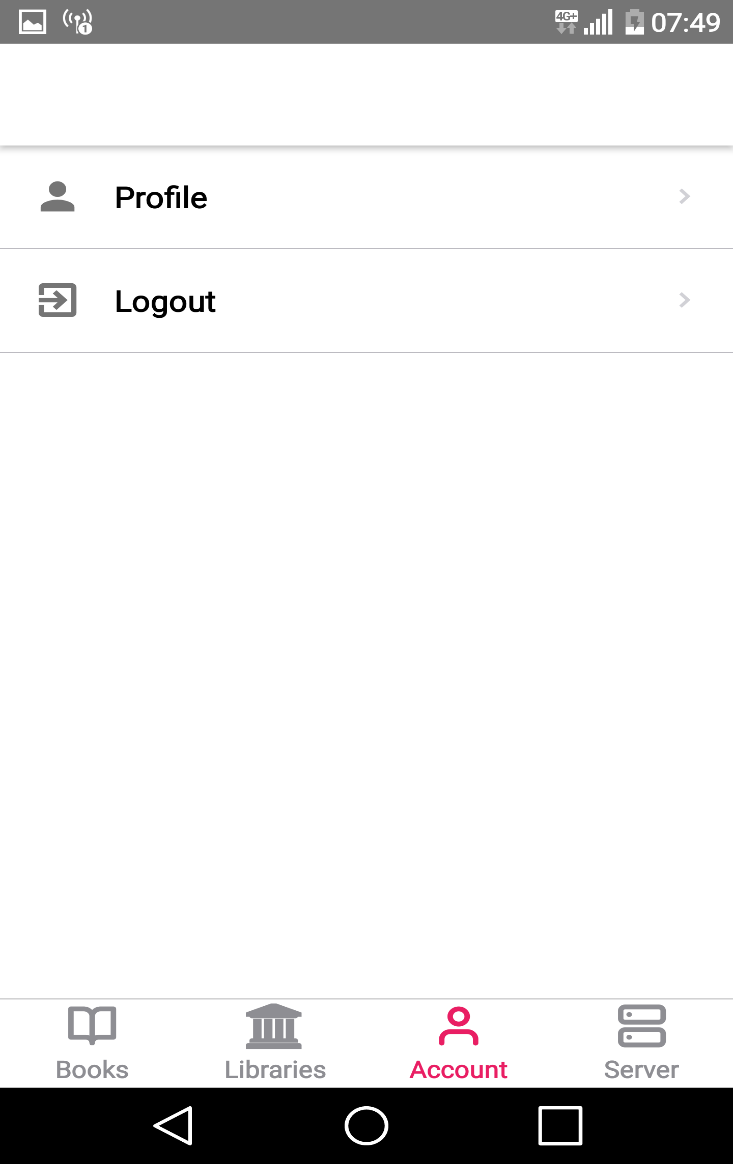
**INTRODUCTION**

In this phase, which is the final phase of our project we will be looking at the coding part of the system which will show the coding we did in order for the system to function. We will specify what coding we did for every feature that is in our application. The next step will be to do the Testing, where we will explain potential testing tools. Furthermore, we will touch on the System testing which has the Test Case and Evaluation of the testing results. The final step here will be the installation, where we will specify which software application we used for our system, we will reveal where our application will be installed; whether it is a mobile application or a desktop application or both.

**CODING**

**Mobile App**

**Account Screen**

****

import {Text, View} from "react-native";

import React from "react";

import {ListItem} from "react-native-elements";

import {inject, observer} from "mobx-react";

import authStore from '../../stores/AuthStore'

import { NavigationStackProp } from "react-navigation-stack";

interface IProps {

navigation: NavigationStackProp,

// @ts-ignore

authStore: AuthStore

}

class AccountScreen extends React.Component<IProps> {

constructor(props: IProps){

super(props)

}

handlePress = (title: string) => {

switch (title) {

case 'Profile':

this.props.navigation.navigate('Profile');

break;

case 'Logout':

this.props.authStore.logout();

this.props.navigation.navigate('Login');

break;

}

};

render() {

const list = [

{

title: 'Profile',

icon: 'person'

