****

**SE3040**

**Application Frameworks (AF)**

**Makeup Exam - 2021**

**Student ID : IT18149654**

**Student Name : Rajapaksha T.N.**

**Category : Online Learning**

**Scenario**

iLearn is an online learning system. Online learning systems are very popular these days due to the due to the social distancing measures in this post Covid-19 period.

The basic idea of this web application is to provide a platform for teachers and students to continue their teaching and learning processes online through a quizzing format. This simple web application allows the teachers to add quizzes with several questions and short answers relevant to different subjects or topics. The students can try the questions out and the results will be provided immediately after the submission of the answers.

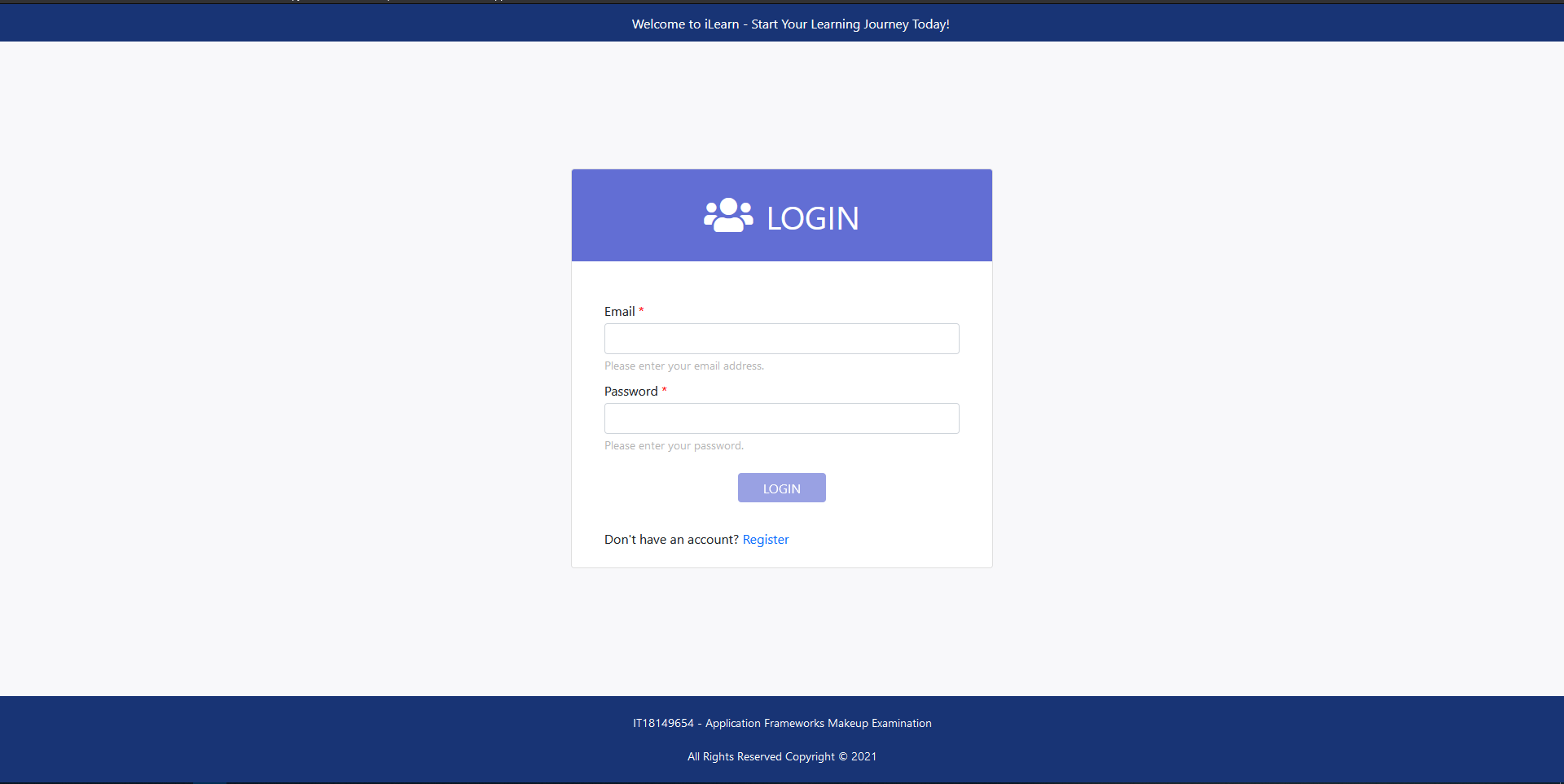
There are so many video streaming platforms these days to connect the teachers and students for delivering lectures and lessons. But there are not many applications for providing this service, the iLearn web application is providing. This question-answer format of learning is very important for students of any level and any age to really grasp the content of the learned material. This iLearn application provides a very simple and basic version of this service.

There are two main actors in this scenario.

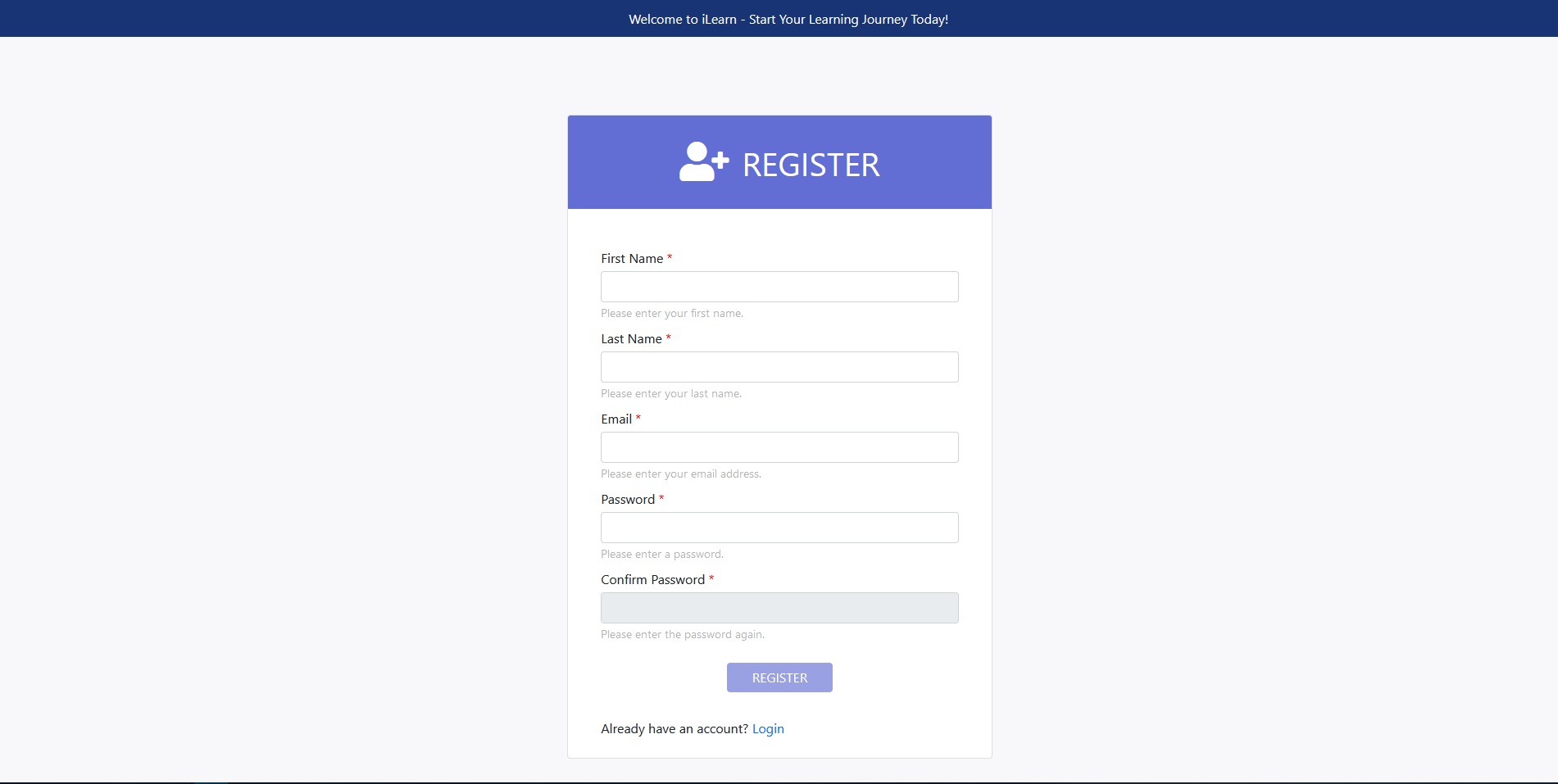
1. Teacher
2. Student

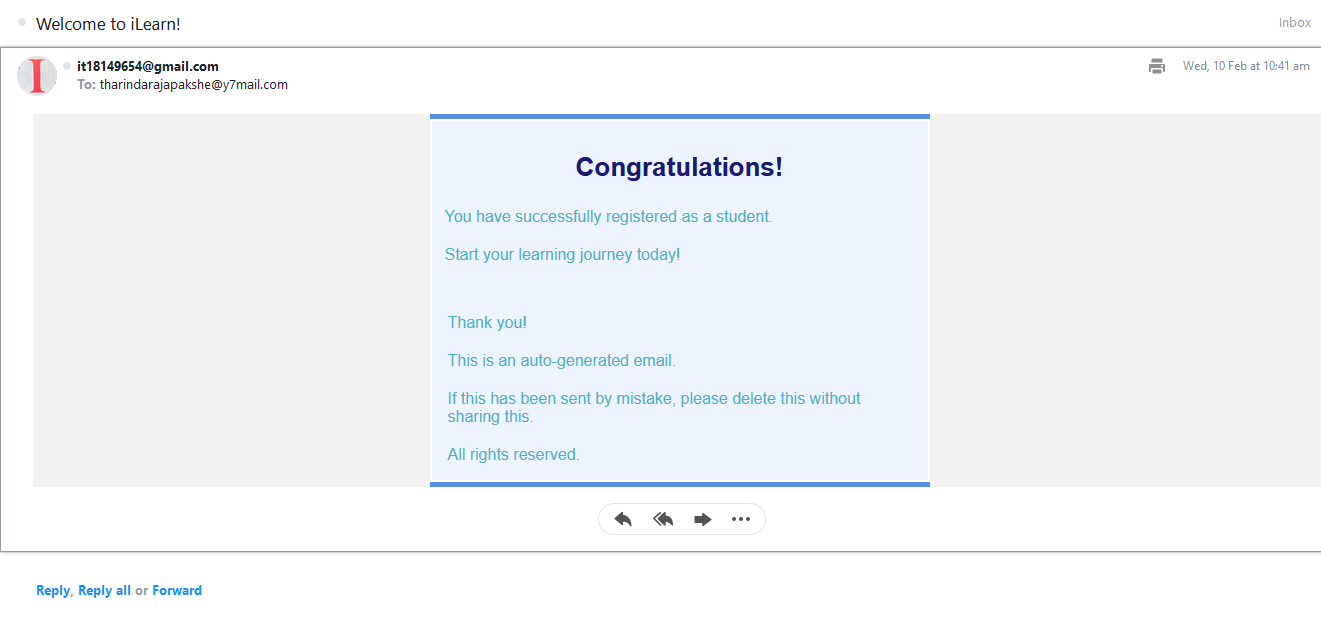
All the users use one login page to login to the system. System can identify the logged in user’s type and provide only the permitted functionality for that user type. The navigation bar will be updated accordingly as well.

All the pages of the application are properly secured so that no unauthorized user can access them without the necessary permissions being granted.



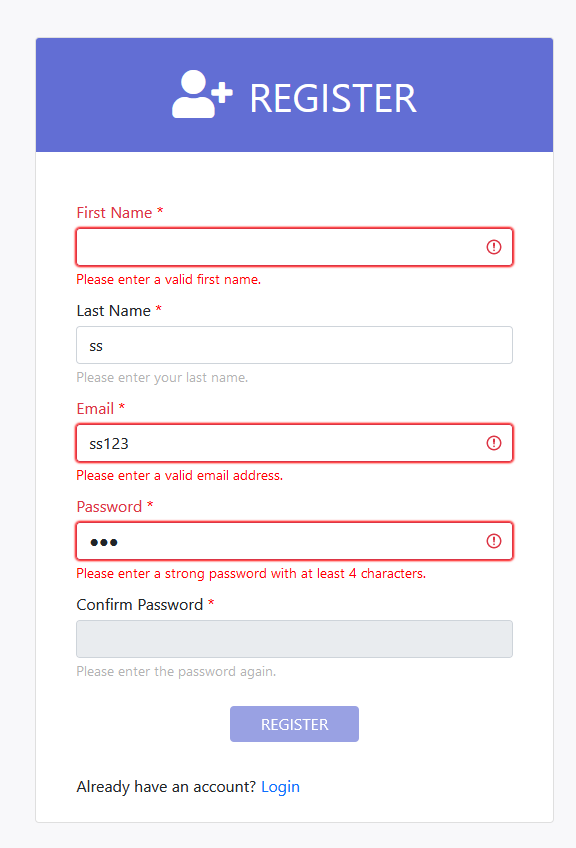
The new students can register to the system and create a new account. They will also receive an email confirming their registration.



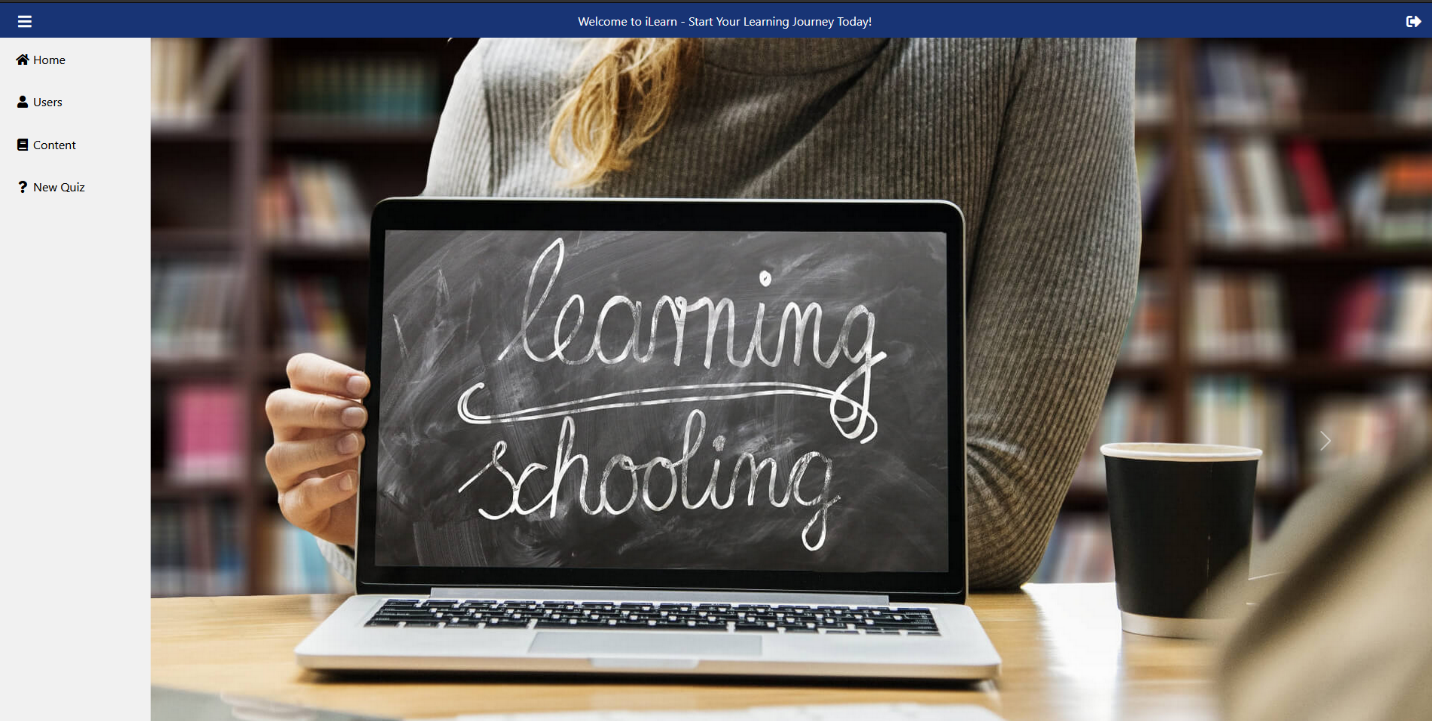


* Student Login:
  + Method 1: Create a student account with 'Register' page to login to the system as a student.
  + Method 2: Use the following login credentials of an already created student account to login to the system as a student.
    - Email : student@gmail.com
    - Password : student
* Teacher Login:
  + Use the following login credentials of an already created teacher account to login to the system as a teacher.
    - Email : teacher@gmail.com
    - Password : teacher

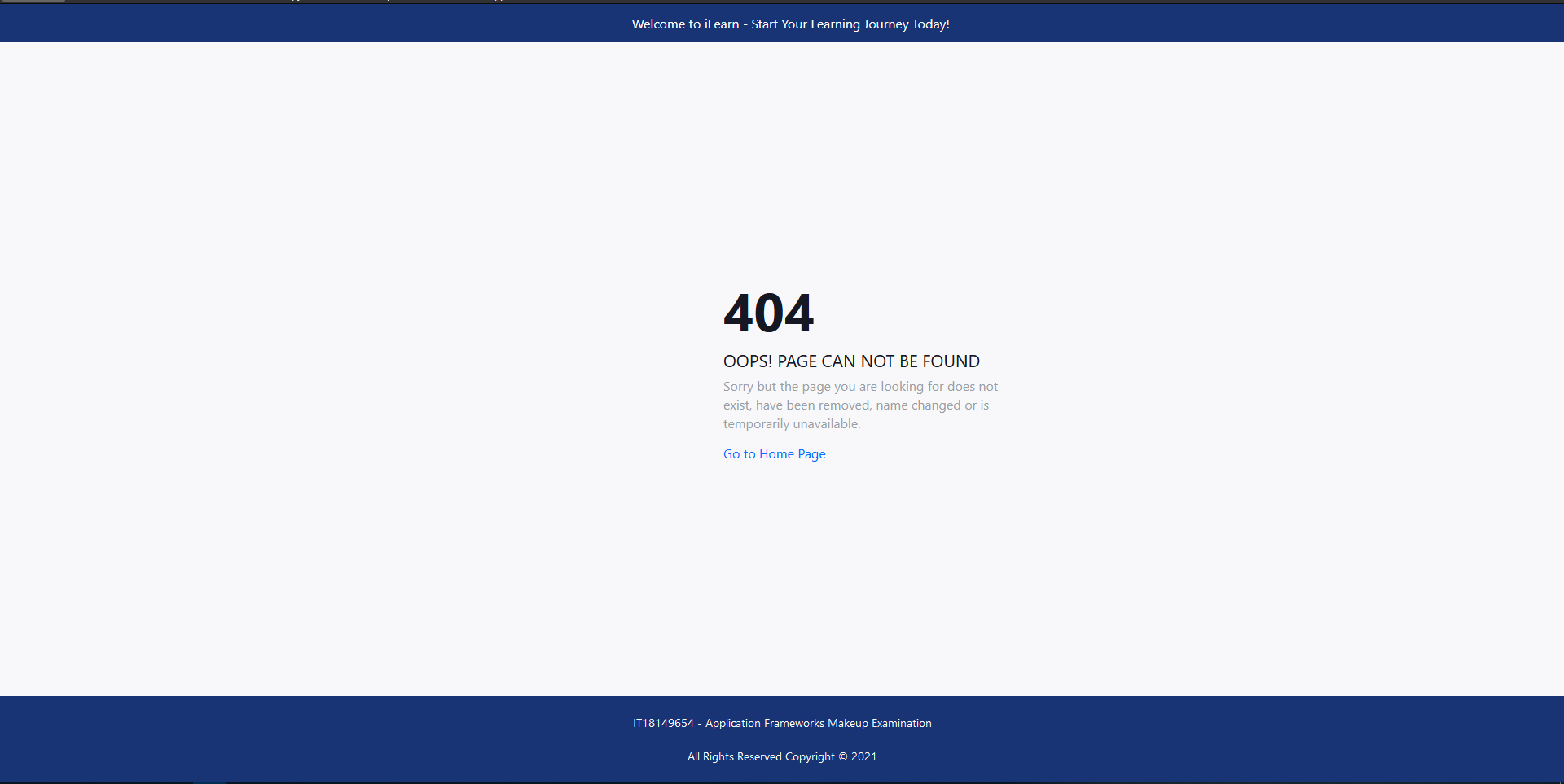
All the forms and input actions in this web application are properly validated so that no invalid data will be saved in the database. Proper validation error messages which are user friendly are also displayed in case of scenarios such as input pattern errors or unique primary key violations etc.



When logged in as the teacher, only the functionalities allowed for that user type is allowed.

