              path="/main"

              component={Main}

              layout={LayoutDefault}

            /> \*/}

            <AppRoute

              path="/anime/:video"

              component={Modal}

              layout={LayoutDefault}

            />

            <AppRoute path="/privacy" component={Privacy} layout={LayoutDefault} />

            <AppRoute path="/pricing" component={Pricing} layout={LayoutDefault} />

          </Switch>

        )}

      />

    </>

  );

};

export default App;

Client/src/App.test.js

import React from 'react';

import ReactDOM from 'react-dom';

import { Router } from 'react-router-dom';

import { createMemoryHistory } from 'history'

import App from './App';

const history = createMemoryHistory();

it('renders without crashing', () => {

  const div = document.createElement('div');

  ReactDOM.render(

    <Router history={history}>

      <App />

    </Router>,

    div

  );

  ReactDOM.unmountComponentAtNode(div);

});

Client/src/index.js

import React from "react";

import ReactDOM from "react-dom";

import { Router } from "react-router-dom";

import { createBrowserHistory } from "history";

import { Provider } from "react-redux";

import { createStore, applyMiddleware, compose } from "redux";

import thunk from "redux-thunk";

import { reducers } from "./reducers";

import App from "./App";

import \* as serviceWorker from "./serviceWorker";//import './App.css';

import "./assets/scss/style.scss";

const history = createBrowserHistory();

const store = createStore(reducers, compose(applyMiddleware(thunk)));

ReactDOM.render(

  <Provider store={store}>

    <Router history={history}>

      <App />

    </Router>

  </Provider>,

  document.getElementById("root")

);

serviceWorker.unregister();

Client/src/seviceWoker.js

const isLocalhost = Boolean(

  window.location.hostname === 'localhost' ||

    // [::1] is the IPv6 localhost address.

    window.location.hostname === '[::1]' ||

    // 127.0.0.1/8 is considered localhost for IPv4.

    window.location.hostname.match(

      /^127(?:\.(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)){3}$/

    )

);

export function register(config) {

  if (process.env.NODE\_ENV === 'production' && 'serviceWorker' in navigator) {// The URL constructor is available in all browsers that support SW.

    const publicUrl = new URL(process.env.PUBLIC\_URL, window.location.href);

    if (publicUrl.origin !== window.location.origin) {

      return;

    }

    window.addEventListener('load', () => {

      const swUrl = `${process.env.PUBLIC\_URL}/service-worker.js`;

      if (isLocalhost) {// This is running on localhost. Let's check if a service worker still exists or not.

        checkValidServiceWorker(swUrl, config);// Add some additional logging to localhost, pointing developers to the // service worker/PWA documentation.

        navigator.serviceWorker.ready.then(() => {

          console.log(

            'This web app is being served cache-first by a service ' +

              'worker. To learn more, visit https://bit.ly/CRA-PWA'

          );

        });

      } else {// Is not localhost. Just register service worker

        registerValidSW(swUrl, config);

      }

    });

  }

}

function registerValidSW(swUrl, config) {

  navigator.serviceWorker

    .register(swUrl)

    .then(registration => {

      registration.onupdatefound = () => {

        const installingWorker = registration.installing;

        if (installingWorker == null) {

          return;

        }

        installingWorker.onstatechange = () => {

          if (installingWorker.state === 'installed') {

            if (navigator.serviceWorker.controller) {// At this point, the updated precached content has been fetched,// but the previous service worker will still serve the older// content until all client tabs are closed.

              console.log(

                'New content is available and will be used when all ' +

                  'tabs for this page are closed. See https://bit.ly/CRA-PWA.'

              );// Execute callback

              if (config && config.onUpdate) {

                config.onUpdate(registration);

              }

            } else {

              // At this point, everything has been precached.

              // It's the perfect time to display a

              // "Content is cached for offline use." message.

              console.log('Content is cached for offline use.');// Execute callback

              if (config && config.onSuccess) {

                config.onSuccess(registration);

              }

            }

          }

        };

      };

    })

    .catch(error => {

      console.error('Error during service worker registration:', error);

    });

}

function checkValidServiceWorker(swUrl, config) {

  // Check if the service worker can be found. If it can't reload the page.

  fetch(swUrl)

    .then(response => {

      // Ensure service worker exists, and that we really are getting a JS file.

      const contentType = response.headers.get('content-type');

      if (

        response.status === 404 ||

        (contentType != null && contentType.indexOf('javascript') === -1)

      ) {

        // No service worker found. Probably a different app. Reload the page.

        navigator.serviceWorker.ready.then(registration => {

          registration.unregister().then(() => {

            window.location.reload();

          });

        });

      } else {

        // Service worker found. Proceed as normal.

        registerValidSW(swUrl, config);

      }

    })

    .catch(() => {

      console.log(

        'No internet connection found. App is running in offline mode.'