},

/\*{

title: 'Change password',

icon: 'flight-takeoff'

},

{

title: 'Delete my account',

icon: 'remove'

},\*/

{

title: 'Logout',

icon: 'exit-to-app'

}

];

return (

<View>

{

list.map((item, i) => (

<ListItem

key={i}

title={item.title}

leftIcon={{ name: item.icon }}

bottomDivider

chevron

onPress={(e) => {

this.handlePress(item.title)

}}

/>

))

}

</View>

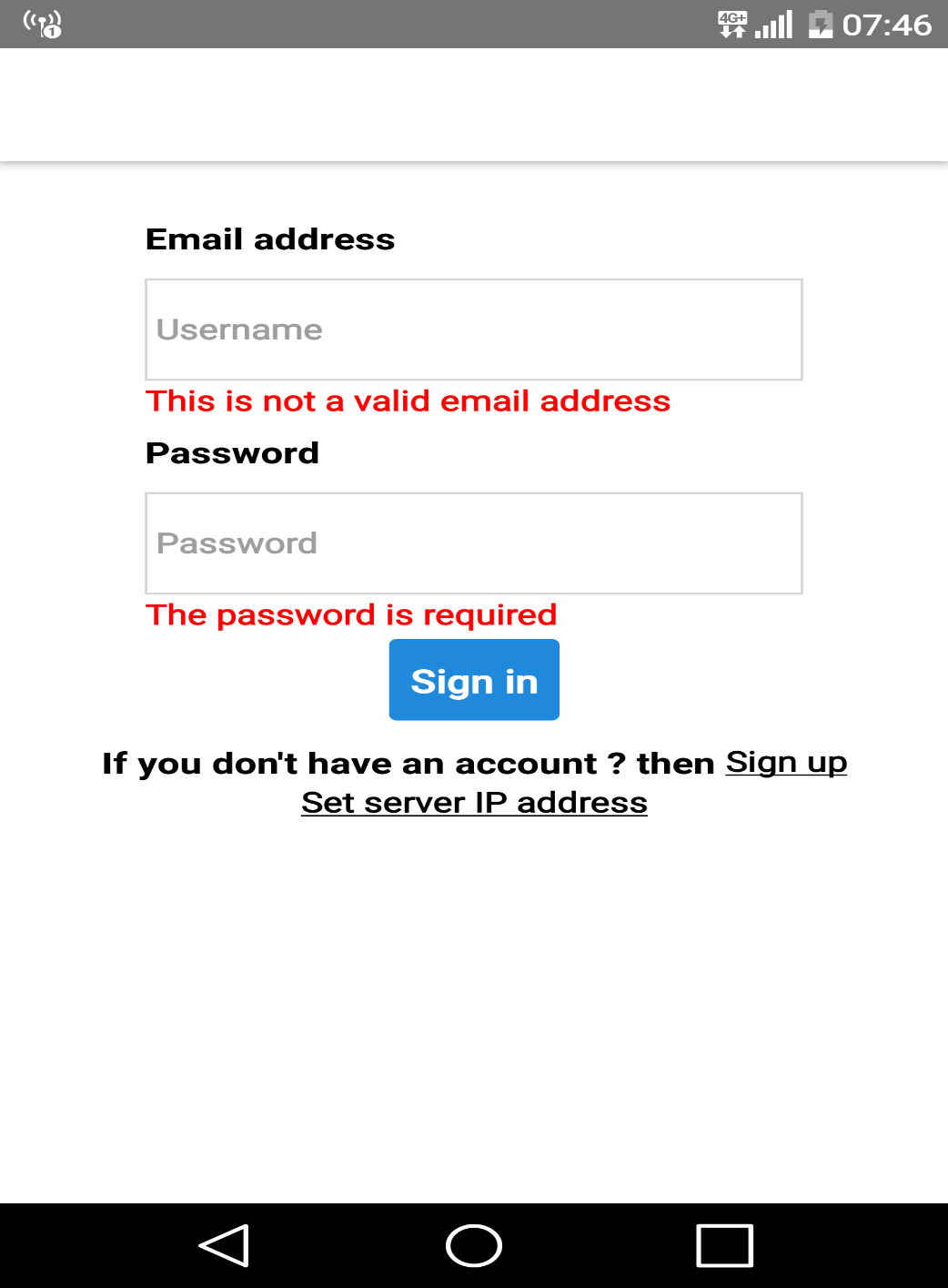
)