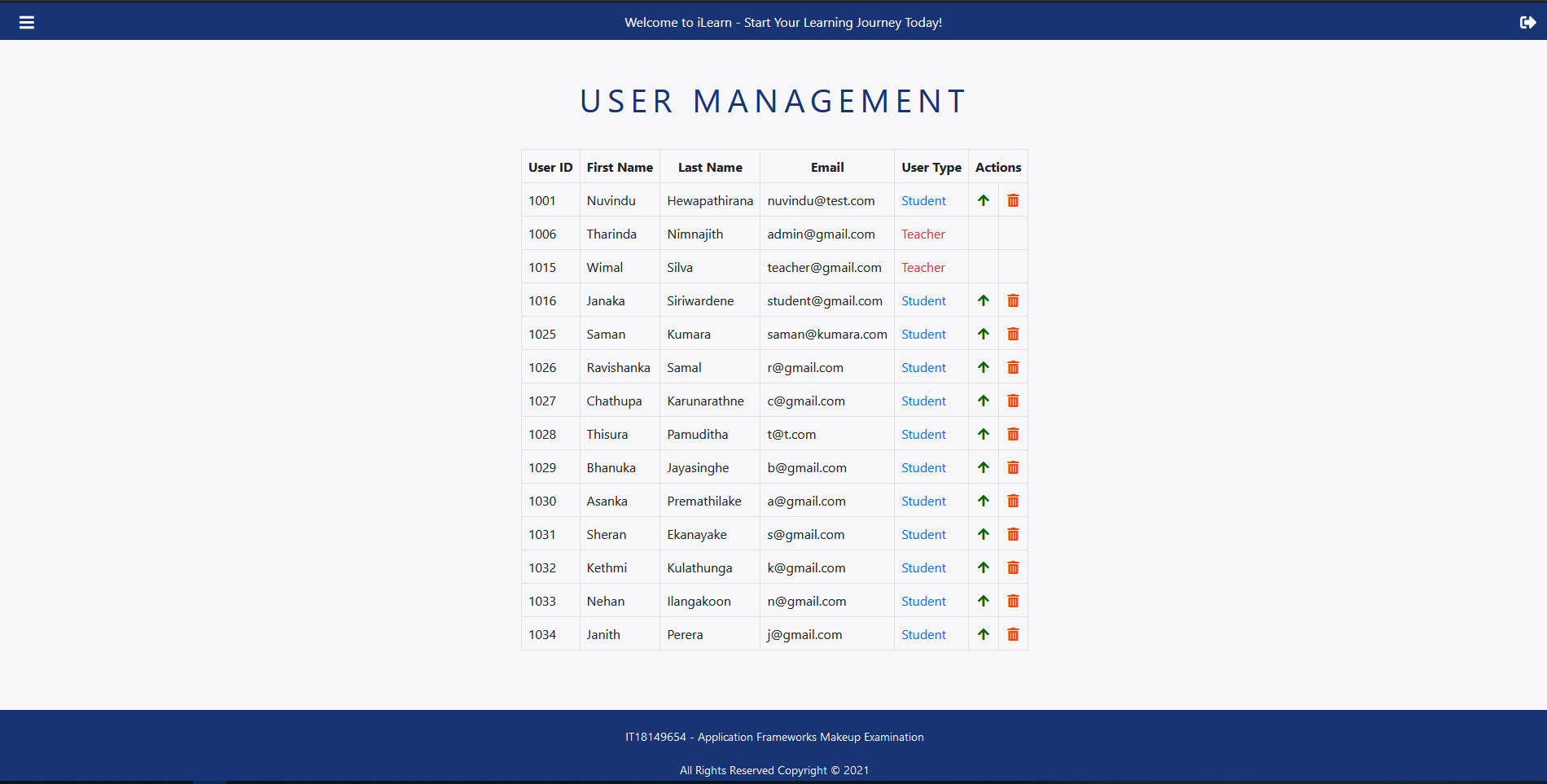


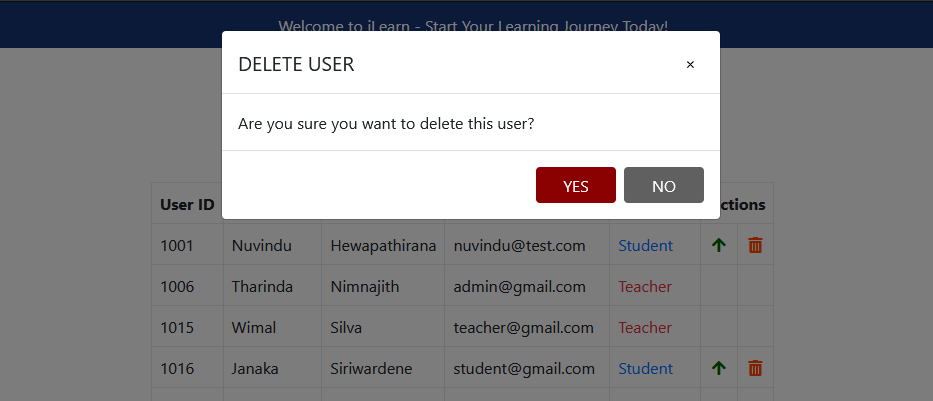
If the user tries to access an undefined route a user friendly 404 error message page is displayed.

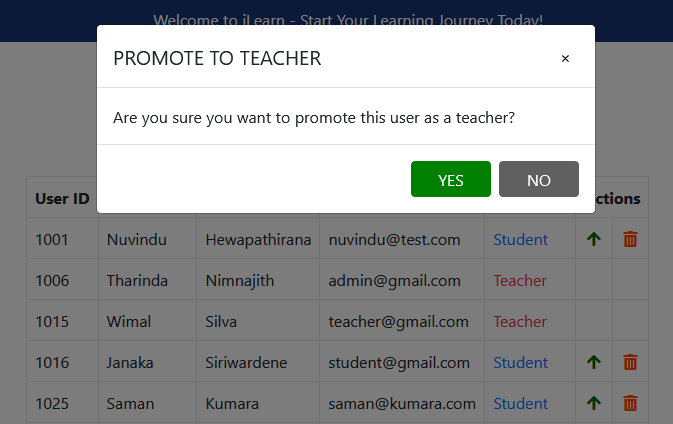


If the user tries to access an existing but not authorized page, the user will be redirected to login page or the relevant home page based on their user type and login status. Further, the logout functionality also works properly.

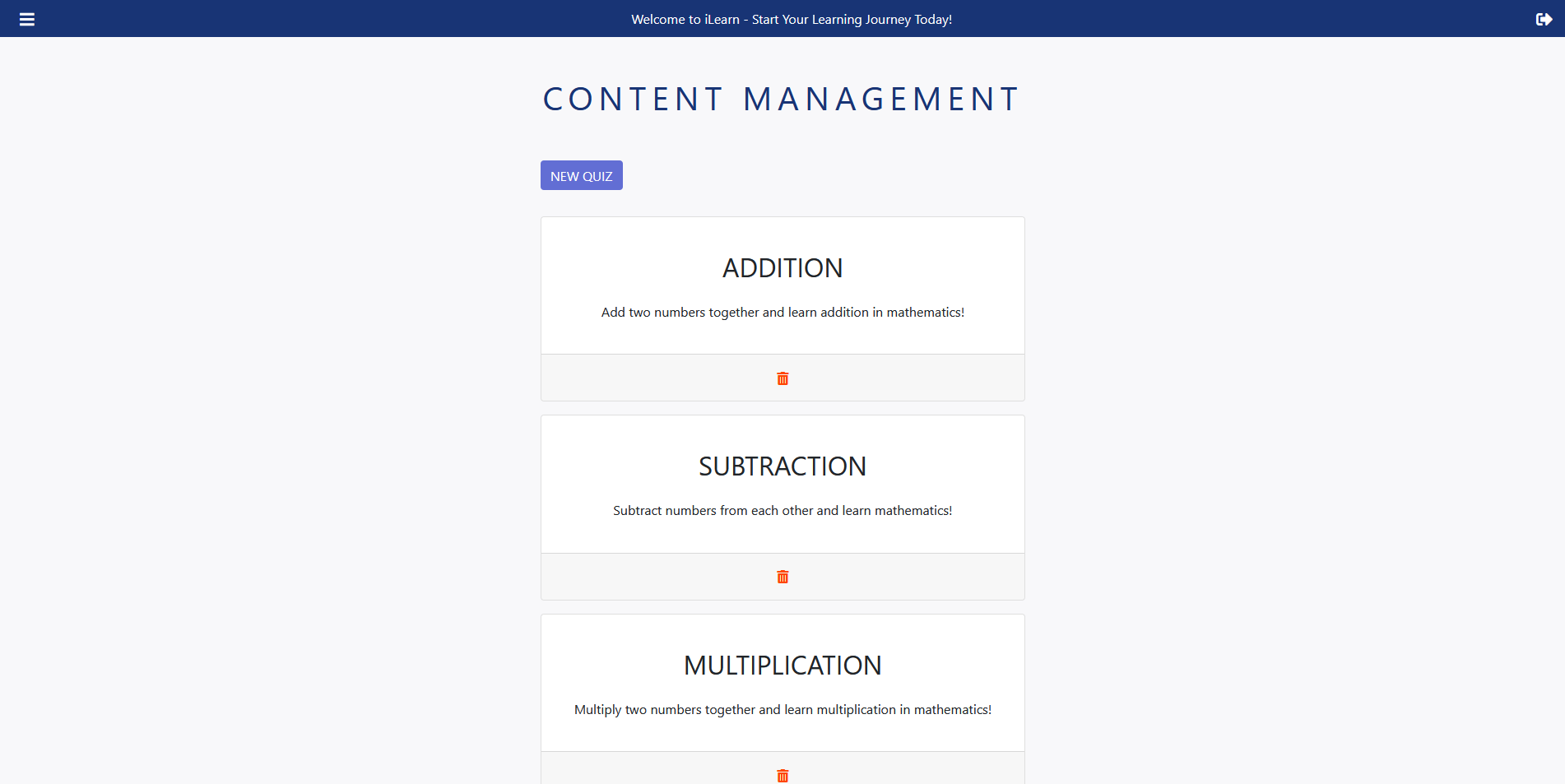
The teacher can view all users of the system. Teacher is allowed to delete any student account from the system as well as promote a student user as a teacher. As these are some critical actions, the system will ask to confirm the action prior to performing it.







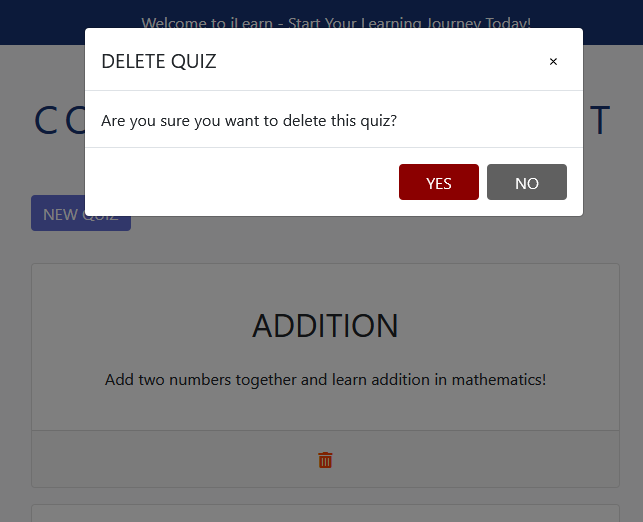
The teacher can view the content / quizzes currently in the system under content management.

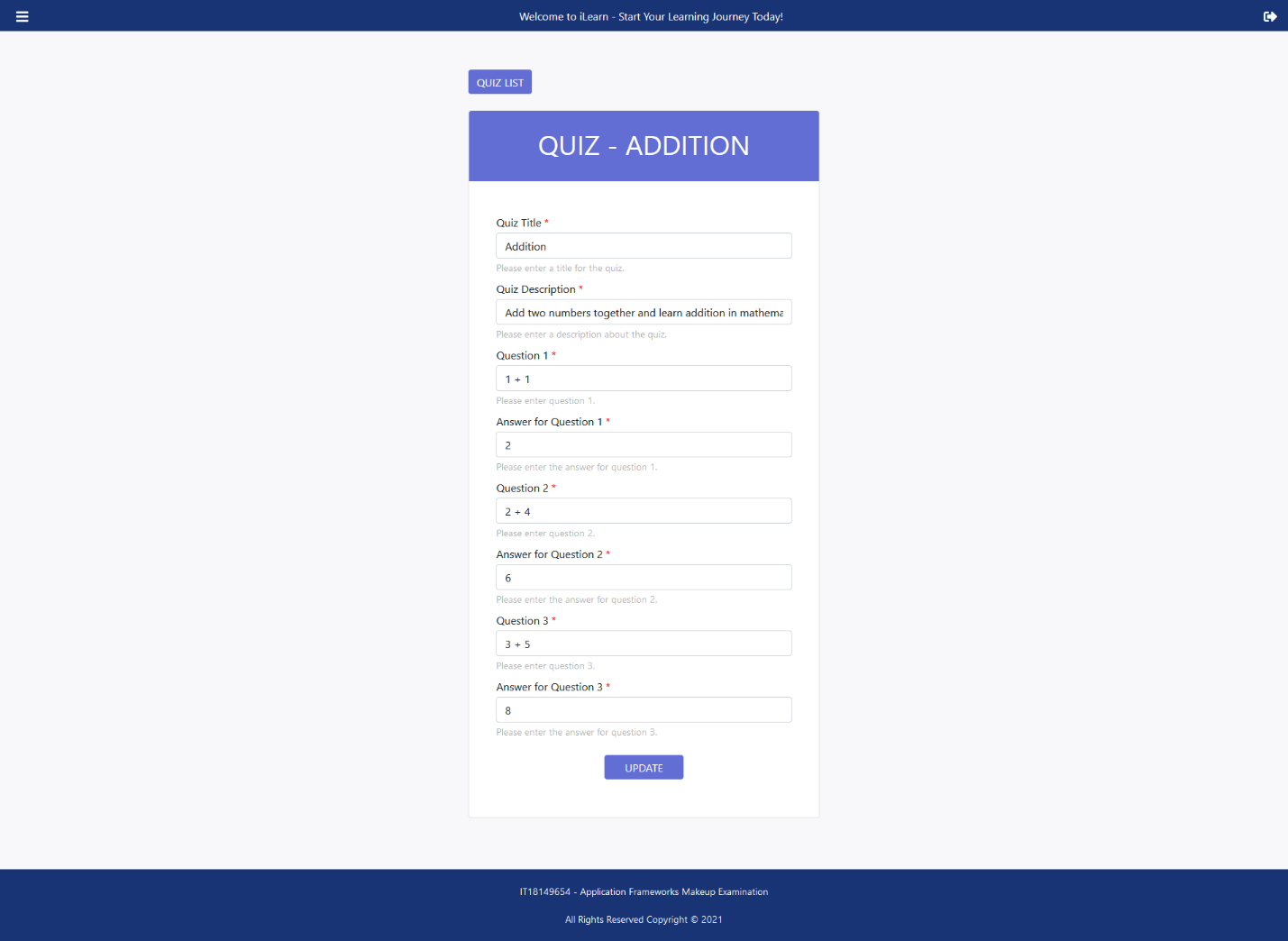


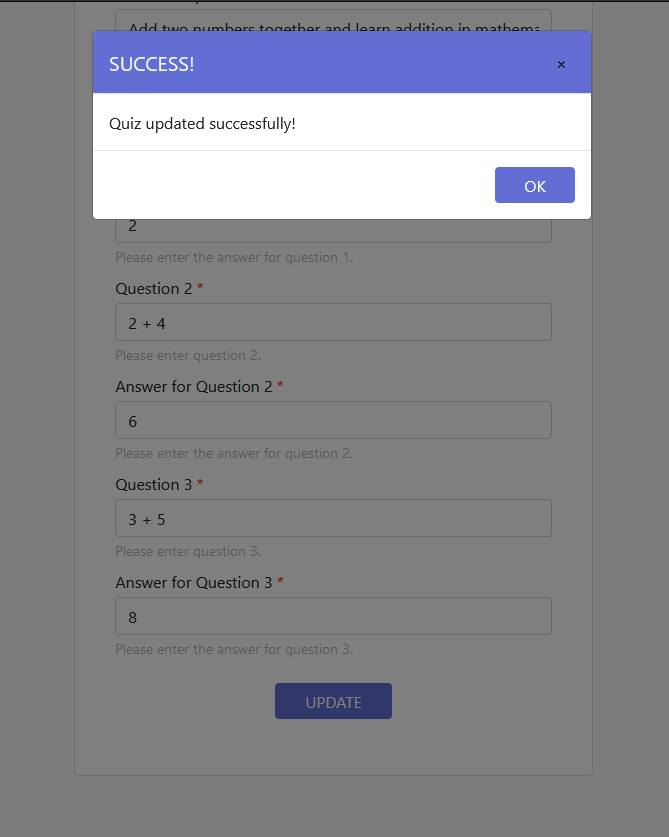
The teacher can delete the quizzes in the list if necessary. The teacher can add new quizzes as well as view a single quiz and update it.

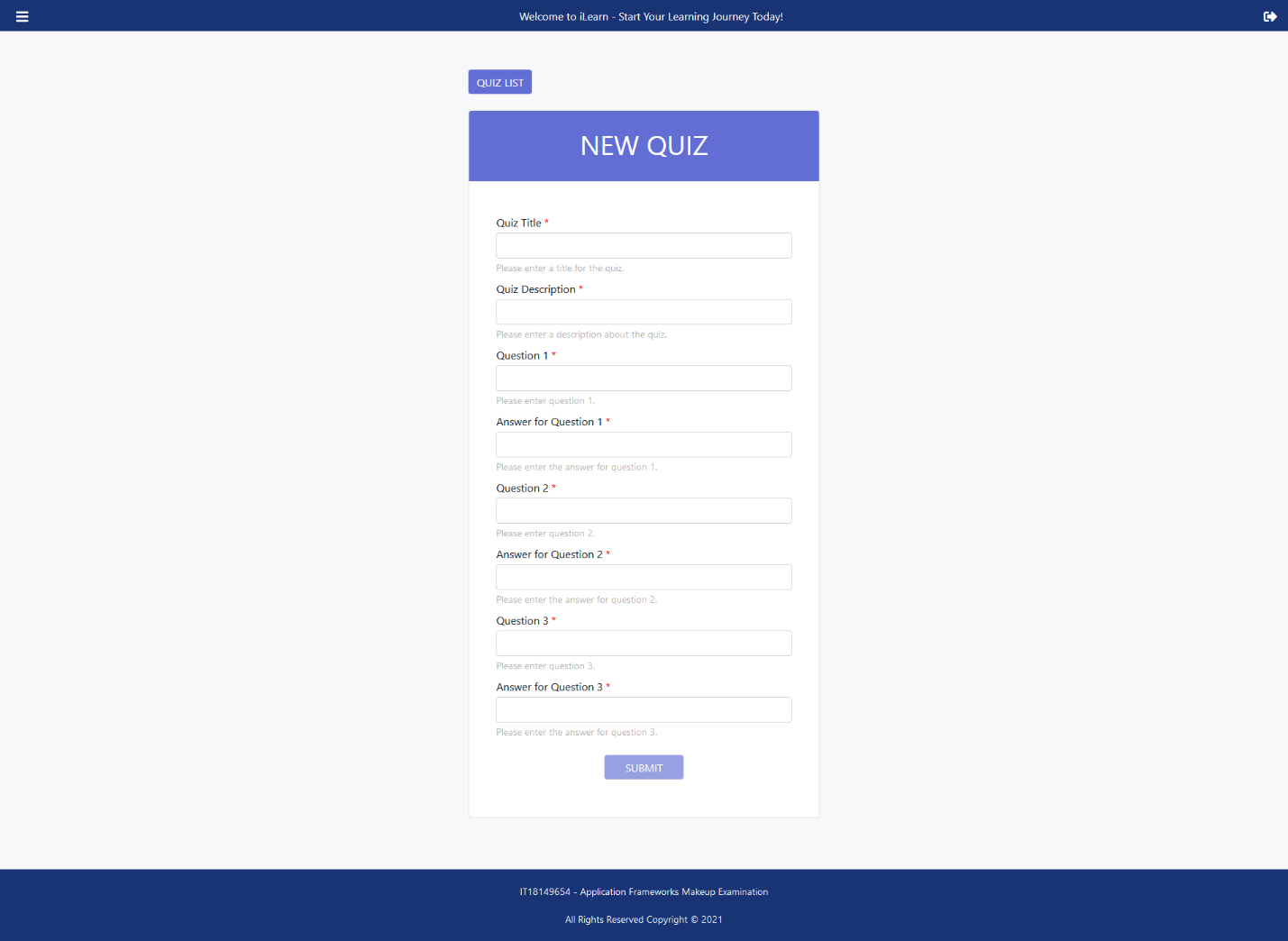
For the simplicity purpose, a quiz contains of three questions and their answers. Clicking on a list item will navigate the teacher to see more details of that specific quiz.

The navigation to new quiz and quiz list can be done in multiple ways to increase the usability of the application.