      );

    });

}

export function unregister() {

  if ('serviceWorker' in navigator) {

    navigator.serviceWorker.ready.then(registration => {

      registration.unregister();

    });

  }

}

Server/controllers/anime.js

import AnimeModal from "../models/anime.js";

export const getAnime = async (req, res) => {

    try {

        const result = await AnimeModal.find({});

        res.status(200).json( result );

    } catch (error) {

        res.status(500).json({ message: "Something went wrong" });

    }

}

Server/controllers/user.js

import bcrypt from "bcryptjs";

import jwt from "jsonwebtoken";

import UserModal from "../models/user.js";

export const signin = async (req, res) => {

  const { email, password } = req.body;

  try {

    const oldUser = await UserModal.findOne({ email });

    if (!oldUser)

      return res.status(404).json({ message: "User doesn't exist" });

    const isPasswordCorrect = await bcrypt.compare(password, oldUser.password);

    if (!isPasswordCorrect)

      return res.status(400).json({ message: "Invalid credentials" });

    const token = jwt.sign({ email: oldUser.email, id: oldUser.\_id }, "test", {

      expiresIn: "1h",

    });

    res.status(200).json({ result: oldUser, token });

  } catch (err) {

    res.status(500).json({ message: "Something went wrong" });

  }

};

export const signuphere = async (req, res) => {

  const { email, password, confirmpass, firstName, lastName } = req.body;

  try {

    const oldUser = await UserModal.findOne({ email });

    if (oldUser)

      return res.status(400).json({ message: "User already exists" });

    // if(password!==confirmpass) return res.status(400).json({ message: "Password Dosent Match" });

    const hashedPassword = await bcrypt.hash(password, 12);

    const result = await UserModal.create({

      email,

      password: hashedPassword,

      name: `${firstName} ${lastName}`,

    });

    const token = jwt.sign({ email: result.email, id: result.\_id }, "test", {

      expiresIn: "1h",

    });

    res.status(201).json({ result, token });

  } catch (error) {

    res.status(500).json({ message: "Something went wrong" });

    console.log(error);

  }

};

Server/models/user.js

import mongoose from "mongoose";

const userSchema = mongoose.Schema({

  name: { type: String, required:  true },

  email: { type: String, required: true },

  password: { type: String, required: true },

  id: { type: String },

});

export default mongoose.model("User", userSchema);

Server/models/anime.js

import mongoose from "mongoose";

const animeSchema = mongoose.Schema({

  name: { type: String, required:  true },

  thumbnail: { type: String, required: true },

  desc: { type: String, required: true },

  delay: { type: String },

  video: { type: String, required: true },

});

export default mongoose.model("Anime", animeSchema);

Server/routes/anime.js

import express from 'express';

import {getAnime} from '../controllers/anime.js';

const router = express.Router();

router.get('/', getAnime);

export default router;

Server/routes/user.js

import express from 'express';

import { signin,signuphere } from '../controllers/user.js';

const router = express.Router();

router.post("/signin", signin);

router.post("/signup", signuphere);

export default router;

Server/index.js

import express from 'express';

import bodyParser from 'body-parser';

import mongoose from 'mongoose';

import cors from 'cors';

import userRoutes from './routes/user.js';

import animeRoutes from './routes/anime.js';

const app = express();

app.use(bodyParser.json({ limit: '30mb', extended: true }))

app.use(bodyParser.urlencoded({ limit: '30mb', extended: true }))

app.use(cors());

app.use('/user',userRoutes);

app.use('/anime',animeRoutes)

const CONNECTION\_URL = 'mongodb+srv://shivam:shivam@cluster0.pmpu4.mongodb.net/AnimeFlixx?retryWrites=true&w=majority&appName=Cluster0';

const PORT = process.env.PORT || 5000;

mongoose.connect(CONNECTION\_URL, { useNewUrlParser: true, useUnifiedTopology: true })

  .then(() => app.listen(PORT, () => console.log(`Server Running on Port: http://localhost:${PORT}`)))

  .catch((error) => console.log(`${error} did not connect`))