}

}

export default inject('authStore')(observer(AccountScreen));

**Login screen**

****

import \* as React from "react";

import { View, Text, TextInput, TouchableOpacity, StyleSheet, ActivityIndicator } from "react-native";

import { inject, observer } from "mobx-react";

import { Button, Divider } from "react-native-elements";

import { PrimaryColor } from "../../AppConstant";

import authStore from "../../stores/AuthStore";

import { NavigationStackProp } from "react-navigation-stack";

import { NavigationActions } from "react-navigation";

interface IProps {

navigation: NavigationStackProp,

// @ts-ignore

authStore: AuthStore

}

class LoginScreen extends React.Component<IProps> {

constructor(props: IProps) {

super(props);

}

componentWillUnmount() {

this.props.authStore.setPassword('');

this.props.authStore.setEmail('');

}

signIn = () => {

this.props.authStore.login();

};

render() {

const { errors, isAuthenticated } = this.props.authStore;

if (isAuthenticated)

this.props.navigation.navigate('Home');

return (

<View style={styles.mainView}>

<View style={styles.rowView}>

<Text style={styles.label}>Email address</Text>

<TextInput

style={styles.textField}

placeholder='Username'

onChangeText={value => {

this.props.authStore.setEmail(value)

}}

/>

{errors.email && errors.email.length > 0 ?

<Text style={styles.error}>

{this.props.authStore.errors.email[0]}

</Text> : null}

</View>

<View style={styles.rowView}>

<Text style={styles.label}>Password</Text>

<TextInput

style={styles.textField}

secureTextEntry={true}

placeholder='Password'

onChangeText={value => {

this.props.authStore.setPassword(value);

}}

/>

{errors.password && errors.password.length > 0 ?

<Text style={styles.error}>

{this.props.authStore.errors.password[0]}

</Text> : null}

</View>

<View style={{ flexDirection: 'row-reverse' }}>

{authStore.isLoading ?

<ActivityIndicator size='large' color={PrimaryColor} />

: null}

<Button

style={{ marginTop: 10 }}

onPress={this.signIn}

title='Sign in' />

</View>

<Divider />

<View style={{ flexDirection: 'row', marginTop: 10 }}>

<Text style={styles.label}>If you don't have an account ? then </Text>

<TouchableOpacity

onPress={() => this.props.navigation.navigate('SignUp')}

>

<Text style={{ textDecorationLine: 'underline' }}>Sign up</Text>

</TouchableOpacity>

</View>

<TouchableOpacity

onPress={() => {

this.props.navigation.push("SConfig", {});

console.log("show server");

}}

>

<Text style={{ textDecorationLine: 'underline' }}>Set server IP address</Text>

</TouchableOpacity>

</View>

)

}

}

export default inject('authStore')(observer(LoginScreen));