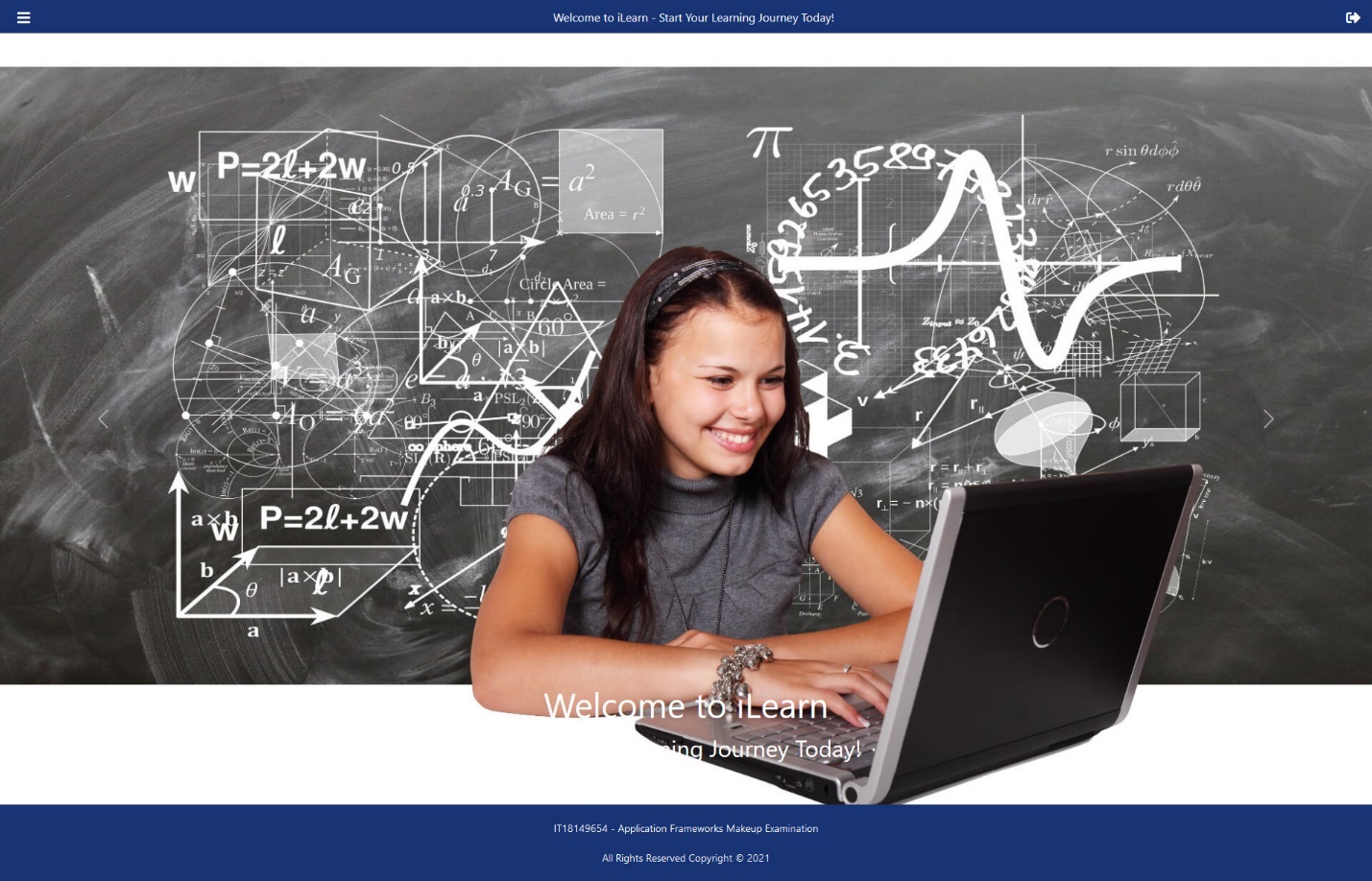


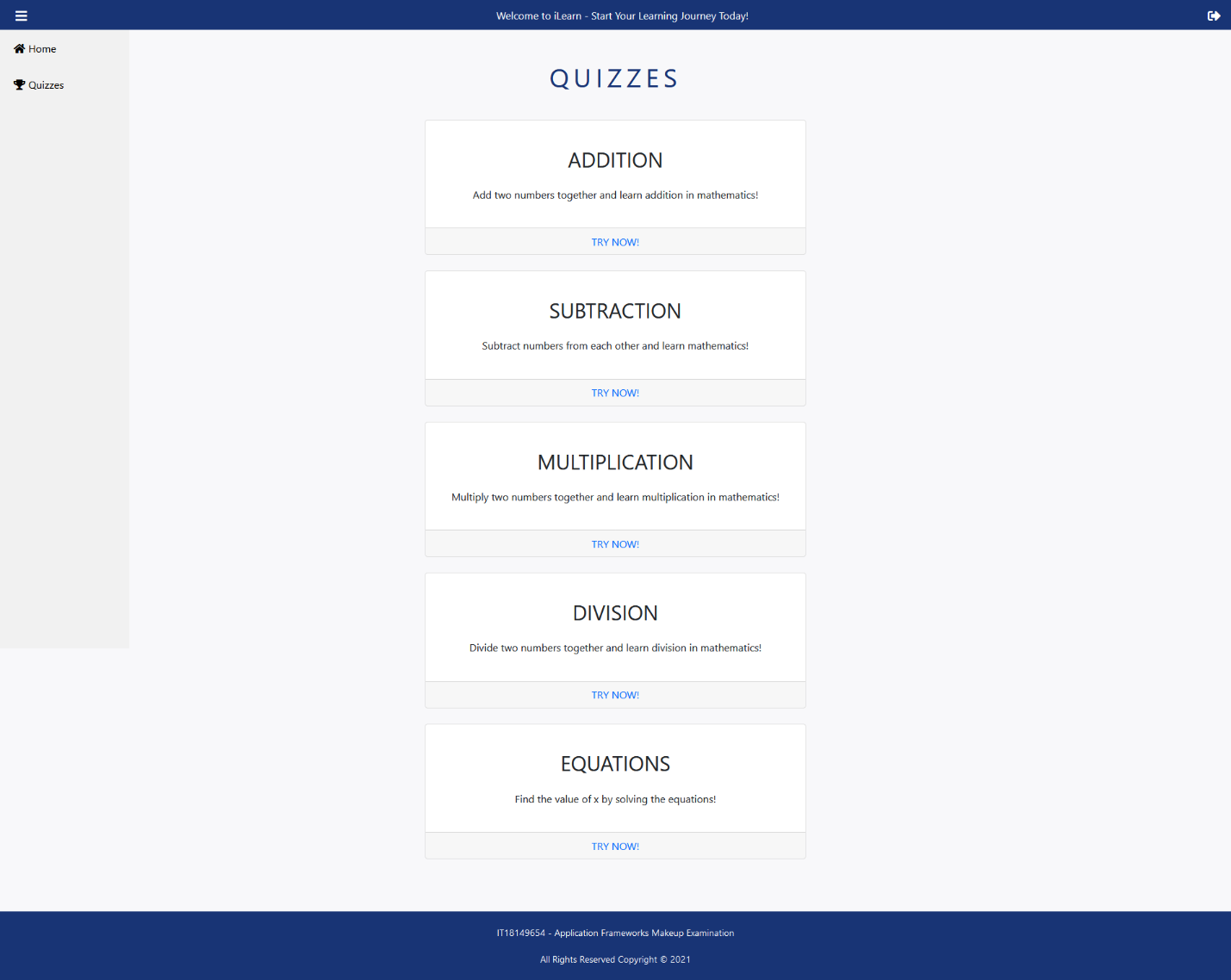


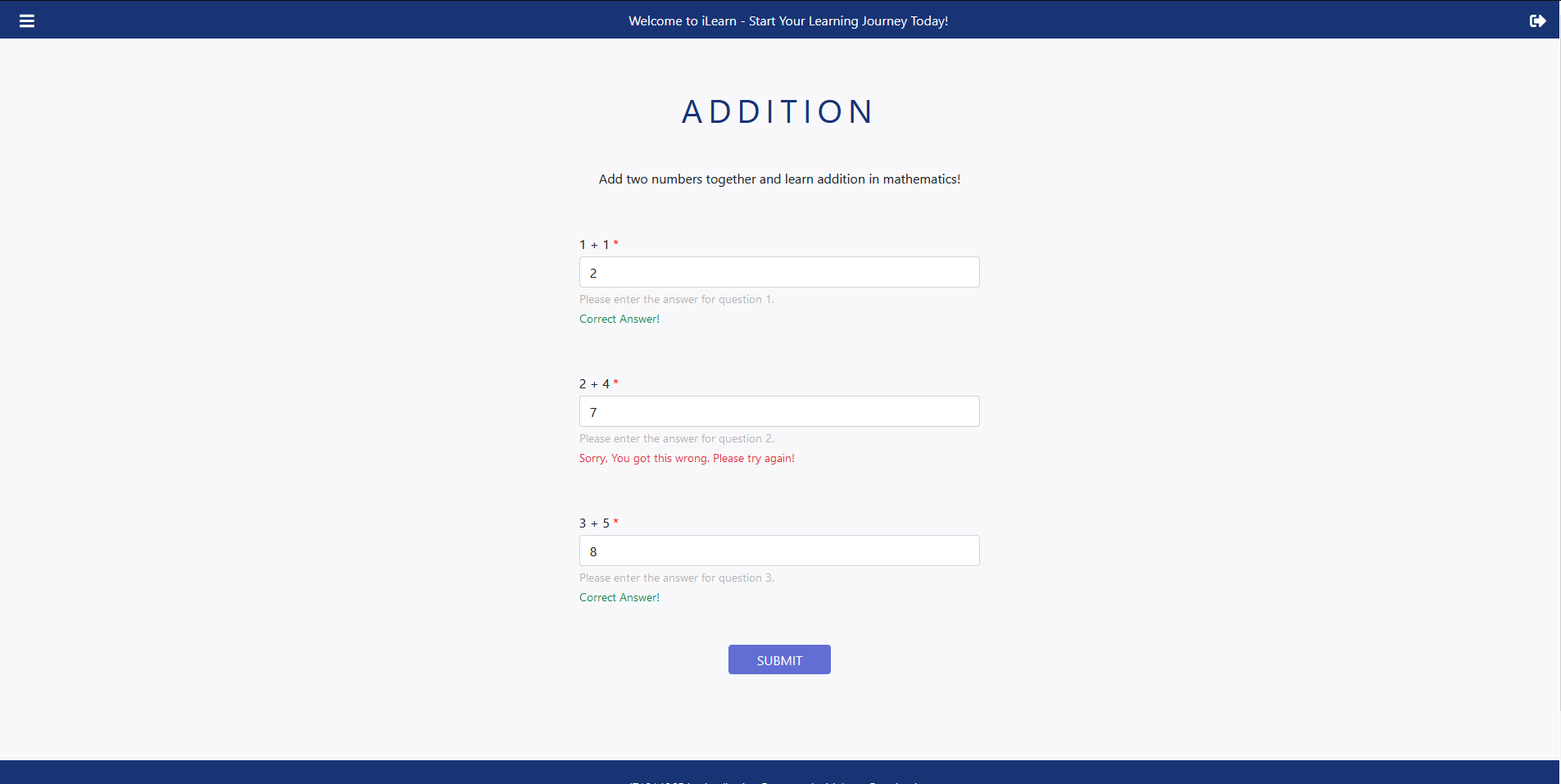


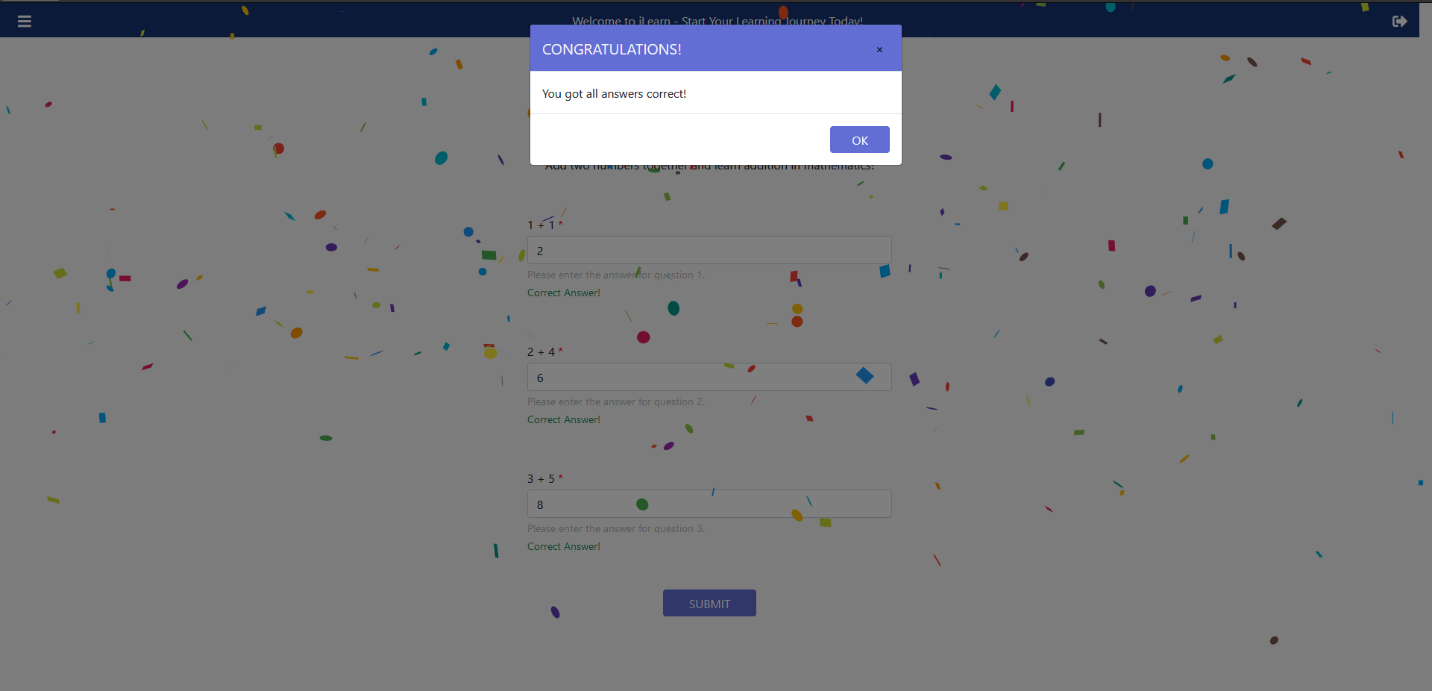
When logged in as the student, only the functionalities allowed for that user type is allowed.

The student can view the quizzes added by teachers and practice them. Immediately after the answers are submitted the results are displayed.





****

****

**Implementation main functionalities (both frontend and backend)**

1. **Login, Logout and Register (Student & Teacher)**
2. **User Management (Teacher)**
3. **Quiz Management (Teacher)**
4. **Playing Quizzes (Student)**
5. **1. Login, Logout and Register – Backend**

**users.model.js**

*const* mongoose = require('mongoose')  
*const* autoIncrement = require('mongoose-auto-increment')  
*const uniqueValidator* = require('mongoose-unique-validator')  
  
*const* Schema = mongoose.Schema  
  
*const* userTypes = [  
 'Admin',  
 'User'  
]  
  
*const* UsersSchema = *new* Schema({  
 userId: {  
 type: Number,  
 required: *false*,  
 unique: *true*,  
 trim: *true* },  
 firstName: {  
 type: String,  
 required: *false*,  
 unique: *false*,  
 trim: *true* },  
 lastName: {  
 type: String,  
 required: *false*,  
 unique: *false*,  
 trim: *true* },  
 email: {  
 type: String,  
 required: *true*,  
 unique: *true*,  
 trim: *true* },  
 password: {  
 type: String,  
 required: *true*,  
 unique: *false*,  
 trim: *true* },  
 userType: {  
 type: String,  
 *enum*: userTypes,  
 required: *false*,  
 unique: *false*,  
 trim: *true*,  
 *default*: 'User'  
 }  
}, {  
 timestamps: *true*,  
 collection: 'Users'  
})  
  
UsersSchema.plugin(*uniqueValidator*)  
  
autoIncrement.initialize(mongoose.connection)  
  
UsersSchema.plugin(autoIncrement.plugin, {  
 model: 'Users',  
 field: 'userId',  
 startAt: 1000,  
 incrementBy: 1  
})  
  
module.exports = mongoose.*model*('Users', UsersSchema)

**auth.routes.js**

*const* express = require('express')  
*const* AuthController = require('../controllers/auth-controller')  
  
*const* router = express.*Router*()  
  
router.post('/login', AuthController.login)  
  
module.exports = router

**users-controller.js**

*const* bcrypt = require('bcrypt')  
*const* nodemailer = require('nodemailer')  
*const* UserModel = require('../models/users.model')  
  
require('dotenv').*config*()  
  
*const* addUser = *async* (req, res) => {  
 *let* existingUser  
  
 *let* {  
 firstName,  
 lastName,  
 email,  
 password  
 } = req.body  
  
 *try* {  
 existingUser = *await* UserModel.findOne({  
 email: email  
 })  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 *if* (existingUser) {  
 res.send({  
 status: 409,  
 message: 'A user with the same email already exists.'  
 })  
 }  
  
 *const* newUser = *new* UserModel({  
 firstName,  
 lastName,  
 email,  
 password: bcrypt.hashSync(password, 10)  
 })  
  
 *try* {  
 *await* newUser.save()  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 *await* sendEmail(email)  
  
 res.send({  
 status: 201,  
 message: 'Congratulations! You have successfully registered as a student in iLearn. ' +  
 'Now please login to iLearn and start your learning journey right now!'  
 })  
}  
  
*const* addAdmin = *async* (req, res) => {  
 *let* existingUser  
  
 *let* {  
 firstName,  
 lastName,  
 email,  
 password  
 } = req.body  
  
 *try* {  
 existingUser = *await* UserModel.findOne({  
 email: email  
 })  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 *if* (existingUser) {  
 res.send({  
 status: 409,  
 message: 'A user with the same email already exists.'  
 })  
 }  
  
 *const* newUser = *new* UserModel({  
 firstName,  
 lastName,  
 email,  
 password: bcrypt.hashSync(password, 10),  
 userType: 'Admin'  
 })  
  
 *try* {  
 *await* newUser.save()  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 res.send({  
 status: 201,  
 message: 'Administrator added successfully!'  
 })  
}  
  
*let* transporter = nodemailer.createTransport({  
 service: 'gmail',  
 auth: {  
 user: 'it18149654@gmail.com',  
 pass: process.env.PASSWORD  
 }  
})  
  
*const* sendEmail = *async* email => {  
 *let* info = {  
 from: 'it18149654@gmail.com',  
 to: email,  
 subject: 'Welcome to iLearn!',  
 text:  
 `Congratulations!  
 You have successfully registered as a student.  
 Start your learning journey today!  
 Thank you!  
 This is an auto-generated email.  
 If this has been sent by mistake, please delete this without sharing this.  
 All rights reserved.`,  
 html:  
 `<div style="margin: 0; padding: 0; background-color: #f2f2f2; font-family: arial, serif;">  
 <table style="margin: 0 auto; background: white; max-width: 500px; padding-bottom: 0; border-top: 5px solid #588dde; border-bottom: 5px solid #588dde; width: 100%;">  
 <tr style="background: rgb(237, 243, 255); padding-left: 20px; padding-right: 20px;">  
 <td>  
 <table align="left" style="width: 100%;">  
 <tr>  
 <td style="padding: 10px;">  
 <h1 style="text-align: center; color: #1a1a72;">Congratulations!</h1>  
 <h2 style="margin-top:25px; margin-bottom: 0; color: #4db0c4; font-weight: 400; font-size: medium;">You have successfully registered as a student.</h2>  
 <h2 style="margin-top:20px; margin-bottom: 0; color: #4db0c4; font-weight: 400; font-size: medium;">Start your learning journey today!</h2>  
 </td>  
 </tr>  
 <tr style="background: rgb(237, 243, 255); padding-left: 20px; padding-right: 20px;">  
 <td>  
 <table align="left" style="width: 100%;">  
 <tr>  
 <td style="padding: 10px;">  
 <h2 style="margin-top:25px; margin-bottom: 0; color: #4db0c4; font-weight: 400; font-size: medium;">Thank you!</h2>  
 <h4 style="margin-top:20px; margin-bottom: 0; color: #4db0c4; font-weight: 400; font-size: medium;">This is an auto-generated email.</h4>  
 <h4 style="margin-top:20px; margin-bottom: 0; color: #4db0c4; font-weight: 400; font-size: medium;">If this has been sent by mistake, please delete this without sharing this.</h4>  
 <h4 style="margin-top:20px; margin-bottom: 0; color: #4db0c4; font-weight: 400; font-size: medium;">All rights reserved.</h4>  
 </td>  
 </tr>  
 </table>  
 </td>  
 </tr>  
 </table>  
 </div>`  
 }  
  
 transporter.sendMail(info, (error, data) => {  
 *if* (error) {  
 *console*.error(error)  
 *console*.error('Email sending failed! Please try again.')  
 } *else* {  
 *console*.error(data)  
 *console*.error('An email is sent successfully to ' + email + '.')  
 }  
 })  
}

**1.2 Login, Logout and Register – Frontend**

**route-filter.jsx**

*import* React, {*useContext*, *useEffect*, *useState*} *from* 'react'  
*import* {Redirect, Route} *from* 'react-router-dom'  
*import* {admin, all, user} *from* '../constants/enumerations/user-types'  
*import* {AppContext} *from* '../global/app-context'  
*import* {*checkUserInLocalStorage*} *from* '../helpers/local-storage.helpers'  
*import Loader from* '../components/loader/loader'  
*import* './route-filter.css'  
  
*const RouteFilter* = (  
 {  
 component: Component,  
 needAuthentication,  
 userType,  
 ...rest  
 }  
) => {  
 *const* appContext = *useContext*(AppContext)  
  
 *const* [authenticated, setAuthenticated] = *useState*(*null*)  
  
 *useEffect*(() => {  
 *const* localeStorageData = *checkUserInLocalStorage*()  
 *if* (localeStorageData.status === *true*) {  
 appContext.login(localeStorageData.result).then(() => {  
 })  
 setAuthenticated(*true*)  
 } *else* {  
 setAuthenticated(*false*)  
 }  
 }, [Component])  
  
 *return* (  
 <div>  
 <Route {...rest}  
 *render*={  
 props => {  
 *if* (needAuthentication && authenticated === *null*) {  
 *return* (  
 <Loader/>  
 )  
 } *else if* (needAuthentication && !authenticated) {  
 *return* (  
 <Redirect *to*={'/login'}/>  
 )  
 } *else if* (!needAuthentication && authenticated) {  
 *if* (appContext.loginData && appContext.loginData.userType === admin) {  
 *return* (  
 <Redirect *to*={'/dashboard'}/>  
 )  
 } *else if* (appContext.loginData && appContext.loginData.userType === user) {  
 *return* (  
 <Redirect *to*={'/home'}/>  
 )  
 }  
 } *else if* (!needAuthentication) {  
 *return* (  
 <Component {...props} />  
 )  
 } *else if* (authenticated) {  
 *if* (userType === all || (appContext.loginData && appContext.loginData.userType === userType)) {  
 *return* (  
 <Component {...props} />  
 )  
 } *else* {  
 *if* (appContext.loginData && appContext.loginData.userType === admin) {  
 *return* (  
 <Redirect *to*={'/dashboard'}/>  
 )  
 } *else if* (appContext.loginData && appContext.loginData.userType === user) {  
 *return* (  
 <Redirect *to*={'/home'}/>  
 )  
 }  
 }  
 } *else* {  
 *return* (  
 <Loader/>  
 )  
 }  
 }}/>  
 </div>  
 )  
}  
  
*export default RouteFilter*

**local-storage.helpers.js**

*import* {authStoreKey} *from* '../config/main.config'  
  
*const setLocalStorageItem* = *async* (key, obj) => {  
 *try* {  
 *await* localStorage.setItem(key, JSON.stringify(obj))  
 *return true* } *catch* (error) {  
 *console*.error(error)  
 *return false* }  
}  
  
*const removeFromLocalStorage* = *async* key => {  
 *try* {  
 *await* localStorage.removeItem(key)  
 *return true* } *catch* (error) {  
 *console*.error(error)  
 *return false* }  
}  
  
*const checkUserInLocalStorage* = () => {  
 *try* {  
 *const* data = getFromLocalStorage(authStoreKey)  
 *if* (data) {  
 *return* {  
 status: *true*,  
 result: data  
 }  
 } *else* {  
 *return* {  
 status: *false* }  
 }  
 } *catch* (error) {  
 *console*.error(error)  
 *return* {  
 status: *false* }  
 }  
}  
  
*const* getFromLocalStorage = key => {  
 *try* {  
 *return* JSON.parse(localStorage.getItem(key))  
 } *catch* (error) {  
 *console*.error(error)  
 *return false* }  
}  
  
*export* {  
 *setLocalStorageItem*,  
 *removeFromLocalStorage*,  
 *checkUserInLocalStorage*}

**common.helpers.js**

*const isEmpty* = *async* value => {  
 *return* value === '' || value === *null* || value === undefined || value === 'null' || value === 'undefined'  
}  
  
*const dateToString* = *async* (value, format = 'dd-mm-YY') => {  
 *const* dateObj = *new* Date(value)  
 *switch* (format) {  
 *case* 'dd-mm-YY':  
 *return* `${dateObj.getDate().toString().padStart(2, '0')}-${(dateObj.getMonth() + 1)  
 .toString().padStart(2, '0')}-${dateObj.getFullYear().toString()}`  
 *default*:  
 *return* value  
 }  
}  
  
*export* {  
 *isEmpty*,  
 *dateToString*}

**app-context.js**

*import* {*createContext*} *from* 'react'  
  
*export const* AppContext = *createContext*({  
 loginData: *null*,  
 login: *async* () => {  
 },  
 logout: *async* () => {  
 }  
})

**global-state.jsx**

*import* React, {*useState*} *from* 'react'  
*import* {AppContext} *from* '../app-context'  
*import* './global-state.css'  
  
*export const GlobalState* = props => {  
 *const* [loginData, setLoginData] = *useState*(*null*)  
  
 *const* login = *async* data => {  
 setLoginData(data)  
 }  
  
 *const* logout = *async* () => {  
 setLoginData(*null*)  
 }  
  
 *return* (  
 <AppContext.Provider *value*={{  
 loginData: loginData,  
 login: login,  
 logout: logout  
 }}>  
 {props.children}  
 </AppContext.Provider>  
 )  
}

**user-types.js**

*export const* all = 'All'  
*export const* admin = 'Admin'  
*export const* user = 'User'

**header.jsx**

*import* React, {Fragment, *useContext*, *useState*} *from* 'react'  
*import* {*withRouter*} *from* 'react-router-dom'  
*import* {Navbar, NavbarBrand} *from* 'reactstrap'  
*import* {*removeFromLocalStorage*} *from* '../../helpers/local-storage.helpers'  
*import* {authStoreKey} *from* '../../config/main.config'  
*import* {AppContext} *from* '../../global/app-context'  
*import NavigationBar from* '../navigation-bar/navigation-bar'  
*import* './header.css'  
  
*const* Header = props => {  
 *const* appContext = *useContext*(AppContext)  
  
 *const* [display, setDisplay] = *useState*(*false*)  
  
 *const* onLogout = *async* () => {  
 *await removeFromLocalStorage*(authStoreKey)  
 *await* appContext.logout()  
 props.history.push('/login')  
 }  
  
 *const* onNavBarDisplay = *async* () => {  
 setDisplay(!display)  
 }  
  
 *return* (  
 <div>  
 <div>  
 <Fragment>  
 <Navbar *className*='header d-flex justify-content-between w-100'  
 *expand*='md'>  
 <div *className*={appContext.loginData === *null* ? 'invisible' : ''}>  
 <NavbarBrand>  
 <i *className*='icon fas fa-bars ms-4'  
 *onClick*={onNavBarDisplay}/>  
 </NavbarBrand>  
 </div>  
 <div>  
 <label *className*='logo mb-0'>  
 Welcome to iLearn - Start Your Learning Journey Today!  
 </label>  
 </div>  
 <div *className*={appContext.loginData === *null* ? 'invisible' : ''}>  
 <NavbarBrand>  
 <i *className*='icon fas fa-sign-out-alt'  
 *title*='Logout'  
 *onClick*={onLogout}/>  
 </NavbarBrand>  
 </div>  
 </Navbar>  
 </Fragment>  
 </div>  
 <div>  
 {  
 display ? (  
 <NavigationBar/>  
 ) : *null* }  
 </div>  
 </div>  
 )  
}  
  
*export default withRouter*(Header)

**navigation-bar.jsx**

*import* React, {*useContext*} *from* 'react'  
*import* {Nav} *from* 'reactstrap'  
*import* {AppContext} *from* '../../global/app-context'  
*import UserNavigationEntries from* './user-navigation-entries/user-navigation-entries'  
*import AdminNavigationEntries from* './admin-navigation-entries/admin-navigation-entries'  
*import* './navigation-bar.css'  
  
*const NavigationBar* = () => {  
 *const* appContext = *useContext*(AppContext)  
  
 *return* (  
 <div *className*='sidebar'>  
 {  
 appContext.loginData && appContext.loginData.userType === 'User' ? (  
 <div>  
 <UserNavigationEntries/>  
 </div>  
 ) : appContext.loginData && appContext.loginData.userType === 'Admin' ? (  
 <div>  
 <AdminNavigationEntries/>  
 </div>  
 ) : (  
 <div>  
 <Nav *vertical*>  
 </Nav>  
 </div>  
 )  
 }  
 </div>  
 )  
}  
  
*export default NavigationBar*

**login.jsx**

*import* React *from* 'react'  
*import* Header *from* '../../../components/header/header'  
*import Footer from* '../../../components/footer/footer'  
*import LoginForm from* './login-form/login-form'  
*import* './login.css'  
  
*const Login* = props => {  
 *return* (  
 <div>  
 <div>  
 <Header/>  
 </div>  
 <div *className*='container login-form'>  
 <LoginForm *history*={props.history}/>  
 </div>  
 <div>  
 <Footer/>  
 </div>  
 </div>  
 )  
}  
  
*export default Login*

**login-form.jsx**

*import* React, {*useContext*, *useState*} *from* 'react'  
*import* {*Link*} *from* 'react-router-dom'  
*import* {Card, CardBody} *from* 'reactstrap'  
*import* axios *from* 'axios'  
*import* {authStoreKey} *from* '../../../../config/main.config'  
*import* {authApi} *from* '../../../../config/api.config'  
*import* {*setLocalStorageItem*} *from* '../../../../helpers/local-storage.helpers'  
*import* {*isEmpty*} *from* '../../../../helpers/common.helpers'  
*import* {AppContext} *from* '../../../../global/app-context'  
*import Loader from* '../../../../components/loader/loader'  
*import TextField from* '../../../../components/text-field/text-field'  
*import ButtonComponent from* '../../../../components/button/button'  
*import* './login-form.css'  
  
*const LoginForm* = props => {  
 *const* appContext = *useContext*(AppContext)  
  
 *const* helperEmail = 'Please enter your email address.'  
 *const* helperPassword = 'Please enter your password.'  
  
 *const* [loader, setLoader] = *useState*(*false*)  
  
 *const* [email, setEmail] = *useState*('')  
 *const* [password, setPassword] = *useState*('')  
  
 *const* [errorEmail, setErrorEmail] = *useState*('')  
 *const* [errorPassword, setErrorPassword] = *useState*('')  
  
 *const* [emailValid, setEmailValid] = *useState*(*false*)  
 *const* [passwordValid, setPasswordValid] = *useState*(*false*)  
  
 *const* [error, setError] = *useState*('')  
  
 *const* onChangeEmail = *async* event => {  
 setEmail(event.value)  
 setEmailValid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorEmail('')  
 setError('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorEmail('Please enter a valid email address.')  
 }  
 }  
  
 *const* onChangePassword = *async* event => {  
 setPassword(event.value)  
 setPasswordValid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorPassword('')  
 setError('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorPassword('Please enter a valid password.')  
 }  
 }  
  
 *function* isDisabled() {  
 *return* !emailValid || !passwordValid  
 }  
  
 *const* onSubmit = *async* () => {  
 setError('')  
 *const* data = {  
 'email': email.trim(),  
 'password': password  
 }  
 setLoader(*true*)  
 axios.post(`${authApi}login`, data).then(res => {  
 *if* (res.data.status === 200) {  
 *setLocalStorageItem*(authStoreKey, res.data.user)  
 appContext.login(res.data.user)  
 props.history.push('/home')  
 } *else if* (res.data.status === 401) {  
 setError(res.data.message)  
 }  
 setLoader(*false*)  
 }).catch(error => {  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
 })  
 }  
  
 *return* (  
 <div *className*='login-wrapper'>  
 {  
 loader ? (  
 <Loader/>  
 ) : *null* }  
 <Card *className*='overflow-hidden'>  
 <div *className*='login-header'>  
 <div *className*='text-primary text-center p-4'>  
 <h1 *className*='text-white font-size-20 text-uppercase'>  
 <i *className*='login-icon fas fa-users mt-2'/>  
 <span *className*='ms-3'>  
 Login  
 </span>  
 </h1>  
 </div>  
 </div>  
 <CardBody *className*='p-4'>  
 <div>  
 <small>  
 {  
 error ? (  
 <span *className*='p-3 error'>  
 {error}  
 </span>  
 ) : *null* }  
 </small>  
 </div>  
 <div *className*='p-3'>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Email'}  
 *type*={'email'}  
 *name*={'email'}  
 *value*={email}  
 *errorText*={errorEmail}  
 *helperText*={helperEmail}  
 *minLength*={6}  
 *maxLength*={100}  
 *onChangeFn*={event => onChangeEmail(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Password'}  
 *type*={'password'}  
 *name*={'password'}  
 *value*={password}  
 *errorText*={errorPassword}  
 *helperText*={helperPassword}  
 *minLength*={4}  
 *maxLength*={50}  
 *onChangeFn*={event => onChangePassword(event)}/>  
 </div>  
 <div *className*='text-center mt-4 mb-3'>  
 <ButtonComponent *btnText*={'Login'}  
 *isFullWidth*={*false*}  
 *elementStyle*={'login-btn'}  
 *disabled*={isDisabled()}  
 *onClickFn*={onSubmit}/>  
 </div>  
 </div>  
 <div *className*='ms-3'>  
 <label>Don't have an account?&nbsp;</label>  
 <Link *to*={'/register'}>  
 <label *className*='register-link'>  
 Register  
 </label>  
 </Link>  
 </div>  
 </CardBody>  
 </Card>  
 </div>  
 )  
}  
  
*export default LoginForm*

**register.jsx**

*import* React *from* 'react'  
*import* Header *from* '../../../components/header/header'  
*import Footer from* '../../../components/footer/footer'  
*import RegisterForm from* './register-form/register-form'  
*import* './register.css'  
  
*const Register* = props => {  
 *return* (  
 <div>  
 <div>  
 <Header/>  
 </div>  
 <div *className*='container register-form'>  
 <RegisterForm *history*={props.history}/>  
 </div>  
 <div>  
 <Footer/>  
 </div>  
 </div>  
 )  
}  
  
*export default Register*

**register-form.jsx**

*import* React, {*useState*} *from* 'react'  
*import* {*Link*} *from* 'react-router-dom'  
*import* {Card, CardBody, Modal, ModalBody, ModalFooter, ModalHeader} *from* 'reactstrap'  
*import* axios *from* 'axios'  
*import* {*isEmpty*} *from* '../../../../helpers/common.helpers'  
*import* {usersApi} *from* '../../../../config/api.config'  
*import Loader from* '../../../../components/loader/loader'  
*import TextField from* '../../../../components/text-field/text-field'  
*import ButtonComponent from* '../../../../components/button/button'  
*import* './register-form.css'  
  
*const RegisterForm* = props => {  
 *const* [successModal, setSuccessModal] = *useState*(*false*)  
 *const* [message, setMessage] = *useState*('')  
  
 *const* helperFirstName = 'Please enter your first name.'  
 *const* helperLastName = 'Please enter your last name.'  
 *const* helperEmail = 'Please enter your email address.'  
 *const* helperPassword = 'Please enter a password.'  
 *const* helperConfirmPassword = 'Please enter the password again.'  
  
 *const* [loader, setLoader] = *useState*(*false*)  
  
 *const* [firstName, setFirstName] = *useState*('')  
 *const* [lastName, setLastName] = *useState*('')  
 *const* [email, setEmail] = *useState*('')  
 *const* [password, setPassword] = *useState*('')  
 *const* [confirmPassword, setConfirmPassword] = *useState*('')  
  
 *const* [errorFirstName, setErrorFirstName] = *useState*('')  
 *const* [errorLastName, setErrorLastName] = *useState*('')  
 *const* [errorEmail, setErrorEmail] = *useState*('')  
 *const* [errorPassword, setErrorPassword] = *useState*('')  
 *const* [errorConfirmPassword, setErrorConfirmPassword] = *useState*('')  
  
 *const* [firstNameValid, setFirstNameValid] = *useState*(*false*)  
 *const* [lastNameValid, setLastNameValid] = *useState*(*false*)  
 *const* [emailValid, setEmailValid] = *useState*(*false*)  
 *const* [passwordValid, setPasswordValid] = *useState*(*false*)  
 *const* [confirmPasswordValid, setConfirmPasswordValid] = *useState*(*false*)  
  
 *const* [error, setError] = *useState*('')  
  
 *const* onChangeFirstName = *async* event => {  
 setFirstName(event.value)  
 setFirstNameValid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorFirstName('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorFirstName('Please enter a valid first name.')  
 }  
 }  
  
 *const* onChangeLastName = *async* event => {  
 setLastName(event.value)  
 setLastNameValid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorLastName('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorLastName('Please enter a valid last name.')  
 }  
 }  
  
 *const* onChangeEmail = *async* event => {  
 setEmail(event.value)  
 setEmailValid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorEmail('')  
 setError('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorEmail('Please enter a valid email address.')  
 }  
 }  
  
 *const* onChangePassword = *async* event => {  
 setPassword(event.value)  
 setPasswordValid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorPassword('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorPassword('Please enter a strong password with at least 4 characters.')  
 }  
 }  
  
 *const* onChangeConfirmPassword = *async* event => {  
 setConfirmPassword(event.value)  
 setConfirmPasswordValid(event.value === password)  
 setErrorConfirmPassword('')  
 *if* (event.value !== password) {  
 setErrorConfirmPassword('Please make sure your passwords match.')  
 }  
 }  
  
 *function* isDisabled() {  
 *return* !firstNameValid || !lastNameValid || !emailValid || !passwordValid || !confirmPasswordValid  
 }  
  
 *const* toggleSuccessModal = *async* () => {  
 setSuccessModal(!successModal)  
 }  
  
 *const* onClick = *async* () => {  
 props.history.push('/login')  
 }  
  
 *const* onSubmit = *async* () => {  
 setError('')  
 *const* data = {  
 'firstName': firstName.trim(),  
 'lastName': lastName.trim(),  
 'email': email.trim(),  
 'password': password  
 }  
 setLoader(*true*)  
 axios.post(`${usersApi}users`, data).then(res => {  
 *if* (res.data.status === 201) {  
 setLoader(*false*)  
 setMessage(res.data.message)  
 toggleSuccessModal()  
 } *else if* (res.data.status === 409) {  
 setError(res.data.message)  
 }  
 setLoader(*false*)  
 }).catch(error => {  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
 })  
 }  
  
 *return* (  
 <div *className*='register-wrapper'>  
 {  
 loader ? (  
 <Loader/>  
 ) : *null* }  
 <div>  
 <Modal *isOpen*={successModal}  
 *toggle*={toggleSuccessModal}  
 *className*='modal-close'>  
 <ModalHeader *toggle*={toggleSuccessModal}  
 *className*='text-uppercase title'>  
 Success!  
 </ModalHeader>  
 <ModalBody>  
 {message}  
 </ModalBody>  
 <ModalFooter>  
 <ButtonComponent *btnText*={'Ok'}  
 *isFullWidth*={*false*}  
 *elementStyle*={'ok-button'}  
 *disabled*={*false*}  
 *onClickFn*={onClick}/>  
 </ModalFooter>  
 </Modal>  
 </div>  
 <div>  
 <Card *className*='overflow-hidden'>  
 <div *className*='register-header'>  
 <div *className*='text-primary text-center p-4'>  
 <h1 *className*='text-white font-size-20 text-uppercase'>  
 <i *className*='register-icon fas fa-user-plus mt-2'/>  
 <span *className*='ms-3'>  
 Register  
 </span>  
 </h1>  
 </div>  
 </div>  
 <CardBody *className*='p-4'>  
 <div>  
 <small>  
 {  
 error ? (  
 <span *className*='p-3 error'>  
 {error}  
 </span>  
 ) : *null* }  
 </small>  
 </div>  
 <div *className*='p-3'>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'First Name'}  
 *name*={'firstName'}  
 *value*={firstName}  
 *errorText*={errorFirstName}  
 *helperText*={helperFirstName}  
 *maxLength*={50}  
 *onChangeFn*={event => onChangeFirstName(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Last Name'}  
 *name*={'lastName'}  
 *value*={lastName}  
 *errorText*={errorLastName}  
 *helperText*={helperLastName}  
 *maxLength*={50}  
 *onChangeFn*={event => onChangeLastName(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Email'}  
 *type*={'email'}  
 *name*={'email'}  
 *value*={email}  
 *errorText*={errorEmail}  
 *helperText*={helperEmail}  
 *minLength*={6}  
 *maxLength*={100}  
 *onChangeFn*={event => onChangeEmail(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Password'}  
 *type*={'password'}  
 *name*={'password'}  
 *value*={password}  
 *errorText*={errorPassword}  
 *helperText*={helperPassword}  
 *minLength*={4}  
 *maxLength*={50}  
 *onChangeFn*={event => onChangePassword(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Confirm Password'}  
 *type*={'password'}  
 *name*={'confirmPassword'}  
 *value*={confirmPassword}  
 *errorText*={errorConfirmPassword}  
 *helperText*={helperConfirmPassword}  
 *disabled*={!passwordValid}  
 *minLength*={4}  
 *maxLength*={50}  
 *onChangeFn*={event => onChangeConfirmPassword(event)}/>  
 </div>  
 <div *className*='text-center mt-4 mb-3'>  
 <ButtonComponent *btnText*={'Register'}  
 *isFullWidth*={*false*}  
 *elementStyle*={'register-btn'}  
 *disabled*={isDisabled()}  
 *onClickFn*={onSubmit}/>  
 </div>  
 </div>  
 <div *className*='ms-3'>  
 <label>Already have an account?&nbsp;</label>  
 <Link *to*={'/login'}>  
 <label *className*='login-link'>  
 Login  
 </label>  
 </Link>  
 </div>  
 </CardBody>  
 </Card>  
 </div>  
 </div>  
 )  
}  
  
*export default RegisterForm*

**2.1 User Management – Backend**

**user-controller.js**

*const* updateUser = *async* (req, res) => {  
 *let* user  
 *let* existingUser  
  
 *const* {  
 id  
 } = req.params  
  
 *const* {  
 firstName,  
 lastName,  
 email,  
 password,  
 userType  
 } = req.body  
  
 *try* {  
 user = *await* UserModel.findById(id)  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 *try* {  
 existingUser = *await* UserModel.findOne({  
 email: email  
 })  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 *if* (existingUser && email !== user.email) {  
 res.send({  
 status: 409,  
 message: 'A user with the same email already exists.'  
 })  
 }  
  
 user.firstName = firstName  
 user.lastName = lastName  
 user.email = email  
 user.password = bcrypt.hashSync(password, 10)  
 user.userType = userType  
  
 *try* {  
 *await* user.save()  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 res.send({  
 status: 200,  
 message: 'User updated successfully!'  
 })  
}  
  
*const* promoteUser = *async* (req, res) => {  
 *let* user  
  
 *const* {  
 id  
 } = req.params  
  
 *try* {  
 user = *await* UserModel.findById(id)  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 user.userType = 'Admin'  
  
 *try* {  
 *await* user.save()  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 res.send({  
 status: 200,  
 message: 'The user has been successfully promoted as a teacher!'  
 })  
}  
  
*const* deleteUser = *async* (req, res) => {  
 *let* user  
  
 *const* {  
 id  
 } = req.params  
  
 *try* {  
 user = *await* UserModel.findById(id)  
 *await* user.remove()  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 res.send({  
 status: 200,  
 message: 'User deleted successfully!'  
 })  
}  
  
*const* getUser = *async* (req, res) => {  
 *let* user  
  
 *const* {  
 id  
 } = req.params  
  
 *try* {  
 user = *await* UserModel.findById(id)  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 res.send({  
 status: 200,  
 user: user  
 })  
}  
  
*const* getUserList = *async* (req, res) => {  
 *let* userList  
  
 *try* {  
 userList = *await* UserModel.find()  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 res.send({  
 status: 200,  
 userList: userList  
 })  
}  
  
exports.addUser = addUser  
exports.addAdmin = addAdmin  
exports.updateUser = updateUser  
exports.promoteUser = promoteUser  
exports.deleteUser = deleteUser  
exports.getUser = getUser  
exports.getUserList = getUserList

**user.routes.js**

*const* express = require('express')  
*const* UsersController = require('../controllers/users-controller')  
  
*const* router = express.*Router*()  
  
router.post('/users', UsersController.addUser)  
router.post('/admin', UsersController.addAdmin)  
router.put('/users/:id', UsersController.updateUser)  
router.put('/users/promote/:id', UsersController.promoteUser)  
router.delete('/users/:id', UsersController.deleteUser)  
router.get('/users/:id', UsersController.getUser)  
router.get('/users', UsersController.getUserList)  
  
module.exports = router

**2.2 User Management – Frontend**

**user-management.jsx**

*import* React *from* 'react'  
*import* Header *from* '../../../components/header/header'  
*import Footer from* '../../../components/footer/footer'  
*import UserManagementComponent from* './user-management-component/user-management-component'  
*import* './user-management.css'  
  
*const UserManagement* = () => {  
 *return* (  
 <div>  
 <div>  
 <Header/>  
 </div>  
 <h1 *className*='text-center text-uppercase mt-5 page-title'>  
 User Management  
 </h1>  
 <div *className*='container user-management-page'>  
 <UserManagementComponent/>  
 </div>  
 <div>  
 <Footer/>  
 </div>  
 </div>  
 )  
}  
  