const styles = StyleSheet.create({

mainView: {

flex: 1,

alignItems: 'center',

marginTop: 25

},

textField: {

width: 250,

borderColor: '#d6d7da',

borderWidth: 1,

marginTop: 10

},

rowView: {

padding: 3

},

label: {

fontWeight: 'bold',

fontSize: 15

},

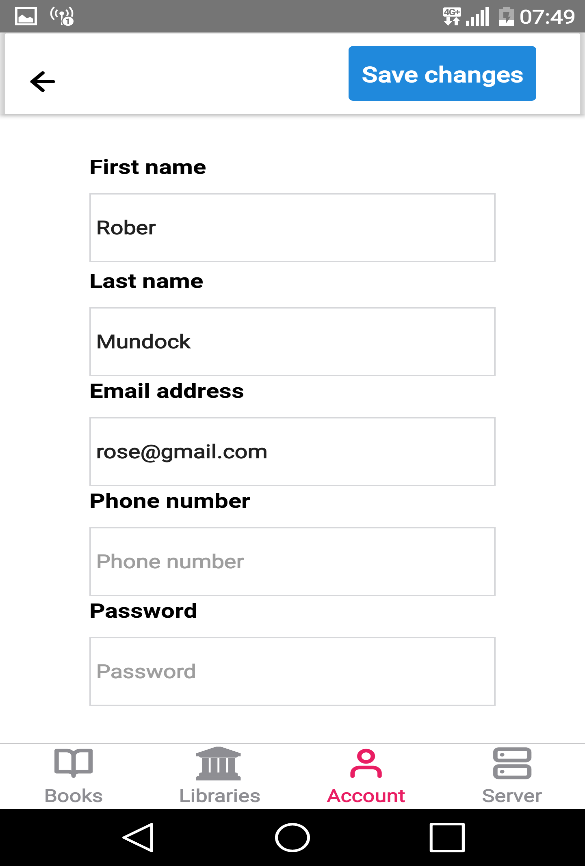
error: {

color: 'red'

}

});

**Profile Screen**

****

import { View, StyleSheet, Text, TextInput, ActivityIndicator, ScrollView, TouchableOpacity } from "react-native";

import \* as React from "react";

import userStore from "../../stores/UserStore";

import { PrimaryColor } from "../../AppConstant";

import { Button } from "react-native-elements";

import Icon from 'react-native-vector-icons/Feather'

import { inject, observer } from "mobx-react";

interface IProps {

navigation: any,

// @ts-ignore

userStore: UserStore

}

class ProfileScreen extends React.Component<IProps> {

componentDidMount() {

this.props.userStore.initializeForm()

}

componentWillUnmount() {

this.props.userStore.initializeForm();

}

update = () => {

this.props.userStore.update();

};

render() {

const { errors, isLoading, updateSuccess } = this.props.userStore;

const store = this.props.userStore;

return (

<>

<View style={styles.topBar}>

<TouchableOpacity

onPress={() => {

this.props.navigation.goBack();

}}

>

<Icon name="arrow-left" size={22} style={styles.icon} />

</TouchableOpacity>

<View style={{

marginLeft: 20,

flexDirection: 'row-reverse',

flex: 1

}}>

<Button

onPress={this.update}

title='Save changes' />

{store.isLoading ?

<ActivityIndicator size='large' color={PrimaryColor} />

: null}

</View>

</View>

<ScrollView>

<View style={styles.mainView}>

{updateSuccess ? <Text>The update was successful.</Text> : null}

<View style={styles.rowView}>

<Text style={styles.label}>First name</Text>

<TextInput

style={styles.textField}

placeholder='First name'

value={store.form.firstName}

onChangeText={value => {

this.props.userStore.setFirstName(value)

}}

/>

{errors.firstName && errors.firstName.length > 0 ?

<Text style={styles.error}>

{errors.firstName[0]}

</Text> : null}

</View>

<View>

<Text style={styles.label}>Last name</Text>

<TextInput

style={styles.textField}

placeholder='Last name'

value={store.form.lastName}

onChangeText={value => {

this.props.userStore.setLastName(value)

}}

/>

{errors.lastName && errors.lastName.length > 0 ?

<Text style={styles.error}>

{errors.lastName[0]}

</Text> : null}

</View>

<View>

<Text style={styles.label}>Email address</Text>

<TextInput

style={styles.textField}

placeholder='Email address'

value={store.form.email}

onChangeText={value => {

this.props.userStore.setEmail(value)

}}

/>

{errors.email && errors.email.length > 0 ?

<Text style={styles.error}>

{errors.email[0]}

</Text> : null}

</View>

<View>

<Text style={styles.label}>Phone number</Text>

<TextInput

style={styles.textField}

placeholder='Phone number'

value={store.form.phone}

onChangeText={value => {

this.props.userStore.setPhone(value)

}}

/>

{errors.phone && errors.phone.length > 0 ?

<Text style={styles.error}>

{errors.phone[0]}

</Text> : null}

</View>

<View>

<Text style={styles.label}>Password</Text>

<TextInput

style={styles.textField}

placeholder='Password'

secureTextEntry={true}

value={store.form.Password}

onChangeText={value => {

this.props.userStore.setPassword(value)

}}

/>

{errors.phone && errors.password.length > 0 ?

<Text style={styles.error}>

{errors.password[0]}

</Text> : null}

</View>

</View>

</ScrollView>

</>

)

}

}

const styles = StyleSheet.create({

topBar: {

flexDirection: 'row',

borderRadius: 1,

elevation: 3,

padding: 10

},

searchBox: {

},

icon: {

marginTop: 15,

marginLeft: 5

},

mainView: {

flex: 1,

alignItems: 'center',

marginTop: 25

},

textField: {

width: 250,

borderColor: '#d6d7da',

borderWidth: 1,

marginTop: 10

},

rowView: {

padding: 3

},

label: {

fontWeight: 'bold',

fontSize: 15

},

error: {

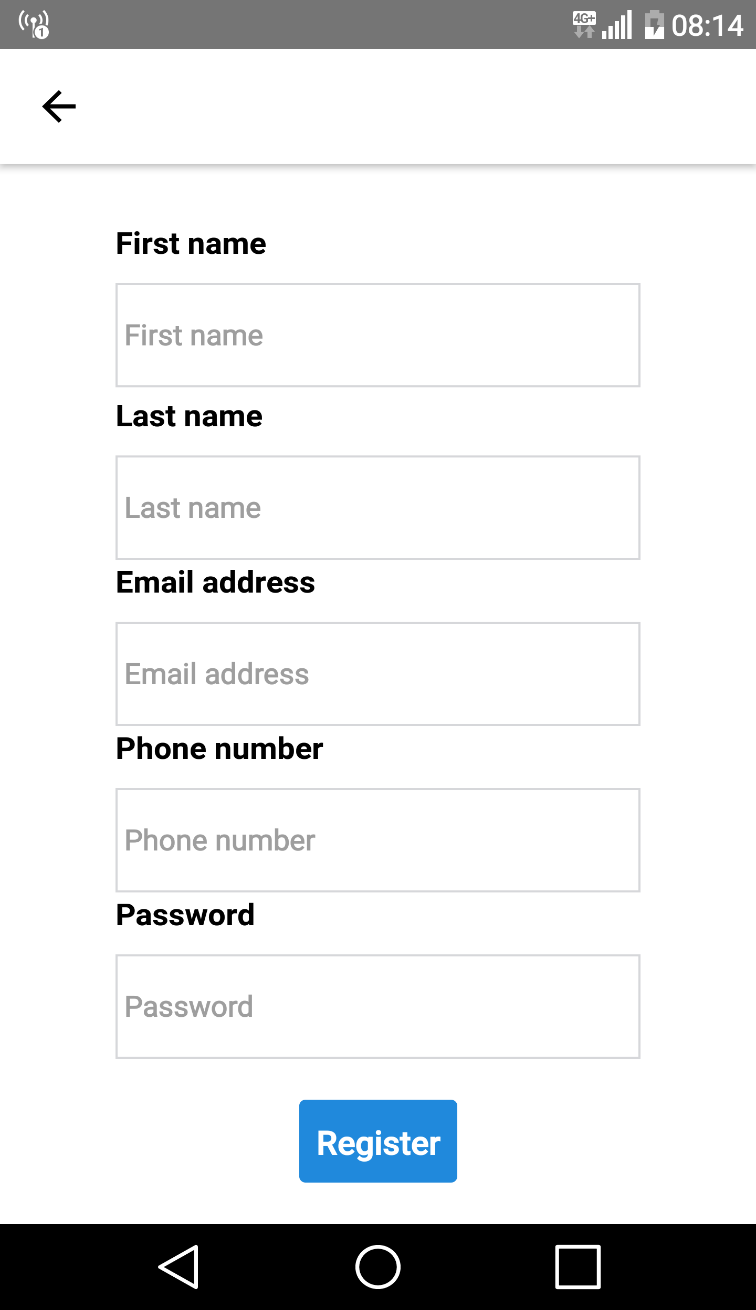
color: 'red'

}

});

export default inject('userStore')(observer(ProfileScreen))