*export default UserManagement*

**user-management-component.jsx**

*import* React, {*useEffect*, *useState*} *from* 'react'  
*import* {Modal, ModalBody, ModalFooter, ModalHeader, Table} *from* 'reactstrap'  
*import* axios *from* 'axios'  
*import* {usersApi} *from* '../../../../config/api.config'  
*import Loader from* '../../../../components/loader/loader'  
*import ButtonComponent from* '../../../../components/button/button'  
*import* './user-management-component.css'  
  
*const UserManagementComponent* = () => {  
 *const* [loader, setLoader] = *useState*(*false*)  
 *const* [successModal, setSuccessModal] = *useState*(*false*)  
 *const* [successModalEdit, setSuccessModalEdit] = *useState*(*false*)  
 *const* [modal, setModal] = *useState*(*false*)  
 *const* [modalEdit, setModalEdit] = *useState*(*false*)  
 *const* [message, setMessage] = *useState*('')  
 *const* [deleteId, setDeleteId] = *useState*('')  
 *const* [editId, setEditId] = *useState*('')  
 *const* [data, setData] = *useState*(*null*)  
 *const* [error, setError] = *useState*('')  
  
 *useEffect*(() => {  
 loadData().then(() => {  
 })  
 }, [])  
  
 *const* loadData = *async* () => {  
 setLoader(*true*)  
 axios.get(`${usersApi}users`).then(res => {  
 setData(res.data.userList)  
 setLoader(*false*)  
 }).catch(error => {  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
 })  
 }  
  
 *const* onEdit = *async* id => {  
 setEditId(id)  
 *await* toggleEdit()  
 }  
  
 *const* toggleEdit = *async* () => {  
 setModalEdit(!modalEdit)  
 }  
  
 *const* toggleSuccessModalEdit = *async* () => {  
 setSuccessModalEdit(!successModalEdit)  
 }  
  
 *const* confirmEdit = *async* () => {  
 setError('')  
 setLoader(*true*)  
 axios.put(`${usersApi}users/promote/${editId}`).then(res => {  
 *if* (res.data.status === 200) {  
 setMessage(res.data.message)  
 toggleEdit()  
 toggleSuccessModalEdit()  
 loadData()  
 } *else* {  
 toggleEdit()  
 setError('An unexpected error occurred. Please try again later.')  
 *console*.error(error)  
 }  
 setLoader(*false*)  
 }).catch(error => {  
 toggleEdit()  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
 })  
 }  
  
 *const* onDelete = *async* id => {  
 setDeleteId(id)  
 *await* toggle()  
 }  
  
 *const* toggle = *async* () => {  
 setModal(!modal)  
 }  
  
 *const* toggleSuccessModal = *async* () => {  
 setSuccessModal(!successModal)  
 }  
  
 *const* confirmDelete = *async* () => {  
 setError('')  
 setLoader(*true*)  
 axios.delete(`${usersApi}users/${deleteId}`).then(res => {  
 *if* (res.data.status === 200) {  
 setData(data.filter(item => item.\_id !== deleteId))  
 setMessage(res.data.message)  
 toggle()  
 toggleSuccessModal()  
 } *else* {  
 toggle()  
 setError('An unexpected error occurred. Please try again later.')  
 *console*.error(error)  
 }  
 setLoader(*false*)  
 }).catch(error => {  
 toggle()  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
 })  
 }  
  
 *return* (  
 <div>  
 {  
 loader ? (  
 <Loader/>  
 ) : *null* }  
 <div>  
 <Modal *isOpen*={successModalEdit}  
 *toggle*={toggleSuccessModalEdit}  
 *className*='modal-close'>  
 <ModalHeader *toggle*={toggleSuccessModalEdit}  
 *className*='text-uppercase title'>  
 Success!  
 </ModalHeader>  
 <ModalBody>  
 {message}  
 </ModalBody>  
 <ModalFooter>  
 <ButtonComponent *btnText*={'Ok'}  
 *isFullWidth*={*false*}  
 *elementStyle*={'ok-button'}  
 *disabled*={*false*}  
 *onClickFn*={toggleSuccessModalEdit}/>  
 </ModalFooter>  
 </Modal>  
 </div>  
 <div>  
 <Modal *isOpen*={modalEdit}  
 *toggle*={toggleEdit}  
 *className*='modal-close'>  
 <ModalHeader *toggle*={toggleEdit}  
 *className*='text-uppercase'>  
 Promote to Teacher  
 </ModalHeader>  
 <ModalBody>  
 Are you sure you want to promote this user as a teacher?  
 </ModalBody>  
 <ModalFooter>  
 <ButtonComponent *btnText*={'Yes'}  
 *isFullWidth*={*false*}  
 *elementStyle*={'yes-button-edit'}  
 *disabled*={*false*}  
 *onClickFn*={confirmEdit}/>  
 <ButtonComponent *btnText*={'No'}  
 *isFullWidth*={*false*}  
 *elementStyle*={'no-button'}  
 *disabled*={*false*}  
 *onClickFn*={toggleEdit}/>  
 </ModalFooter>  
 </Modal>  
 </div>  
 <div>  
 <Modal *isOpen*={successModal}  
 *toggle*={toggleSuccessModal}  
 *className*='modal-close'>  
 <ModalHeader *toggle*={toggleSuccessModal}  
 *className*='text-uppercase title'>  
 Success!  
 </ModalHeader>  
 <ModalBody>  
 {message}  
 </ModalBody>  
 <ModalFooter>  
 <ButtonComponent *btnText*={'Ok'}  
 *isFullWidth*={*false*}  
 *elementStyle*={'ok-button'}  
 *disabled*={*false*}  
 *onClickFn*={toggleSuccessModal}/>  
 </ModalFooter>  
 </Modal>  
 </div>  
 <div>  
 <Modal *isOpen*={modal}  
 *toggle*={toggle}  
 *className*='modal-close'>  
 <ModalHeader *toggle*={toggle}  
 *className*='text-uppercase'>  
 Delete User  
 </ModalHeader>  
 <ModalBody>  
 Are you sure you want to delete this user?  
 </ModalBody>  
 <ModalFooter>  
 <ButtonComponent *btnText*={'Yes'}  
 *isFullWidth*={*false*}  
 *elementStyle*={'yes-button'}  
 *disabled*={*false*}  
 *onClickFn*={confirmDelete}/>  
 <ButtonComponent *btnText*={'No'}  
 *isFullWidth*={*false*}  
 *elementStyle*={'no-button'}  
 *disabled*={*false*}  
 *onClickFn*={toggle}/>  
 </ModalFooter>  
 </Modal>  
 </div>  
 <div>  
 <small>  
 {  
 error ? (  
 <span *className*='error'>  
 {error}  
 </span>  
 ) : *null* }  
 </small>  
 </div>  
 <div>  
 <Table *bordered*>  
 <thead>  
 <tr *className*='text-center'>  
 <th>User ID</th>  
 <th>First Name</th>  
 <th>Last Name</th>  
 <th>Email</th>  
 <th>User Type</th>  
 <th *colSpan*={2}>  
 Actions  
 </th>  
 </tr>  
 </thead>  
 <tbody>  
 {  
 data && data.map((item) => {  
 *return* (  
 <tr *key*={item.\_id}>  
 <td>  
 {item.userId}  
 </td>  
 <td>  
 {item.firstName}  
 </td>  
 <td>  
 {item.lastName}  
 </td>  
 <td>  
 {item.email}  
 </td>  
 <td>  
 {  
 item.userType === 'Admin' ? (  
 <span *className*='text-danger'>  
 Teacher  
 </span>  
 ) : (  
 <span *className*='text-primary'>  
 Student  
 </span>  
 )  
 }  
 </td>  
 <td *className*='text-center'>  
 {  
 item.userType === 'User' ? (  
 <i *className*='fas fa-arrow-up edit'  
 *title*='Promote to Teacher'  
 *onClick*={() => onEdit(item.\_id)}/>  
 ) : *null* }  
 </td>  
 <td *className*='text-center'>  
 {  
 item.userType === 'User' ? (  
 <i *className*='fas fa-trash-alt delete'  
 *title*='Delete Student'  
 *onClick*={() => onDelete(item.\_id)}/>  
 ) : *null* }  
 </td>  
 </tr>  
 )  
 })  
 }  
 </tbody>  
 </Table>  
 </div>  
 </div>  
 )  
}  
  
*export default UserManagementComponent*

**3.1 Quiz Management - Backend**

**quizzes.model.js**

*const* mongoose = require('mongoose')  
*const* autoIncrement = require('mongoose-auto-increment')  
*const uniqueValidator* = require('mongoose-unique-validator')  
  
*const* Schema = mongoose.Schema  
  
*const* QuizzesSchema = *new* Schema({  
 quizId: {  
 type: Number,  
 required: *false*,  
 unique: *true*,  
 trim: *true* },  
 quizTitle: {  
 type: String,  
 required: *true*,  
 unique: *true*,  
 trim: *true* },  
 quizDescription: {  
 type: String,  
 required: *true*,  
 unique: *false*,  
 trim: *true* },  
 questions: [{  
 question: {  
 type: String,  
 required: *true*,  
 unique: *false*,  
 trim: *true* },  
 answer: {  
 type: String,  
 required: *true*,  
 unique: *false*,  
 trim: *true* }  
 }]  
}, {  
 timestamps: *true*,  
 collection: 'Quizzes'  
})  
  
QuizzesSchema.plugin(*uniqueValidator*)  
  
autoIncrement.initialize(mongoose.connection)  
  
QuizzesSchema.plugin(autoIncrement.plugin, {  
 model: 'Quizzes',  
 field: 'quizId',  
 startAt: 1000,  
 incrementBy: 1  
})  
  
module.exports = mongoose.*model*('Quizzes', QuizzesSchema)

**quizzes.routes.js**

*const* express = require('express')  
*const* QuizController = require('../controllers/quizzes-controller')  
  
*const* router = express.*Router*()  
  
router.post('/quizzes', QuizController.addQuiz)  
router.put('/quizzes/:id', QuizController.updateQuiz)  
router.delete('/quizzes/:id', QuizController.deleteQuiz)  
router.get('/quizzes/:id', QuizController.getQuiz)  
router.get('/quizzes', QuizController.getQuizList)  
  
module.exports = router

* 1. **Quiz Management – Frontend**

**quiz-list.jsx**

*import* React *from* 'react'  
*import* Header *from* '../../../../components/header/header'  
*import Footer from* '../../../../components/footer/footer'  
*import QuizListComponent from* './quiz-list-component/quiz-list-component'  
*import* './quiz-list.css'  
  
*const QuizList* = props => {  
 *return* (  
 <div>  
 <div>  
 <Header/>  
 </div>  
 <h1 *className*='text-center text-uppercase mt-5 page-title'>  
 Content Management  
 </h1>  
 <div *className*='container quiz-list-page'>  
 <QuizListComponent *history*={props.history}/>  
 </div>  
 <div>  
 <Footer/>  
 </div>  
 </div>  
 )  
}  
  
*export default QuizList*

**quiz-list-component.jsx**

*import* React, {*useEffect*, *useState*} *from* 'react'  
*import* {  
 Card,  
 CardBody,  
 CardDeck,  
 CardFooter,  
 CardText,  
 CardTitle,  
 Modal,  
 ModalBody,  
 ModalFooter,  
 ModalHeader  
} *from* 'reactstrap'  
*import* axios *from* 'axios'  
*import* {quizzesApi} *from* '../../../../../config/api.config'  
*import Loader from* '../../../../../components/loader/loader'  
*import ButtonComponent from* '../../../../../components/button/button'  
*import* './quiz-list-component.css'  
  
*const QuizListComponent* = props => {  
 *const* [loader, setLoader] = *useState*(*false*)  
 *const* [successModal, setSuccessModal] = *useState*(*false*)  
 *const* [modal, setModal] = *useState*(*false*)  
 *const* [message, setMessage] = *useState*('')  
 *const* [data, setData] = *useState*(*null*)  
 *const* [deleteId, setDeleteId] = *useState*('')  
 *const* [error, setError] = *useState*('')  
  
 *useEffect*(() => {  
 loadData().then(() => {  
 })  
 }, [])  
  
 *const* loadData = *async* () => {  
 setLoader(*true*)  
 axios.get(`${quizzesApi}quizzes`).then(res => {  
 setData(res.data.quizList)  
 setLoader(*false*)  
 }).catch(error => {  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
 })  
 }  
  
 *const* onDelete = *async* id => {  
 setDeleteId(id)  
 *await* toggle()  
 }  
  
 *const* onView = *async* id => {  
 props.history.push('/single-quiz/' + id)  
 }  
  
 *const* onNewQuiz = *async* () => {  
 props.history.push('/add-quiz')  
 }  
  
 *const* toggle = *async* () => {  
 setModal(!modal)  
 }  
  
 *const* toggleSuccessModal = *async* () => {  
 setSuccessModal(!successModal)  
 }  
  
 *const* confirmDelete = *async* () => {  
 setError('')  
 setLoader(*true*)  
 axios.delete(`${quizzesApi}quizzes/${deleteId}`).then(res => {  
 *if* (res.data.status === 200) {  
 setData(data.filter(item => item.\_id !== deleteId))  
 setMessage(res.data.message)  
 toggle()  
 toggleSuccessModal()  
 } *else* {  
 toggle()  
 setError('An unexpected error occurred. Please try again later.')  
 *console*.error(error)  
 }  
 setLoader(*false*)  
 }).catch(error => {  
 toggle()  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
 })  
 }  
  
 *return* (  
 <div>  
 {  
 loader ? (  
 <Loader/>  
 ) : *null* }  
 <div>  
 <Modal *isOpen*={successModal}  
 *toggle*={toggleSuccessModal}  
 *className*='modal-close'>  
 <ModalHeader *toggle*={toggleSuccessModal}  
 *className*='text-uppercase title'>  
 Success!  
 </ModalHeader>  
 <ModalBody>  
 {message}  
 </ModalBody>  
 <ModalFooter>  
 <ButtonComponent *btnText*={'Ok'}  
 *isFullWidth*={*false*}  
 *elementStyle*={'ok-button'}  
 *disabled*={*false*}  
 *onClickFn*={toggleSuccessModal}/>  
 </ModalFooter>  
 </Modal>  
 </div>  
 <div>  
 <Modal *isOpen*={modal}  
 *toggle*={toggle}  
 *className*='modal-close'>  
 <ModalHeader *toggle*={toggle}  
 *className*='text-uppercase'>  
 Delete Quiz  
 </ModalHeader>  
 <ModalBody>  
 Are you sure you want to delete this quiz?  
 </ModalBody>  
 <ModalFooter>  
 <ButtonComponent *btnText*={'Yes'}  
 *isFullWidth*={*false*}  
 *elementStyle*={'yes-button'}  
 *disabled*={*false*}  
 *onClickFn*={confirmDelete}/>  
 <ButtonComponent *btnText*={'No'}  
 *isFullWidth*={*false*}  
 *elementStyle*={'no-button'}  
 *disabled*={*false*}  
 *onClickFn*={toggle}/>  
 </ModalFooter>  
 </Modal>  
 </div>  
 <div>  
 <small>  
 {  
 error ? (  
 <span *className*='error'>  
 {error}  
 </span>  
 ) : *null* }  
 </small>  
 </div>  
 <div>  
 <div *className*='m-3'>  
 <ButtonComponent *btnText*={'New Quiz'}  
 *isFullWidth*={*false*}  
 *disabled*={*false*}  
 *onClickFn*={onNewQuiz}/>  
 </div>  
 <CardDeck *className*='card-deck'>  
 <div>  
 {  
 data && data.map(item => {  
 *return* (  
 <div>  
 <Card *key*={item.\_id}  
 *title*='View Quiz'  
 *className*='m-3 card-item justify-content-center'>  
 <CardBody *onClick*={() => onView(item.\_id)}>  
 <CardTitle *className*='text-uppercase text-center m-4'  
 *tag*='h2'>  
 <label>  
 {item.quizTitle}  
 </label>  
 </CardTitle>  
 <CardText *className*='m-4 text-center'>  
 <label>  
 {item.quizDescription}  
 </label>  
 </CardText>  
 </CardBody>  
 <CardFooter>  
 <div *className*='text-center m-2'>  
 <i *className*='fas fa-trash-alt delete'  
 *title*='Delete Quiz'  
 *onClick*={() => onDelete(item.\_id)}/>  
 </div>  
 </CardFooter>  
 </Card>  
 </div>  
 )  
 })  
 }  
 </div>  
 </CardDeck>  
 </div>  
 </div>  
 )  
}  
  
*export default QuizListComponent*

**add-quiz.jsx**

*import* React *from* 'react'  
*import* Header *from* '../../../../components/header/header'  
*import Footer from* '../../../../components/footer/footer'  
*import AddQuizComponent from* './add-quiz-component/add-quiz-component'  
*import* './add-quiz.css'  
  
*const AddQuiz* = props => {  
 *return* (  
 <div>  
 <div>  
 <Header/>  
 </div>  
 <div *className*='container add-quiz-page'>  
 <AddQuizComponent *history*={props.history}/>  
 </div>  
 <div>  
 <Footer/>  
 </div>  
 </div>  
 )  
}  
  
*export default AddQuiz*

**add-quiz-component.jsx**

*import* React, {*useState*} *from* 'react'  
*import* axios *from* 'axios'  
*import* {Card, CardBody, Modal, ModalBody, ModalFooter, ModalHeader} *from* 'reactstrap'  
*import* {*isEmpty*} *from* '../../../../../helpers/common.helpers'  
*import* {quizzesApi} *from* '../../../../../config/api.config'  
*import Loader from* '../../../../../components/loader/loader'  
*import ButtonComponent from* '../../../../../components/button/button'  
*import TextField from* '../../../../../components/text-field/text-field'  
*import* './add-quiz-component.css'  
  
*const AddQuizComponent* = props => {  
 *const* [successModal, setSuccessModal] = *useState*(*false*)  
 *const* [message, setMessage] = *useState*('')  
  
 *const* helperQuizTitle = 'Please enter a title for the quiz.'  
 *const* helperQuizDescription = 'Please enter a description about the quiz.'  
 *const* helperAnswer1 = 'Please enter the answer for question 1.'  
 *const* helperAnswer2 = 'Please enter the answer for question 2.'  
 *const* helperAnswer3 = 'Please enter the answer for question 3.'  
 *const* helperQuestion1 = 'Please enter question 1.'  
 *const* helperQuestion2 = 'Please enter question 2.'  
 *const* helperQuestion3 = 'Please enter question 3.'  
  
 *const* [loader, setLoader] = *useState*(*false*)  
  
 *const* [quizTitle, setQuizTitle] = *useState*('')  
 *const* [quizDescription, setQuizDescription] = *useState*('')  
 *const* [question1, setQuestion1] = *useState*('')  
 *const* [answer1, setAnswer1] = *useState*('')  
 *const* [question2, setQuestion2] = *useState*('')  
 *const* [answer2, setAnswer2] = *useState*('')  
 *const* [question3, setQuestion3] = *useState*('')  
 *const* [answer3, setAnswer3] = *useState*('')  
  
 *const* [errorQuizTitle, setErrorQuizTitle] = *useState*('')  
 *const* [errorQuizDescription, setErrorQuizDescription] = *useState*('')  
 *const* [errorQuestion1, setErrorQuestion1] = *useState*('')  
 *const* [errorAnswer1, setErrorAnswer1] = *useState*('')  
 *const* [errorQuestion2, setErrorQuestion2] = *useState*('')  
 *const* [errorAnswer2, setErrorAnswer2] = *useState*('')  
 *const* [errorQuestion3, setErrorQuestion3] = *useState*('')  
 *const* [errorAnswer3, setErrorAnswer3] = *useState*('')  
  
 *const* [quizTitleValid, setQuizTitleValid] = *useState*(*false*)  
 *const* [quizDescriptionValid, setQuizDescriptionValid] = *useState*(*false*)  
 *const* [question1Valid, setQuestion1Valid] = *useState*(*false*)  
 *const* [answer1Valid, setAnswer1Valid] = *useState*(*false*)  
 *const* [question2Valid, setQuestion2Valid] = *useState*(*false*)  
 *const* [answer2Valid, setAnswer2Valid] = *useState*(*false*)  
 *const* [question3Valid, setQuestion3Valid] = *useState*(*false*)  
 *const* [answer3Valid, setAnswer3Valid] = *useState*(*false*)  
  
 *const* [error, setError] = *useState*('')  
  
 *const* onChangeQuizTitle = *async* event => {  
 setQuizTitle(event.value)  
 setQuizTitleValid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorQuizTitle('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorQuizTitle('Please enter a valid quiz title.')  
 }  
 }  
  
 *const* onChangeQuizDescription = *async* event => {  
 setQuizDescription(event.value)  
 setQuizDescriptionValid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorQuizDescription('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorQuizDescription('Please enter a valid quiz description.')  
 }  
 }  
  
 *const* onChangeQuestion1 = *async* event => {  
 setQuestion1(event.value)  
 setQuestion1Valid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorQuestion1('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorQuestion1('Please enter a valid question.')  
 }  
 }  
  
 *const* onChangeQuestion2 = *async* event => {  
 setQuestion2(event.value)  
 setQuestion2Valid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorQuestion2('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorQuestion2('Please enter a valid question.')  
 }  
 }  
  
 *const* onChangeQuestion3 = *async* event => {  
 setQuestion3(event.value)  
 setQuestion3Valid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorQuestion3('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorQuestion3('Please enter a valid question.')  
 }  
 }  
  
 *const* onChangeAnswer1 = *async* event => {  
 setAnswer1(event.value)  
 setAnswer1Valid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorAnswer1('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorAnswer1('Please enter a valid answer.')  
 }  
 }  
  
 *const* onChangeAnswer2 = *async* event => {  
 setAnswer2(event.value)  
 setAnswer2Valid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorAnswer2('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorAnswer2('Please enter a valid answer.')  
 }  
 }  
  
 *const* onChangeAnswer3 = *async* event => {  
 setAnswer3(event.value)  
 setAnswer3Valid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorAnswer3('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorAnswer3('Please enter a valid answer.')  
 }  
 }  
  
 *function* isDisabled() {  
 *return* !quizTitleValid || !quizDescriptionValid || !question1Valid || !question2Valid || !question3Valid ||  
 !answer1Valid || !answer2Valid || !answer3Valid  
 }  
  
 *const* toggleSuccessModal = *async* () => {  
 setSuccessModal(!successModal)  
 }  
  
 *const* onClick = *async* () => {  
 props.history.push('/content-management')  
 }  
  
 *const* onSubmit = *async* () => {  
 setError('')  
 *const* data = {  
 'quizTitle': quizTitle.trim(),  
 'quizDescription': quizDescription.trim(),  
 'questions': [  
 {  
 question: question1.trim(),  
 answer: answer1.trim()  
 },  
 {  
 question: question2.trim(),  
 answer: answer2.trim()  
 },  
 {  
 question: question3.trim(),  
 answer: answer3.trim()  
 }  
 ]  
 }  
 setLoader(*true*)  
 axios.post(`${quizzesApi}quizzes`, data).then(res => {  
 *if* (res.data.status === 201) {  
 setLoader(*false*)  
 setMessage(res.data.message)  
 toggleSuccessModal()  
 } *else if* (res.data.status === 409) {  
 setError(res.data.message)  
 }  
 setLoader(*false*)  
 }).catch(error => {  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
 })  
 }  
  
 *return* (  
 <div *className*='quiz-wrapper'>  
 {  
 loader ? (  
 <Loader/>  
 ) : *null* }  
 <div>  
 <Modal *isOpen*={successModal}  
 *toggle*={toggleSuccessModal}  
 *className*='modal-close'>  
 <ModalHeader *toggle*={toggleSuccessModal}  
 *className*='text-uppercase title'>  
 Success!  
 </ModalHeader>  
 <ModalBody>  
 {message}  
 </ModalBody>  
 <ModalFooter>  
 <ButtonComponent *btnText*={'Ok'}  
 *isFullWidth*={*false*}  
 *elementStyle*={'ok-button'}  
 *disabled*={*false*}  
 *onClickFn*={onClick}/>  
 </ModalFooter>  
 </Modal>  
 </div>  
 <div>  
 <div *className*='mb-4'>  
 <ButtonComponent *btnText*={'Quiz List'}  
 *isFullWidth*={*false*}  
 *disabled*={*false*}  
 *onClickFn*={onClick}/>  
 </div>  
 <div>  
 <Card *className*='overflow-hidden'>  
 <div *className*='quiz-header'>  
 <div *className*='text-primary text-center p-4'>  
 <h1 *className*='text-white font-size-20 text-uppercase'>  
 New Quiz  
 </h1>  
 </div>  
 </div>  
 <CardBody *className*='p-4'>  
 <div>  
 <small>  
 {  
 error ? (  
 <span *className*='p-3 error'>  
 {error}  
 </span>  
 ) : *null* }  
 </small>  
 </div>  
 <div *className*='p-3'>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Quiz Title'}  
 *name*={'quizTitle'}  
 *value*={quizTitle}  
 *errorText*={errorQuizTitle}  
 *helperText*={helperQuizTitle}  
 *maxLength*={50}  
 *onChangeFn*={event => onChangeQuizTitle(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Quiz Description'}  
 *name*={'quizDescription'}  
 *value*={quizDescription}  
 *errorText*={errorQuizDescription}  
 *helperText*={helperQuizDescription}  
 *maxLength*={200}  
 *onChangeFn*={event => onChangeQuizDescription(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Question 1'}  
 *name*={'question1'}  
 *value*={question1}  
 *errorText*={errorQuestion1}  
 *helperText*={helperQuestion1}  
 *maxLength*={200}  
 *onChangeFn*={event => onChangeQuestion1(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Answer for Question 1'}  
 *name*={'answer1'}  
 *value*={answer1}  
 *errorText*={errorAnswer1}  
 *helperText*={helperAnswer1}  
 *maxLength*={50}  
 *onChangeFn*={event => onChangeAnswer1(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Question 2'}  
 *name*={'question2'}  
 *value*={question2}  
 *errorText*={errorQuestion2}  
 *helperText*={helperQuestion2}  
 *maxLength*={200}  
 *onChangeFn*={event => onChangeQuestion2(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Answer for Question 2'}  
 *name*={'answer2'}  
 *value*={answer2}  
 *errorText*={errorAnswer2}  
 *helperText*={helperAnswer2}  
 *maxLength*={50}  
 *onChangeFn*={event => onChangeAnswer2(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Question 3'}  
 *name*={'question3'}  
 *value*={question3}  
 *errorText*={errorQuestion3}  
 *helperText*={helperQuestion3}  
 *maxLength*={200}  
 *onChangeFn*={event => onChangeQuestion3(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Answer for Question 3'}  
 *name*={'answer3'}  
 *value*={answer3}  
 *errorText*={errorAnswer3}  
 *helperText*={helperAnswer3}  
 *maxLength*={50}  
 *onChangeFn*={event => onChangeAnswer3(event)}/>  
 </div>  
 <div *className*='text-center mt-4 mb-3'>  
 <ButtonComponent *btnText*={'Submit'}  
 *isFullWidth*={*false*}  
 *elementStyle*={'submit-btn'}  
 *disabled*={isDisabled()}  
 *onClickFn*={onSubmit}/>  
 </div>  
 </div>  
 </CardBody>  
 </Card>  
 </div>  
 </div>  
 </div>  
 )  
}  
  
*export default AddQuizComponent*

**single-quiz.jsx**

*import* React *from* 'react'  
*import* Header *from* '../../../../components/header/header'  
*import Footer from* '../../../../components/footer/footer'  
*import SingleQuizPage from* './single-quiz-component/single-quiz-component'  
*import* './single-quiz.css'  
  
*const SingleQuiz* = props => {  
 *return* (  
 <div>  
 <div>  
 <Header/>  
 </div>  
 <div *className*='container single-quiz-page'>  
 <SingleQuizPage *history*={props.history}/>  
 </div>  
 <div>  
 <Footer/>  
 </div>  
 </div>  
 )  
}  
  
*export default SingleQuiz*

**single-quiz-component.jsx**

*import* React, {*useEffect*, *useState*} *from* 'react'  
*import* {useParams} *from* 'react-router'  
*import* {Card, CardBody, Modal, ModalBody, ModalFooter, ModalHeader} *from* 'reactstrap'  
*import* axios *from* 'axios'  
*import* {*isEmpty*} *from* '../../../../../helpers/common.helpers'  
*import* {quizzesApi} *from* '../../../../../config/api.config'  
*import Loader from* '../../../../../components/loader/loader'  
*import ButtonComponent from* '../../../../../components/button/button'  
*import TextField from* '../../../../../components/text-field/text-field'  
*import* './single-quiz-component.css'  
  
*const SingleQuizComponent* = props => {  
 *const* [successModal, setSuccessModal] = *useState*(*false*)  
 *const* [message, setMessage] = *useState*('')  
  
 *const* helperQuizTitle = 'Please enter a title for the quiz.'  
 *const* helperQuizDescription = 'Please enter a description about the quiz.'  
 *const* helperAnswer1 = 'Please enter the answer for question 1.'  
 *const* helperAnswer2 = 'Please enter the answer for question 2.'  
 *const* helperAnswer3 = 'Please enter the answer for question 3.'  
 *const* helperQuestion1 = 'Please enter question 1.'  
 *const* helperQuestion2 = 'Please enter question 2.'  
 *const* helperQuestion3 = 'Please enter question 3.'  
  
 *const* [loader, setLoader] = *useState*(*false*)  
  
 *const* [quizTitle, setQuizTitle] = *useState*('')  
 *const* [quizDescription, setQuizDescription] = *useState*('')  
 *const* [question1, setQuestion1] = *useState*('')  
 *const* [answer1, setAnswer1] = *useState*('')  
 *const* [question2, setQuestion2] = *useState*('')  
 *const* [answer2, setAnswer2] = *useState*('')  
 *const* [question3, setQuestion3] = *useState*('')  
 *const* [answer3, setAnswer3] = *useState*('')  
  
 *const* [errorQuizTitle, setErrorQuizTitle] = *useState*('')  
 *const* [errorQuizDescription, setErrorQuizDescription] = *useState*('')  
 *const* [errorQuestion1, setErrorQuestion1] = *useState*('')  
 *const* [errorAnswer1, setErrorAnswer1] = *useState*('')  
 *const* [errorQuestion2, setErrorQuestion2] = *useState*('')  
 *const* [errorAnswer2, setErrorAnswer2] = *useState*('')  
 *const* [errorQuestion3, setErrorQuestion3] = *useState*('')  
 *const* [errorAnswer3, setErrorAnswer3] = *useState*('')  
  
 *const* [quizTitleValid, setQuizTitleValid] = *useState*(*true*)  
 *const* [quizDescriptionValid, setQuizDescriptionValid] = *useState*(*true*)  
 *const* [question1Valid, setQuestion1Valid] = *useState*(*true*)  
 *const* [answer1Valid, setAnswer1Valid] = *useState*(*true*)  
 *const* [question2Valid, setQuestion2Valid] = *useState*(*true*)  
 *const* [answer2Valid, setAnswer2Valid] = *useState*(*true*)  
 *const* [question3Valid, setQuestion3Valid] = *useState*(*true*)  
 *const* [answer3Valid, setAnswer3Valid] = *useState*(*true*)  
  
 *const* [error, setError] = *useState*('')  
  
 *const* {  
 id  
 } = useParams()  
  
 *useEffect*(() => {  
 loadData().then(() => {  
 })  
 }, [])  
  
 *const* loadData = *async* () => {  
 setLoader(*true*)  
 axios.get(`${quizzesApi}quizzes/${id}`).then(res => {  
 *let* data = res.data.quiz  
 setQuizTitle(data.quizTitle)  
 setQuizDescription(data.quizDescription)  
 setQuestion1(data.questions[0].question)  
 setAnswer1(data.questions[0].answer)  
 setQuestion2(data.questions[1].question)  
 setAnswer2(data.questions[1].answer)  
 setQuestion3(data.questions[2].question)  
 setAnswer3(data.questions[2].answer)  
 setLoader(*false*)  
 }).catch(error => {  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
 })  
 }  
  
 *const* onChangeQuizTitle = *async* event => {  
 setQuizTitle(event.value)  
 setQuizTitleValid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorQuizTitle('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorQuizTitle('Please enter a valid quiz title.')  
 }  
 }  
  
 *const* onChangeQuizDescription = *async* event => {  
 setQuizDescription(event.value)  
 setQuizDescriptionValid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorQuizDescription('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorQuizDescription('Please enter a valid quiz description.')  
 }  
 }  
  
 *const* onChangeQuestion1 = *async* event => {  
 setQuestion1(event.value)  
 setQuestion1Valid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorQuestion1('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorQuestion1('Please enter a valid question.')  
 }  
 }  
  
 *const* onChangeQuestion2 = *async* event => {  
 setQuestion2(event.value)  
 setQuestion2Valid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorQuestion2('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorQuestion2('Please enter a valid question.')  
 }  
 }  
  
 *const* onChangeQuestion3 = *async* event => {  
 setQuestion3(event.value)  
 setQuestion3Valid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorQuestion3('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorQuestion3('Please enter a valid question.')  
 }  
 }  
  
 *const* onChangeAnswer1 = *async* event => {  
 setAnswer1(event.value)  
 setAnswer1Valid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorAnswer1('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorAnswer1('Please enter a valid answer.')  
 }  
 }  
  
 *const* onChangeAnswer2 = *async* event => {  
 setAnswer2(event.value)  
 setAnswer2Valid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorAnswer2('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorAnswer2('Please enter a valid answer.')  
 }  
 }  
  
 *const* onChangeAnswer3 = *async* event => {  
 setAnswer3(event.value)  
 setAnswer3Valid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorAnswer3('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorAnswer3('Please enter a valid answer.')  
 }  
 }  
  
 *function* isDisabled() {  
 *return* !quizTitleValid || !quizDescriptionValid || !question1Valid || !question2Valid || !question3Valid ||  
 !answer1Valid || !answer2Valid || !answer3Valid  
 }  
  
 *const* toggleSuccessModal = *async* () => {  
 setSuccessModal(!successModal)  
 }  
  
 *const* onClick = *async* () => {  
 props.history.push('/content-management')  
 }  
  
 *const* onSubmit = *async* () => {  
 setError('')  
 *const* data = {  
 'quizTitle': quizTitle.trim(),  
 'quizDescription': quizDescription.trim(),  
 'questions': [  
 {  
 question: question1.trim(),  
 answer: answer1.trim()  
 },  
 {  
 question: question2.trim(),  
 answer: answer2.trim()  
 },  
 {  
 question: question3.trim(),  
 answer: answer3.trim()  
 }  
 ]  
 }  
 setLoader(*true*)  
 axios.put(`${quizzesApi}quizzes/${id}`, data).then(res => {  
 *if* (res.data.status === 200) {  
 setLoader(*false*)  
 setMessage(res.data.message)  
 toggleSuccessModal()  
 } *else if* (res.data.status === 409) {  
 setError(res.data.message)  
 }  
 setLoader(*false*)  
 }).catch(error => {  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
 })  
 }  
  
 *return* (  
 <div *className*='quiz-wrapper'>  
 {  
 loader ? (  
 <Loader/>  
 ) : *null* }  
 <div>  
 <Modal *isOpen*={successModal}  
 *toggle*={toggleSuccessModal}  
 *className*='modal-close'>  
 <ModalHeader *toggle*={toggleSuccessModal}  
 *className*='text-uppercase title'>  
 Success!  
 </ModalHeader>  
 <ModalBody>  
 {message}  
 </ModalBody>  
 <ModalFooter>  
 <ButtonComponent *btnText*={'Ok'}  
 *isFullWidth*={*false*}  
 *elementStyle*={'ok-button'}  
 *disabled*={*false*}  
 *onClickFn*={onClick}/>  
 </ModalFooter>  
 </Modal>  
 </div>  
 <div>  
 <div *className*='mb-4'>  
 <ButtonComponent *btnText*={'Quiz List'}  
 *isFullWidth*={*false*}  
 *disabled*={*false*}  
 *onClickFn*={onClick}/>  
 </div>  
 <div>  
 <Card *className*='overflow-hidden'>  
 <div *className*='quiz-header'>  
 <div *className*='text-primary text-center p-4'>  
 <h1 *className*='text-white font-size-20 text-uppercase'>  
 QUIZ - {quizTitle}  
 </h1>  
 </div>  
 </div>  
 <CardBody *className*='p-4'>  
 <div>  
 <small>  
 {  
 error ? (  
 <span *className*='p-3 error'>  
 {error}  
 </span>  
 ) : *null* }  
 </small>  
 </div>  
 <div *className*='p-3'>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Quiz Title'}  
 *name*={'quizTitle'}  
 *value*={quizTitle}  
 *errorText*={errorQuizTitle}  
 *helperText*={helperQuizTitle}  
 *maxLength*={50}  
 *onChangeFn*={event => onChangeQuizTitle(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Quiz Description'}  
 *name*={'quizDescription'}  
 *value*={quizDescription}  
 *errorText*={errorQuizDescription}  
 *helperText*={helperQuizDescription}  
 *maxLength*={200}  
 *onChangeFn*={event => onChangeQuizDescription(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Question 1'}  
 *name*={'question1'}  
 *value*={question1}  
 *errorText*={errorQuestion1}  
 *helperText*={helperQuestion1}  
 *maxLength*={200}  
 *onChangeFn*={event => onChangeQuestion1(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Answer for Question 1'}  
 *name*={'answer1'}  
 *value*={answer1}  
 *errorText*={errorAnswer1}  
 *helperText*={helperAnswer1}  
 *maxLength*={50}  
 *onChangeFn*={event => onChangeAnswer1(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Question 2'}  
 *name*={'question2'}  
 *value*={question2}  
 *errorText*={errorQuestion2}  
 *helperText*={helperQuestion2}  
 *maxLength*={200}  
 *onChangeFn*={event => onChangeQuestion2(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Answer for Question 2'}  
 *name*={'answer2'}  
 *value*={answer2}  
 *errorText*={errorAnswer2}  
 *helperText*={helperAnswer2}  
 *maxLength*={50}  
 *onChangeFn*={event => onChangeAnswer2(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Question 3'}  
 *name*={'question3'}  
 *value*={question3}  
 *errorText*={errorQuestion3}  
 *helperText*={helperQuestion3}  
 *maxLength*={200}  
 *onChangeFn*={event => onChangeQuestion3(event)}/>  
 </div>  
 <div>  
 <TextField *isRequired*={*true*}  
 *labelText*={'Answer for Question 3'}  
 *name*={'answer3'}  
 *value*={answer3}  
 *errorText*={errorAnswer3}  
 *helperText*={helperAnswer3}  
 *maxLength*={50}  
 *onChangeFn*={event => onChangeAnswer3(event)}/>  
 </div>  
 <div *className*='text-center mt-4 mb-3'>  
 <ButtonComponent *btnText*={'Update'}  
 *isFullWidth*={*false*}  
 *elementStyle*={'submit-btn'}  
 *disabled*={isDisabled()}  
 *onClickFn*={onSubmit}/>  
 </div>  
 </div>  
 </CardBody>  
 </Card>  
 </div>  
 </div>  
 </div>  
 )  
}  
  
*export default SingleQuizComponent*

**4. Playing Quizzes - Frontend**

**quizzes.jsx**

*import* React *from* 'react'  
*import* Header *from* '../../../../components/header/header'  
*import Footer from* '../../../../components/footer/footer'  
*import QuizzesComponent from* './quizzes-component/quizzes-component'  
*import* './quizzes.css'  
  
*const Quizzes* = props => {  
 *return* (  
 <div>  
 <div>  
 <Header/>  
 </div>  
 <h1 *className*='text-center text-uppercase mt-5 page-title'>  
 Quizzes  
 </h1>  
 <div *className*='container quizzes-page'>  
 <QuizzesComponent *history*={props.history}/>  
 </div>  
 <div>  
 <Footer/>  
 </div>  
 </div>  
 )  
}  
  
*export default Quizzes*

**quizzes-component.jsx**

*import* React, {*useEffect*, *useState*} *from* 'react'  
*import* {Card, CardBody, CardDeck, CardFooter, CardText, CardTitle} *from* 'reactstrap'  
*import* axios *from* 'axios'  
*import* {quizzesApi} *from* '../../../../../config/api.config'  
*import Loader from* '../../../../../components/loader/loader'  
*import* './quizzes-component.css'  
  
*const QuizzesComponent* = props => {  
 *const* [loader, setLoader] = *useState*(*false*)  
 *const* [data, setData] = *useState*(*null*)  
 *const* [error, setError] = *useState*('')  
  
 *useEffect*(() => {  
 loadData().then(() => {  
 })  
 }, [])  
  
 *const* loadData = *async* () => {  
 setLoader(*true*)  
 axios.get(`${quizzesApi}quizzes`).then(res => {  
 setData(res.data.quizList)  
 setLoader(*false*)  
 }).catch(error => {  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
 })  
 }  
  
 *const* onView = *async* id => {  
 props.history.push('/play/' + id)  
 }  
  
 *return* (  
 <div>  
 {  
 loader ? (  
 <Loader/>  
 ) : *null* }  
 <div>  
 <small>  
 {  
 error ? (  
 <span *className*='error'>  
 {error}  
 </span>  
 ) : *null* }  
 </small>  
 </div>  
 <div>  
 <CardDeck *className*='card-deck'>  
 <div>  
 {  
 data && data.map(item => {  
 *return* (  
 <div>  
 <Card *key*={item.\_id}  
 *title*='Try Now!'  
 *className*='m-4 card-item justify-content-center'  
 *onClick*={() => onView(item.\_id)}>  
 <CardBody>  
 <CardTitle *className*='text-uppercase text-center m-4'  
 *tag*='h2'>  
 <label>  
 {item.quizTitle}  
 </label>  
 </CardTitle>  
 <CardText *className*='m-4 text-center'>  
 <label>  
 {item.quizDescription}  
 </label>  
 </CardText>  
 </CardBody>  
 <CardFooter *className*='text-center text-uppercase text-primary'>  
 <label>Try Now!</label>  
 </CardFooter>  
 </Card>  
 </div>  
 )  
 })  
 }  
 </div>  
 </CardDeck>  
 </div>  
 </div>  
 )  
}  
  
*export default QuizzesComponent*

**play.jsx**

*import* React *from* 'react'  
*import* Header *from* '../../../../components/header/header'  
*import Footer from* '../../../../components/footer/footer'  
*import PlayComponent from* './play-component/play-component'  
*import* './play.css'  
  
*const Play* = props => {  
 *return* (  
 <div>  
 <div>  
 <Header/>  
 </div>  
 <div *className*='container play-page'>  
 <PlayComponent *history*={props.history}/>  
 </div>  
 <div>  
 <Footer/>  
 </div>  
 </div>  
 )  
}  
  
*export default Play*

**play-component.jsx**

*import* React, {*useEffect*, *useState*} *from* 'react'  
*import* {useParams} *from* 'react-router'  
*import* useWindowSize *from* 'react-use/lib/useWindowSize'  
*import* {Modal, ModalBody, ModalFooter, ModalHeader} *from* 'reactstrap'  
*import* Confetti *from* 'react-confetti'  
*import* axios *from* 'axios'  
*import* {*isEmpty*} *from* '../../../../../helpers/common.helpers'  
*import* {quizzesApi} *from* '../../../../../config/api.config'  
*import Loader from* '../../../../../components/loader/loader'  
*import ButtonComponent from* '../../../../../components/button/button'  
*import TextField from* '../../../../../components/text-field/text-field'  
*import* './play-component.css'  
  
*const PlayComponent* = props => {  
 *const* {width, height} = useWindowSize()  
  
 *const* [successModal, setSuccessModal] = *useState*(*false*)  
  
 *const* helperAnswer1 = 'Please enter the answer for question 1.'  
 *const* helperAnswer2 = 'Please enter the answer for question 2.'  
 *const* helperAnswer3 = 'Please enter the answer for question 3.'  
  