**Registration Screen**

****

import \* as React from "react";

import {TextInput, Text, View, StyleSheet, ScrollView, ActivityIndicator} from "react-native";

import {inject, observer} from "mobx-react";

import UserStore from "../../stores/UserStore";

import {Button} from "react-native-elements";

import {UserViewModel} from "../../models/UserModels";

import {PrimaryColor} from "../../AppConstant";

interface IProps {

// @ts-ignore

userStore: UserStore,

navigation: any

}

class RegistrationScreen extends React.Component<IProps> {

constructor(props: IProps) {

super(props);

}

register = () => {

this.props.userStore.register()

};

render() {

const {errors, operationSucceeded } = this.props.userStore;

const store = this.props.userStore;

if (operationSucceeded){

// Go home page

this.props.navigation.navigate('Home');

}

return (

<ScrollView>

<View style={styles.mainView}>

<View style={styles.rowView}>

<Text style={styles.label}>First name</Text>

<TextInput

style={styles.textField}

placeholder='First name'

value={store.form.firstName}

onChangeText={value => {

this.props.userStore.setFirstName(value)

}}

/>

{errors.firstName && errors.firstName.length > 0 ?

<Text style={styles.error}>

{errors.firstName[0]}

</Text> : null}

</View>

<View>

<Text style={styles.label}>Last name</Text>

<TextInput

style={styles.textField}

placeholder='Last name'

value={store.form.lastName}

onChangeText={value => {

this.props.userStore.setLastName(value)

}}

/>

{errors.lastName && errors.lastName.length > 0 ?

<Text style={styles.error}>

{errors.lastName[0]}

</Text> : null}

</View>

<View>

<Text style={styles.label}>Email address</Text>

<TextInput

style={styles.textField}

placeholder='Email address'

value={store.form.email}

onChangeText={value => {

this.props.userStore.setEmail(value)

}}

/>

{errors.email && errors.email.length > 0 ?

<Text style={styles.error}>

{errors.email[0]}

</Text> : null}

</View>

<View>

<Text style={styles.label}>Phone number</Text>

<TextInput

style={styles.textField}

placeholder='Phone number'

value={store.form.phone}

onChangeText={value => {

this.props.userStore.setPhone(value)

}}

/>

{errors.phone && errors.phone.length > 0 ?

<Text style={styles.error}>

{errors.phone[0]}

</Text> : null}

</View>

<View>

<Text style={styles.label}>Password</Text>

<TextInput

style={styles.textField}

placeholder='Password'

secureTextEntry={true}

value={store.form.password}

onChangeText={value => {

this.props.userStore.setPassword(value)

}}

/>

{errors.password && errors.password.length > 0 ?

<Text style={styles.error}>

{errors.password[0]}

</Text> : null}

</View>

<View style={{flexDirection: 'row-reverse', marginTop: 20}}>

{store.isLoading ?

<ActivityIndicator size='large' color={PrimaryColor}/>

: null}

<Button

onPress={this.register}

title='Register'/>

</View>

</View>

</ScrollView>

)

}

}

export default inject('userStore')(observer(RegistrationScreen))

const styles = StyleSheet.create({

mainView: {

flex: 1,

alignItems: 'center',

marginTop: 25

},

textField: {

width: 250,

borderColor: '#d6d7da',

borderWidth: 1,

marginTop: 10

},

rowView: {

padding: 3

},

label: {

fontWeight: 'bold',

fontSize: 15

},

error: {

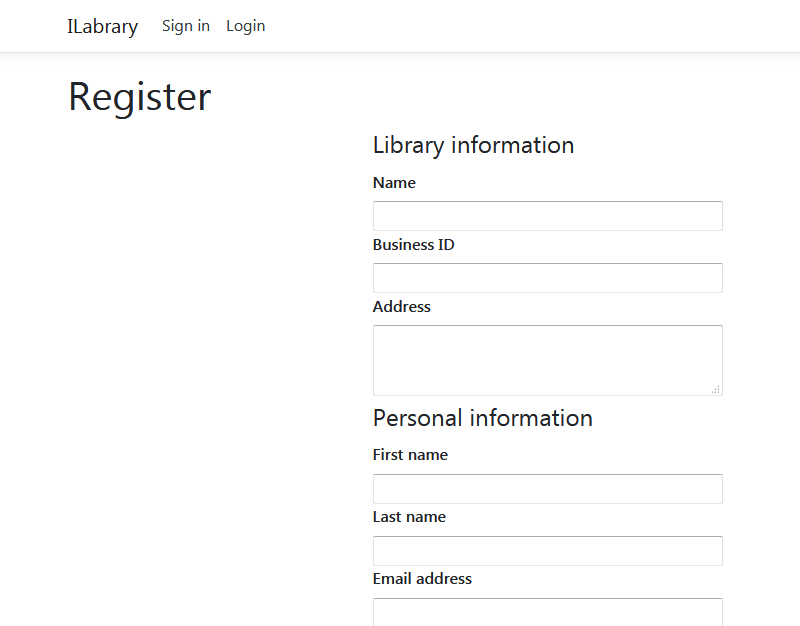
color: 'red'

}

});

**Website**

**Registration**

****

/// <summary>

///

/// </summary>

/// <returns></returns>

public IActionResult Register()

{

return View();

}

/// <summary>

/// Create a library

/// </summary>

/// <param name="viewModel"></param>

/// <returns></returns>

[HttpPost]

public IActionResult Register(LibraryCreateViewModel viewModel)

{

if (!ModelState.IsValid)

{

return View(viewModel);

}

var user = \_mapper.Map<LibraryUser>(viewModel);

var library = \_mapper.Map<Library>(viewModel);

user.Library = library;

// Create the session

var session = new LibraryUserSession

{

User = user,

Token = Guid.NewGuid().ToString()

};

\_db.Add(session);

\_db.SaveChanges();

Response.Cookies.Append(SesssionKeys.LibrarySessionKey, session.Token, new CookieOptions

{

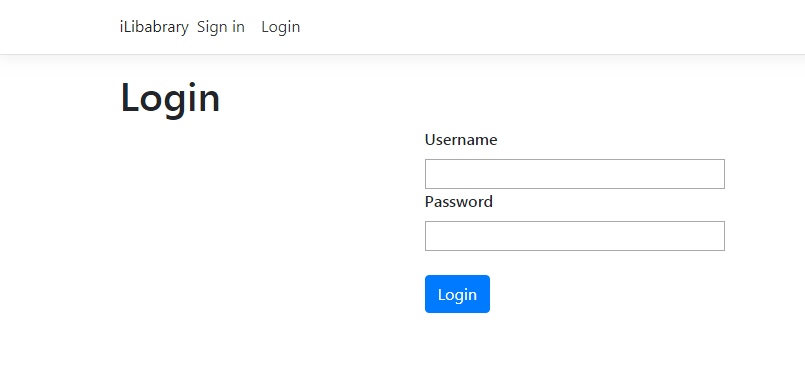
Path = "/"

});

return RedirectToAction("Index", "Home");

}

**Login page**

****

/// <summary>

///

/// </summary>

/// <param name="viewModel"></param>

/// <returns></returns>

[HttpPost]

public IActionResult Login(LibraryLoginViewModel viewModel)

{

if (!ModelState.IsValid)

return View(viewModel);

IActionResult InvalidUsername()

{

ModelState.AddModelError(nameof(LibraryLoginViewModel.Username), "Invalid username.");

return View(viewModel);

}

var username = viewModel.Username.Split("@");

if (username.Count() < 2)

{

return InvalidUsername();

}

var library = \_db.Libraries.FirstOrDefault(x => x.UniqueId == username[1]);

if (library == null)

{

return InvalidUsername();

}

// Get the user

var user = \_db.LibraryUsers.FirstOrDefault(x => x.Username == username[0] && x.LibraryId == library.Id && x.Password == viewModel.Password);

if (user == null)

{

return InvalidUsername();