 *const* [loader, setLoader] = *useState*(*false*)  
  
 *const* [userAnswer1, setUserAnswer1] = *useState*('')  
 *const* [userAnswer2, setUserAnswer2] = *useState*('')  
 *const* [userAnswer3, setUserAnswer3] = *useState*('')  
  
 *const* [data, setData] = *useState*(*null*)  
  
 *const* [errorAnswer1, setErrorAnswer1] = *useState*('')  
 *const* [errorAnswer2, setErrorAnswer2] = *useState*('')  
 *const* [errorAnswer3, setErrorAnswer3] = *useState*('')  
  
 *const* [answer1Valid, setAnswer1Valid] = *useState*(*false*)  
 *const* [answer2Valid, setAnswer2Valid] = *useState*(*false*)  
 *const* [answer3Valid, setAnswer3Valid] = *useState*(*false*)  
  
 *const* [markResults, setMarkResults] = *useState*(*false*)  
  
 *const* [answer1Correct, setAnswer1Correct] = *useState*(*false*)  
 *const* [answer2Correct, setAnswer2Correct] = *useState*(*false*)  
 *const* [answer3Correct, setAnswer3Correct] = *useState*(*false*)  
  
 *const* [error, setError] = *useState*('')  
  
 *const* {  
 id  
 } = useParams()  
  
 *useEffect*(() => {  
 loadData().then(() => {  
 })  
 }, [])  
  
 *const* loadData = *async* () => {  
 setLoader(*true*)  
 axios.get(`${quizzesApi}quizzes/${id}`).then(res => {  
 setData(res.data.quiz)  
 setLoader(*false*)  
 }).catch(error => {  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
 })  
 }  
  
 *const* onChangeAnswer1 = *async* event => {  
 setMarkResults(*false*)  
 setUserAnswer1(event.value)  
 setAnswer1Valid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorAnswer1('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorAnswer1('Please enter a valid answer.')  
 }  
 *if* (event.value.trim() === data.questions[0].answer) {  
 setAnswer1Correct(*true*)  
 } *else* {  
 setAnswer1Correct(*false*)  
 }  
 }  
  
 *const* onChangeAnswer2 = *async* event => {  
 setMarkResults(*false*)  
 setUserAnswer2(event.value)  
 setAnswer2Valid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorAnswer2('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorAnswer2('Please enter a valid answer.')  
 }  
 *if* (event.value.trim() === data.questions[1].answer) {  
 setAnswer2Correct(*true*)  
 } *else* {  
 setAnswer2Correct(*false*)  
 }  
 }  
  
 *const* onChangeAnswer3 = *async* event => {  
 setMarkResults(*false*)  
 setUserAnswer3(event.value)  
 setAnswer3Valid(event.eventInfo.target.validity.valid && !*await isEmpty*(event.value))  
 setErrorAnswer3('')  
 *if* (!event.eventInfo.target.validity.valid) {  
 setErrorAnswer3('Please enter a valid answer.')  
 }  
 *if* (event.value.trim() === data.questions[2].answer) {  
 setAnswer3Correct(*true*)  
 } *else* {  
 setAnswer3Correct(*false*)  
 }  
 }  
  
 *function* isDisabled() {  
 *return* !answer1Valid || !answer2Valid || !answer3Valid  
 }  
  
 *const* toggleSuccessModal = *async* () => {  
 setSuccessModal(!successModal)  
 }  
  
 *const* onClick = *async* () => {  
 props.history.push('/quizzes')  
 }  
  
 *const* onSubmit = *async* () => {  
 setMarkResults(*true*)  
 *if* (answer1Correct && answer2Correct && answer3Correct) {  
 *await* toggleSuccessModal()  
 }  
 }  
  
 *return* (  
 <div>  
 {  
 loader ? (  
 <Loader/>  
 ) : *null* }  
 <div>  
 {  
 markResults && answer1Correct && answer2Correct && answer3Correct ? (  
 <Confetti *width*={width}  
 *height*={height}/>  
 ) : *null* }  
 </div>  
 <div>  
 <Modal *isOpen*={successModal}  
 *toggle*={toggleSuccessModal}  
 *className*='modal-close'>  
 <ModalHeader *toggle*={toggleSuccessModal}  
 *className*='text-uppercase title'>  
 Congratulations!  
 </ModalHeader>  
 <ModalBody>  
 You got all answers correct!  
 </ModalBody>  
 <ModalFooter>  
 <ButtonComponent *btnText*={'Ok'}  
 *isFullWidth*={*false*}  
 *elementStyle*={'ok-button'}  
 *disabled*={*false*}  
 *onClickFn*={onClick}/>  
 </ModalFooter>  
 </Modal>  
 </div>  
 <div>  
 <div>  
 <small>  
 {  
 error ? (  
 <span *className*='p-3 error'>  
 {error}  
 </span>  
 ) : *null* }  
 </small>  
 </div>  
 {  
 data ? (  
 <div>  
 <div>  
 <h1 *className*='text-center text-uppercase m-4 page-title'>  
 {data.quizTitle}  
 </h1>  
 <label *className*='m-4 mb-5'>  
 {data.quizDescription}  
 </label>  
 </div>  
 <div>  
 <div *className*='mb-5'>  
 <TextField *isRequired*={*true*}  
 *labelText*={data.questions[0].question}  
 *name*={'answer1'}  
 *value*={userAnswer1}  
 *errorText*={errorAnswer1}  
 *helperText*={helperAnswer1}  
 *maxLength*={50}  
 *onChangeFn*={event => onChangeAnswer1(event)}/>  
 {  
 markResults && answer1Correct ? (  
 <span>  
 <small *className*='text-success'>  
 Correct Answer!  
 </small>  
 </span>  
 ) : markResults && !answer1Correct ? (  
 <span>  
 <small *className*='text-danger'>  
 Sorry, You got this wrong. Please try again!  
 </small>  
 </span>  
 ) : *null* }  
 </div>  
 <div *className*='mb-5'>  
 <TextField *isRequired*={*true*}  
 *labelText*={data.questions[1].question}  
 *name*={'answer2'}  
 *value*={userAnswer2}  
 *errorText*={errorAnswer2}  
 *helperText*={helperAnswer2}  
 *maxLength*={50}  
 *onChangeFn*={event => onChangeAnswer2(event)}/>  
 {  
 markResults && answer2Correct ? (  
 <span>  
 <small *className*='text-success'>  
 Correct Answer!  
 </small>  
 </span>  
 ) : markResults && !answer2Correct ? (  
 <span>  
 <small *className*='text-danger'>  
 Sorry, You got this wrong. Please try again!  
 </small>  
 </span>  
 ) : *null* }  
 </div>  
 <div *className*='mb-5'>  
 <TextField *isRequired*={*true*}  
 *labelText*={data.questions[2].question}  
 *name*={'answer3'}  
 *value*={userAnswer3}  
 *errorText*={errorAnswer3}  
 *helperText*={helperAnswer3}  
 *maxLength*={50}  
 *onChangeFn*={event => onChangeAnswer3(event)}/>  
 {  
 markResults && answer3Correct ? (  
 <span>  
 <small *className*='text-success'>  
 Correct Answer!  
 </small>  
 </span>  
 ) : markResults && !answer3Correct ? (  
 <span>  
 <small *className*='text-danger'>  
 Sorry, You got this wrong. Please try again!  
 </small>  
 </span>  
 ) : *null* }  
 </div>  
 <div *className*='text-center mt-5 mb-5'>  
 <ButtonComponent *btnText*={'Submit'}  
 *isFullWidth*={*false*}  
 *elementStyle*={'submit-btn'}  
 *disabled*={isDisabled()}  
 *onClickFn*={onSubmit}/>  
 </div>  
 </div>  
 </div>  
 ) : *null* }  
 </div>  
 </div>  
 )  
}  
  
*export default PlayComponent*

**RESTful web service (implemented code)**

**auth-controller.js**

*const* bcrypt = require('bcrypt')  
*const* UserModel = require('../models/users.model')  
  
*const* login = *async* (req, res) => {  
 *let* user  
  
 *const* {  
 email,  
 password  
 } = req.body  
  
 *try* {  
 user = *await* UserModel.findOne({  
 email: email  
 })  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 *if* (user && bcrypt.compareSync(password, user.password)) {  
 res.send({  
 status: 200,  
 user: user  
 })  
 } *else* {  
 res.send({  
 status: 401,  
 message: 'Incorrect email or password! Please double check and try again.'  
 })  
 }  
}  
  
module.exports = {  
 login  
}

**auth.routes.js**

*const* express = require('express')  
*const* AuthController = require('../controllers/auth-controller')  
  
*const* router = express.*Router*()  
  
router.post('/login', AuthController.login)  
  
module.exports = router

**users-controller.js**

*const* bcrypt = require('bcrypt')  
*const* nodemailer = require('nodemailer')  
*const* UserModel = require('../models/users.model')  
  
require('dotenv').*config*()  
  
*const* addUser = *async* (req, res) => {  
 *let* existingUser  
  
 *let* {  
 firstName,  
 lastName,  
 email,  
 password  
 } = req.body  
  
 *try* {  
 existingUser = *await* UserModel.findOne({  
 email: email  
 })  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 *if* (existingUser) {  
 res.send({  
 status: 409,  
 message: 'A user with the same email already exists.'  
 })  
 }  
  
 *const* newUser = *new* UserModel({  
 firstName,  
 lastName,  
 email,  
 password: bcrypt.hashSync(password, 10)  
 })  
  
 *try* {  
 *await* newUser.save()  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 *await* sendEmail(email)  
  
 res.send({  
 status: 201,  
 message: 'Congratulations! You have successfully registered as a student in iLearn. ' +  
 'Now please login to iLearn and start your learning journey right now!'  
 })  
}  
  
*const* addAdmin = *async* (req, res) => {  
 *let* existingUser  
  
 *let* {  
 firstName,  
 lastName,  
 email,  
 password  
 } = req.body  
  
 *try* {  
 existingUser = *await* UserModel.findOne({  
 email: email  
 })  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 *if* (existingUser) {  
 res.send({  
 status: 409,  
 message: 'A user with the same email already exists.'  
 })  
 }  
  
 *const* newUser = *new* UserModel({  
 firstName,  
 lastName,  
 email,  
 password: bcrypt.hashSync(password, 10),  
 userType: 'Admin'  
 })  
  
 *try* {  
 *await* newUser.save()  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 res.send({  
 status: 201,  
 message: 'Administrator added successfully!'  
 })  
}  
  
*let* transporter = nodemailer.createTransport({  
 service: 'gmail',  
 auth: {  
 user: 'it18149654@gmail.com',  
 pass: process.env.PASSWORD  
 }  
})  
  
*const* sendEmail = *async* email => {  
 *let* info = {  
 from: 'it18149654@gmail.com',  
 to: email,  
 subject: 'Welcome to iLearn!',  
 text:  
 `Congratulations!  
 You have successfully registered as a student.  
 Start your learning journey today!  
 Thank you!  
 This is an auto-generated email.  
 If this has been sent by mistake, please delete this without sharing this.  
 All rights reserved.`,  
 html:  
 `<div style="margin: 0; padding: 0; background-color: #f2f2f2; font-family: arial, serif;">  
 <table style="margin: 0 auto; background: white; max-width: 500px; padding-bottom: 0; border-top: 5px solid #588dde; border-bottom: 5px solid #588dde; width: 100%;">  
 <tr style="background: rgb(237, 243, 255); padding-left: 20px; padding-right: 20px;">  
 <td>  
 <table align="left" style="width: 100%;">  
 <tr>  
 <td style="padding: 10px;">  
 <h1 style="text-align: center; color: #1a1a72;">Congratulations!</h1>  
 <h2 style="margin-top:25px; margin-bottom: 0; color: #4db0c4; font-weight: 400; font-size: medium;">You have successfully registered as a student.</h2>  
 <h2 style="margin-top:20px; margin-bottom: 0; color: #4db0c4; font-weight: 400; font-size: medium;">Start your learning journey today!</h2>  
 </td>  
 </tr>  
 <tr style="background: rgb(237, 243, 255); padding-left: 20px; padding-right: 20px;">  
 <td>  
 <table align="left" style="width: 100%;">  
 <tr>  
 <td style="padding: 10px;">  
 <h2 style="margin-top:25px; margin-bottom: 0; color: #4db0c4; font-weight: 400; font-size: medium;">Thank you!</h2>  
 <h4 style="margin-top:20px; margin-bottom: 0; color: #4db0c4; font-weight: 400; font-size: medium;">This is an auto-generated email.</h4>  
 <h4 style="margin-top:20px; margin-bottom: 0; color: #4db0c4; font-weight: 400; font-size: medium;">If this has been sent by mistake, please delete this without sharing this.</h4>  
 <h4 style="margin-top:20px; margin-bottom: 0; color: #4db0c4; font-weight: 400; font-size: medium;">All rights reserved.</h4>  
 </td>  
 </tr>  
 </table>  
 </td>  
 </tr>  
 </table>  
 </div>`  
 }  
  
 transporter.sendMail(info, (error, data) => {  
 *if* (error) {  
 *console*.error(error)  
 *console*.error('Email sending failed! Please try again.')  
 } *else* {  
 *console*.error(data)  
 *console*.error('An email is sent successfully to ' + email + '.')  
 }  
 })  
}  
  
*const* updateUser = *async* (req, res) => {  
 *let* user  
 *let* existingUser  
  
 *const* {  
 id  
 } = req.params  
  
 *const* {  
 firstName,  
 lastName,  
 email,  
 password,  
 userType  
 } = req.body  
  
 *try* {  
 user = *await* UserModel.findById(id)  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 *try* {  
 existingUser = *await* UserModel.findOne({  
 email: email  
 })  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 *if* (existingUser && email !== user.email) {  
 res.send({  
 status: 409,  
 message: 'A user with the same email already exists.'  
 })  
 }  
  
 user.firstName = firstName  
 user.lastName = lastName  
 user.email = email  
 user.password = bcrypt.hashSync(password, 10)  
 user.userType = userType  
  
 *try* {  
 *await* user.save()  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 res.send({  
 status: 200,  
 message: 'User updated successfully!'  
 })  
}  
  
*const* promoteUser = *async* (req, res) => {  
 *let* user  
  
 *const* {  
 id  
 } = req.params  
  
 *try* {  
 user = *await* UserModel.findById(id)  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 user.userType = 'Admin'  
  
 *try* {  
 *await* user.save()  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 res.send({  
 status: 200,  
 message: 'The user has been successfully promoted as a teacher!'  
 })  
}  
  
*const* deleteUser = *async* (req, res) => {  
 *let* user  
  
 *const* {  
 id  
 } = req.params  
  
 *try* {  
 user = *await* UserModel.findById(id)  
 *await* user.remove()  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 res.send({  
 status: 200,  
 message: 'User deleted successfully!'  
 })  
}  
  
*const* getUser = *async* (req, res) => {  
 *let* user  
  
 *const* {  
 id  
 } = req.params  
  
 *try* {  
 user = *await* UserModel.findById(id)  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 res.send({  
 status: 200,  
 user: user  
 })  
}  
  
*const* getUserList = *async* (req, res) => {  
 *let* userList  
  
 *try* {  
 userList = *await* UserModel.find()  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 res.send({  
 status: 200,  
 userList: userList  
 })  
}  
  
exports.addUser = addUser  
exports.addAdmin = addAdmin  
exports.updateUser = updateUser  
exports.promoteUser = promoteUser  
exports.deleteUser = deleteUser  
exports.getUser = getUser  
exports.getUserList = getUserList

**users.routes.js**

*const* express = require('express')  
*const* UsersController = require('../controllers/users-controller')  
  
*const* router = express.*Router*()  
  
router.post('/users', UsersController.addUser)  
router.post('/admin', UsersController.addAdmin)  
router.put('/users/:id', UsersController.updateUser)  
router.put('/users/promote/:id', UsersController.promoteUser)  
router.delete('/users/:id', UsersController.deleteUser)  
router.get('/users/:id', UsersController.getUser)  
router.get('/users', UsersController.getUserList)  
  
module.exports = router

**quizzes-controller.js**

*const* QuizModel = require('../models/quizzes.model')  
  
*const* addQuiz = *async* (req, res) => {  
 *let* existingQuiz  
  
 *let* {  
 quizTitle,  
 quizDescription,  
 questions  
 } = req.body  
  
 *try* {  
 existingQuiz = *await* QuizModel.findOne({  
 quizTitle: quizTitle  
 })  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 *if* (existingQuiz) {  
 res.send({  
 status: 409,  
 message: 'A quiz with the same title already exists.'  
 })  
 }  
  
 *const* newQuiz = *new* QuizModel({  
 quizTitle,  
 quizDescription,  
 questions  
 })  
  
 *try* {  
 *await* newQuiz.save()  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 res.send({  
 status: 201,  
 message: 'New quiz added successfully!'  
 })  
}  
  
*const* updateQuiz = *async* (req, res) => {  
 *let* quiz  
 *let* existingQuiz  
  
 *const* {  
 id  
 } = req.params  
  
 *const* {  
 quizTitle,  
 quizDescription,  
 questions  
 } = req.body  
  
 *try* {  
 quiz = *await* QuizModel.findById(id)  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 *try* {  
 existingQuiz = *await* QuizModel.findOne({  
 quizTitle: quizTitle  
 })  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 *if* (existingQuiz && quizTitle !== quiz.quizTitle) {  
 res.send({  
 status: 409,  
 message: 'A quiz with the same title already exists.'  
 })  
 }  
  
 quiz.quizTitle = quizTitle  
 quiz.quizDescription = quizDescription  
 quiz.questions = questions  
  
 *try* {  
 *await* quiz.save()  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 res.send({  
 status: 200,  
 message: 'Quiz updated successfully!'  
 })  
}  
  
*const* deleteQuiz = *async* (req, res) => {  
 *let* quiz  
  
 *const* {  
 id  
 } = req.params  
  
 *try* {  
 quiz = *await* QuizModel.findById(id)  
 *await* quiz.remove()  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 res.send({  
 status: 200,  
 message: 'Quiz deleted successfully!'  
 })  
}  
  
*const* getQuiz = *async* (req, res) => {  
 *let* quiz  
  
 *const* {  
 id  
 } = req.params  
  
 *try* {  
 quiz = *await* QuizModel.findById(id)  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 res.send({  
 status: 200,  
 quiz: quiz  
 })  
}  
  
*const* getQuizList = *async* (req, res) => {  
 *let* quizList  
  
 *try* {  
 quizList = *await* QuizModel.find()  
 } *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
 }  
  
 res.send({  
 status: 200,  
 quizList: quizList  
 })  
}  
  
exports.addQuiz = addQuiz  
exports.updateQuiz = updateQuiz  
exports.deleteQuiz = deleteQuiz  
exports.getQuiz = getQuiz  
exports.getQuizList = getQuizList

**quizzes.routes.js**

*const* express = require('express')  
*const* QuizController = require('../controllers/quizzes-controller')  
  
*const* router = express.*Router*()  
  
router.post('/quizzes', QuizController.addQuiz)  
router.put('/quizzes/:id', QuizController.updateQuiz)  
router.delete('/quizzes/:id', QuizController.deleteQuiz)  
router.get('/quizzes/:id', QuizController.getQuiz)  
router.get('/quizzes', QuizController.getQuizList)  
  
module.exports = router

**main.config.js**

*export const* authStoreKey = '@af2021'  
*export const* baseApi = 'http://localhost:5000/'

**api.config.js**

*import* {baseApi} *from* './main.config'  
  
*export const* authApi = `${baseApi}auth/`  
*export const* uploadsApi = `${baseApi}uploads/`  
*export const* usersApi = `${baseApi}users/`  
*export const* quizzesApi = `${baseApi}quizzes/`

**quizzes.component.jsx**

*const* loadData = *async* () => {  
 setLoader(*true*)  
 axios.get(`${quizzesApi}quizzes`).then(res => {  
 setData(res.data.quizList)  
 setLoader(*false*)  
 }).catch(error => {  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
 })  
}

**login-form.jsx**

axios.post(`${authApi}login`, data).then(res => {  
 *if* (res.data.status === 200) {  
 *setLocalStorageItem*(authStoreKey, res.data.user)  
 appContext.login(res.data.user)  
 props.history.push('/home')  
 } *else if* (res.data.status === 401) {  
 setError(res.data.message)  
 }  
 setLoader(*false*)  
}).catch(error => {  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
})

**register-form.jsx**

axios.post(`${usersApi}users`, data).then(res => {  
 *if* (res.data.status === 201) {  
 setLoader(*false*)  
 setMessage(res.data.message)  
 toggleSuccessModal()  
 } *else if* (res.data.status === 409) {  
 setError(res.data.message)  
 }  
 setLoader(*false*)  
}).catch(error => {  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
})

**user-management-component.jsx**

axios.get(`${usersApi}users`).then(res => {  
 setData(res.data.userList)  
 setLoader(*false*)  
}).catch(error => {  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
})

axios.put(`${usersApi}users/promote/${editId}`).then(res => {  
 *if* (res.data.status === 200) {  
 setMessage(res.data.message)  
 toggleEdit()  
 toggleSuccessModalEdit()  
 loadData()  
 } *else* {  
 toggleEdit()  
 setError('An unexpected error occurred. Please try again later.')  
 *console*.error(error)  
 }  
 setLoader(*false*)  
}).catch(error => {  
 toggleEdit()  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
})

axios.delete(`${usersApi}users/${deleteId}`).then(res => {  
 *if* (res.data.status === 200) {  
 setData(data.filter(item => item.\_id !== deleteId))  
 setMessage(res.data.message)  
 toggle()  
 toggleSuccessModal()  
 } *else* {  
 toggle()  
 setError('An unexpected error occurred. Please try again later.')  
 *console*.error(error)  
 }  
 setLoader(*false*)  
}).catch(error => {  
 toggle()  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
})

**quiz-list-component.jsx**

axios.get(`${quizzesApi}quizzes`).then(res => {  
 setData(res.data.quizList)  
 setLoader(*false*)  
}).catch(error => {  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
})

axios.delete(`${quizzesApi}quizzes/${deleteId}`).then(res => {  
 *if* (res.data.status === 200) {  
 setData(data.filter(item => item.\_id !== deleteId))  
 setMessage(res.data.message)  
 toggle()  
 toggleSuccessModal()  
 } *else* {  
 toggle()  
 setError('An unexpected error occurred. Please try again later.')  
 *console*.error(error)  
 }  
 setLoader(*false*)  
}).catch(error => {  
 toggle()  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
})

**add-quiz-component.jsx**

axios.post(`${quizzesApi}quizzes`, data).then(res => {  
 *if* (res.data.status === 201) {  
 setLoader(*false*)  
 setMessage(res.data.message)  
 toggleSuccessModal()  
 } *else if* (res.data.status === 409) {  
 setError(res.data.message)  
 }  
 setLoader(*false*)  
}).catch(error => {  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
})

**single-quiz-component.jsx**

axios.get(`${quizzesApi}quizzes/${id}`).then(res => {  
 *let* data = res.data.quiz  
 setQuizTitle(data.quizTitle)  
 setQuizDescription(data.quizDescription)  
 setQuestion1(data.questions[0].question)  
 setAnswer1(data.questions[0].answer)  
 setQuestion2(data.questions[1].question)  
 setAnswer2(data.questions[1].answer)  
 setQuestion3(data.questions[2].question)  
 setAnswer3(data.questions[2].answer)  
 setLoader(*false*)  
}).catch(error => {  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
})

axios.put(`${quizzesApi}quizzes/${id}`, data).then(res => {  
 *if* (res.data.status === 200) {  
 setLoader(*false*)  
 setMessage(res.data.message)  
 toggleSuccessModal()  
 } *else if* (res.data.status === 409) {  
 setError(res.data.message)  
 }  
 setLoader(*false*)  
}).catch(error => {  
 setError('An unexpected error occurred. Please try again later.')  
 setLoader(*false*)  
 *console*.error(error)  
})

**app.js**

*const* express = require('express')  
*const* mongoose = require('mongoose')  
*const* bodyParser = require('body-parser')  
*const* helmet = require('helmet')  
*const* path = require('path')  
*const cors* = require('cors')  
*const compression* = require('compression')  
*const* HttpErrors = require('./config/errors.config')  
*const* UsersRoutes = require('./routes/users.routes')  
*const* AuthRoutes = require('./routes/auth.routes')  
*const* UploadsRoutes = require('./routes/uploads.routes')  
*const* QuizzesRoutes = require('./routes/quizzes.routes')  
  
require('dotenv').*config*()  
  
*const* app = express()  
  
app.use(bodyParser.urlencoded({  
 extended: *true*}))  
  
app.use(bodyParser.json())  
app.use(*compression*())  
app.use(*cors*())  
app.use(helmet())  
  
app.use('/public', express.static(path.join(\_\_dirname, 'public')))  
  
app.use('/users', UsersRoutes)  
app.use('/auth', AuthRoutes)  
app.use('/uploads', UploadsRoutes)  
app.use('/quizzes', QuizzesRoutes)  
  
app.get('\*', (req, res) => {  
 res.status(200).send('Server is running');  
})  
  
app.use(() => {  
 *throw new* HttpErrors('Could not find this route.', 404)  
})

**Mongo query for a specific Mongo collection (implemented code)**

**app.js**

*const* uri = process.env.ATLAS\_URI  
*const* port = process.env.PORT  
*const* dbName = process.env.DATABASE  
  
*const* options = {  
 useNewUrlParser: *true*,  
 useUnifiedTopology: *true*,  
 useCreateIndex: *true*,  
 dbName: dbName  
}  
  
mongoose.*connect*(uri, options).then(() => {  
 app.listen(port)  
 *console*.log(`Server is running on port: ${port}`)  
}).catch((error) => {  
 *console*.error(error)  
})

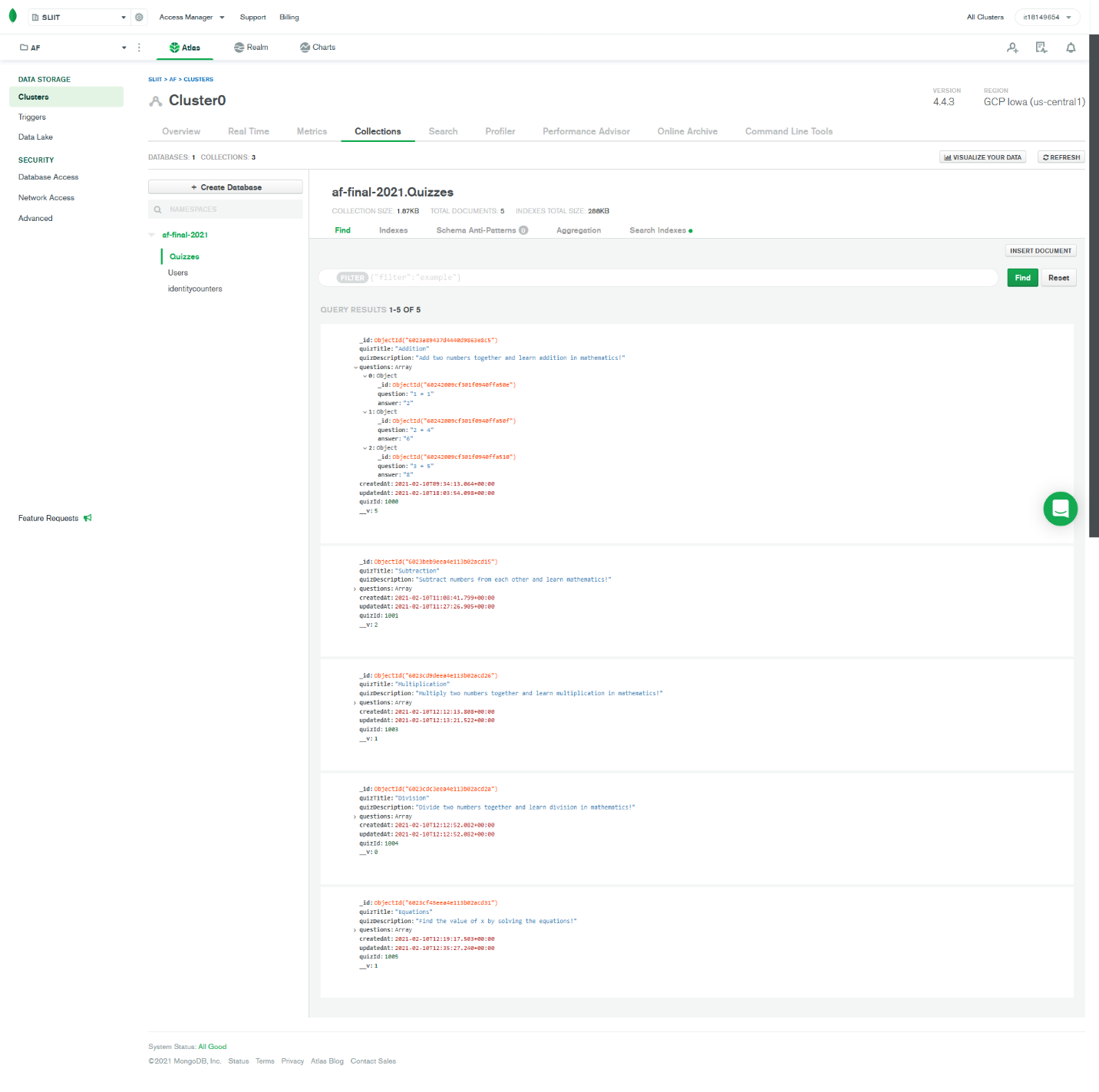
**quizzes.model.js**

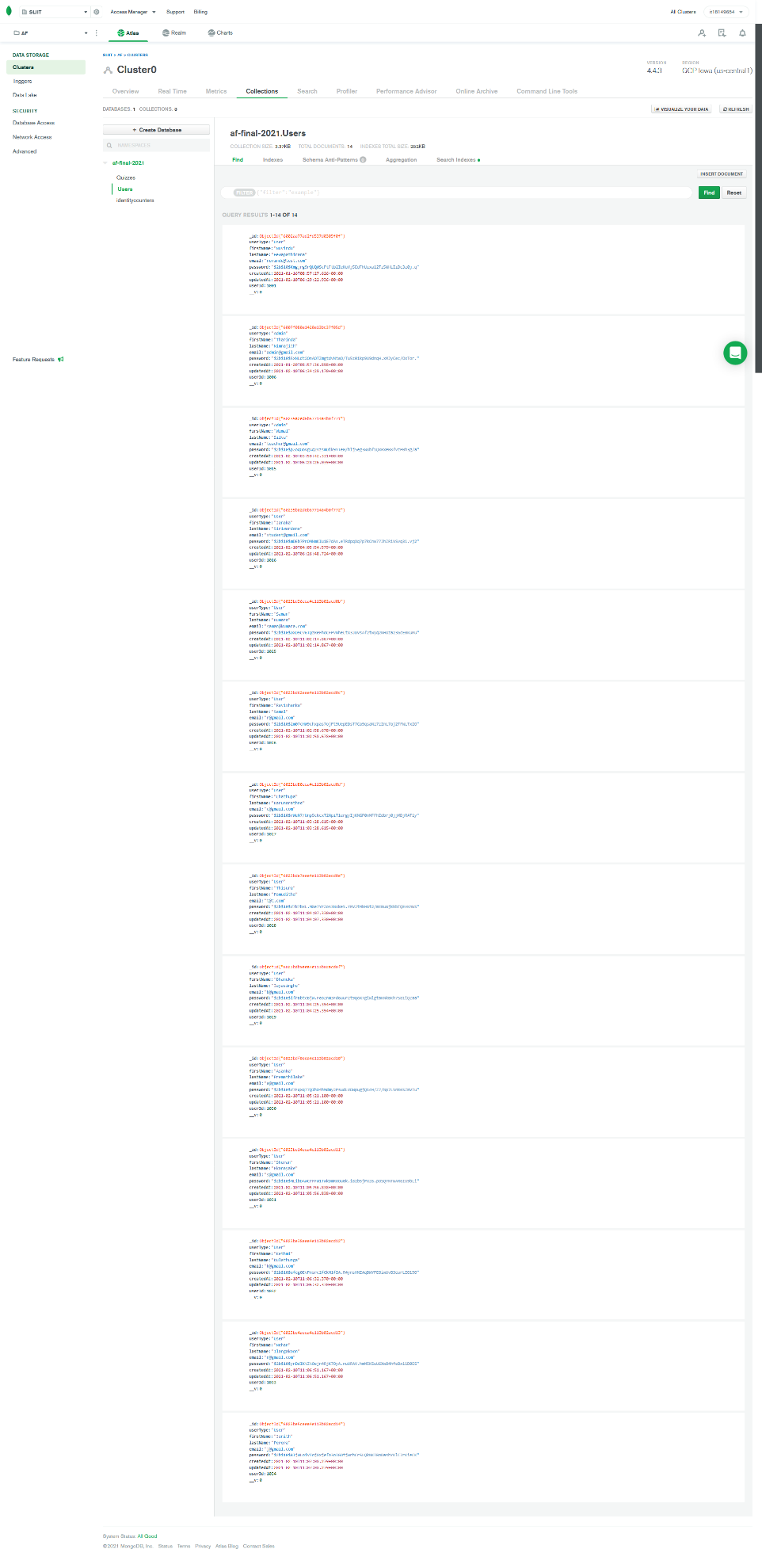
*const* mongoose = require('mongoose')  
*const* autoIncrement = require('mongoose-auto-increment')  
*const uniqueValidator* = require('mongoose-unique-validator')  
  
*const* Schema = mongoose.Schema  
  
*const* QuizzesSchema = *new* Schema({  
 quizId: {  
 type: Number,  
 required: *false*,  
 unique: *true*,  
 trim: *true* },  
 quizTitle: {  
 type: String,  
 required: *true*,  
 unique: *true*,  
 trim: *true* },  
 quizDescription: {  
 type: String,  
 required: *true*,  
 unique: *false*,  
 trim: *true* },  
 questions: [{  
 question: {  
 type: String,  
 required: *true*,  
 unique: *false*,  
 trim: *true* },  
 answer: {  
 type: String,  
 required: *true*,  
 unique: *false*,  
 trim: *true* }  
 }]  
}, {  
 timestamps: *true*,  
 collection: 'Quizzes'  
})  
  
QuizzesSchema.plugin(*uniqueValidator*)  
  
autoIncrement.initialize(mongoose.connection)  
  
QuizzesSchema.plugin(autoIncrement.plugin, {  
 model: 'Quizzes',  
 field: 'quizId',  
 startAt: 1000,  
 incrementBy: 1  
})  
  
module.exports = mongoose.*model*('Quizzes', QuizzesSchema)

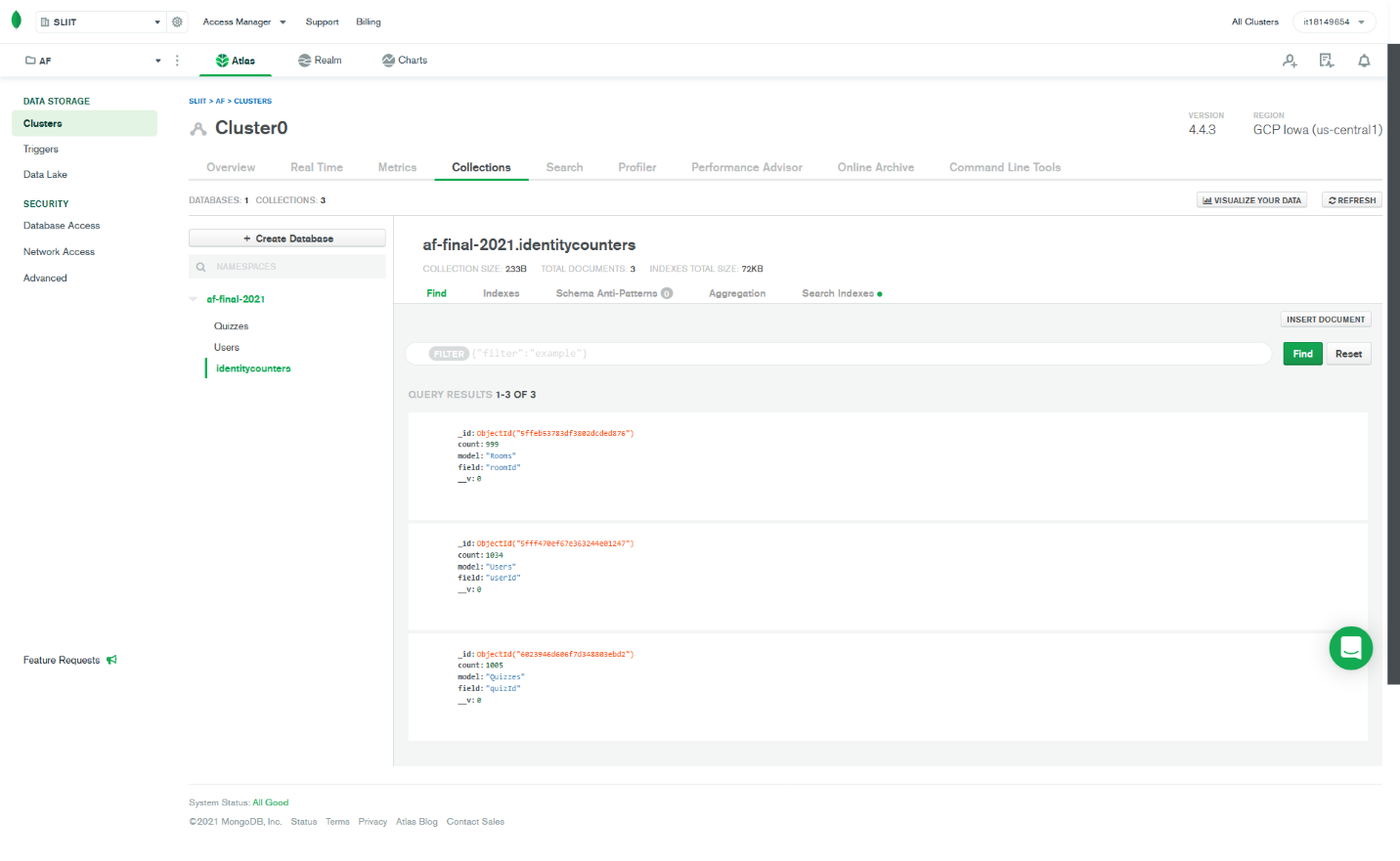
**users.model.js**

*const* mongoose = require('mongoose')  
*const* autoIncrement = require('mongoose-auto-increment')  
*const uniqueValidator* = require('mongoose-unique-validator')  
  
*const* Schema = mongoose.Schema  
  
*const* userTypes = [  
 'Admin',  
 'User'  
]  
  
*const* UsersSchema = *new* Schema({  
 userId: {  
 type: Number,  
 required: *false*,  
 unique: *true*,  
 trim: *true* },  
 firstName: {  
 type: String,  
 required: *false*,  
 unique: *false*,  
 trim: *true* },  
 lastName: {  
 type: String,  
 required: *false*,  
 unique: *false*,  
 trim: *true* },  
 email: {  
 type: String,  
 required: *true*,  
 unique: *true*,  
 trim: *true* },  
 password: {  
 type: String,  
 required: *true*,  
 unique: *false*,  
 trim: *true* },  
 userType: {  
 type: String,  
 *enum*: userTypes,  
 required: *false*,  
 unique: *false*,  
 trim: *true*,  
 *default*: 'User'  
 }  
}, {  
 timestamps: *true*,  
 collection: 'Users'  
})  
  
UsersSchema.plugin(*uniqueValidator*)  
  
autoIncrement.initialize(mongoose.connection)  
  
UsersSchema.plugin(autoIncrement.plugin, {  
 model: 'Users',  
 field: 'userId',  
 startAt: 1000,  
 incrementBy: 1  
})  
  
module.exports = mongoose.*model*('Users', UsersSchema)

**MongoDB Atlas**







**Mongo queries in quizzes controller**

*try* {  
 quizList = *await* QuizModel.find()  
} *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
}

*try* {  
 quiz = *await* QuizModel.findById(id)  
} *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
}

*try* {  
 quiz = *await* QuizModel.findById(id)  
 *await* quiz.remove()  
} *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
}

*try* {  
 existingQuiz = *await* QuizModel.findOne({  
 quizTitle: quizTitle  
 })  
} *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
}

*try* {  
 *await* quiz.save()  
} *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
}

**Mongo queries in users and auth controllers**

*try* {  
 user = *await* UserModel.findOne({  
 email: email  
 })  
} *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
}

*try* {  
 *await* newUser.save()  
} *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
}

*try* {  
 user = *await* UserModel.findById(id)  
} *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
}

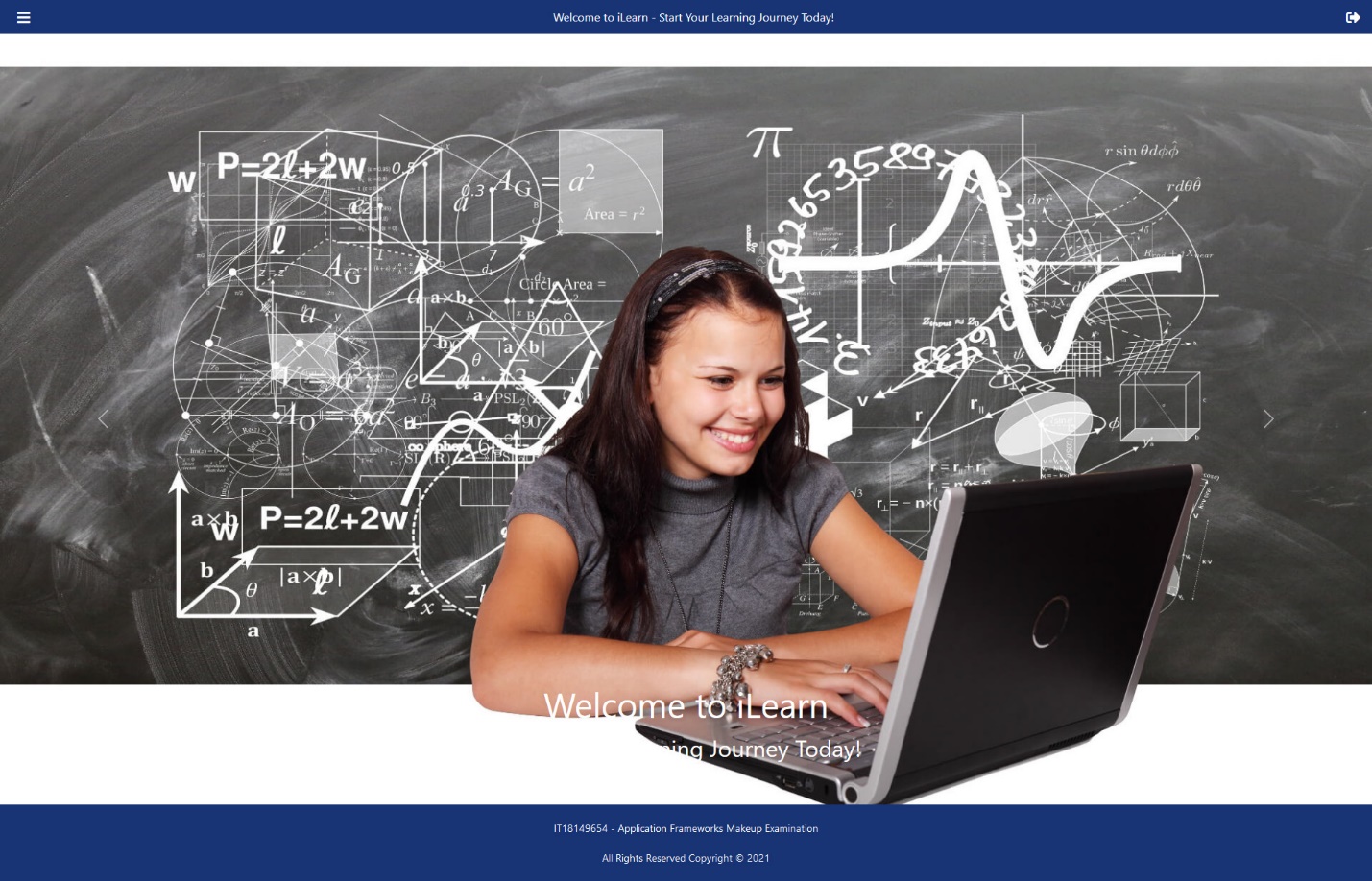
*try* {  
 quiz = *await* QuizModel.findById(id)  
 *await* quiz.remove()  
} *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
}

*try* {  
 quiz = *await* QuizModel.findById(id)  
} *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
}

*try* {  
 quizList = *await* QuizModel.find()  
} *catch* (error) {  
 *console*.error(error)  
 res.status(500).send(error)  
}

**Unit test using JEST to evaluate a core logic or a service (implemented code)**

**Screenshot of the home page of the running application on localhost**



**Thank You!**