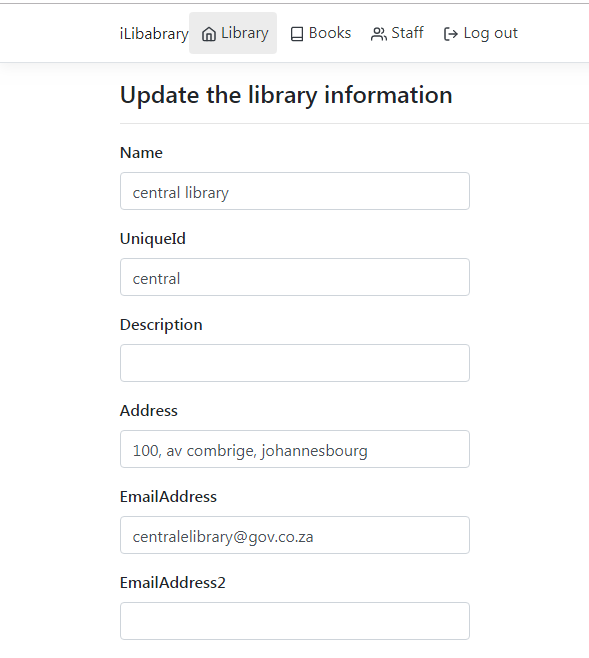
}

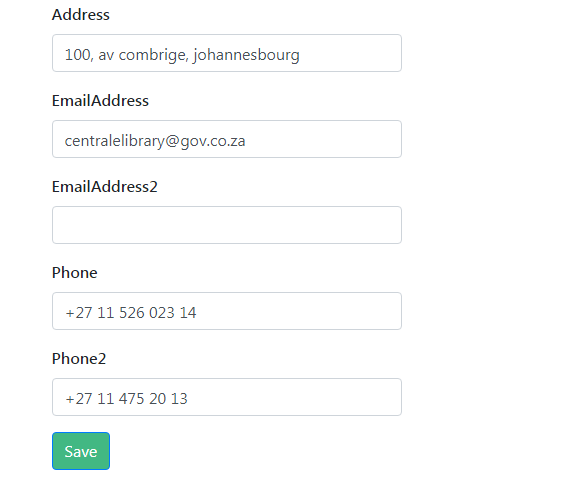
// Create session

return CreateSession(user);

}

**Update library**

****

****

public async Task<IActionResult> Update()

{

if (! await \_sessionService.IsAuthentificateAsync())

{

return RedirectToAction("Login");

}

return View(\_db.Libraries.FirstOrDefault(x => x.Id == \_sessionService.GetCurrentLibraryId()));

}

[HttpPost]

public async Task<IActionResult> Update(Library library)

{

if (!await \_sessionService.IsAuthentificateAsync())

{

return RedirectToAction("Login");

}

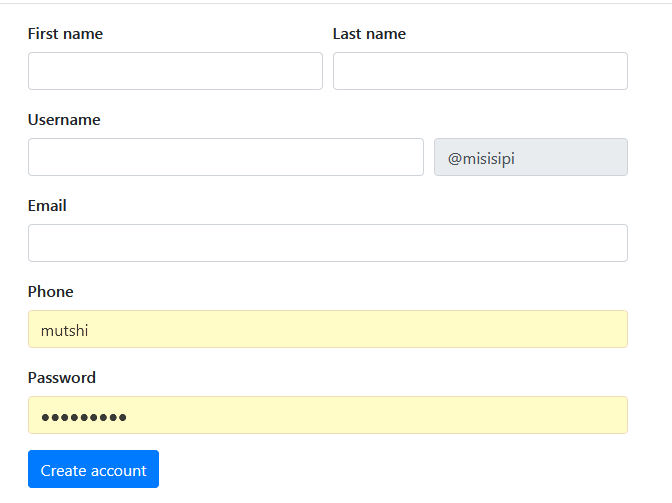
\_db.Update(library);

\_db.SaveChanges();

return RedirectToAction("Index", "Book");

}

**Library add new user**

****

/// <summary>

///

/// </summary>

/// <returns></returns>

public async Task<IActionResult> Create()

{

if (!await \_sessionService.IsAuthentificateAsync())

{

return RedirectToAction("Login", "Library");

}

return View();

}

/// <summary>

///

/// </summary>

/// <returns></returns>

[HttpPost]

public async Task<IActionResult> Create(LibraryUserCreateViewModel viewModel)

{

if (!await \_sessionService.IsAuthentificateAsync())

{

return RedirectToAction("Login", "Library");

}

if (!ModelState.IsValid)

{

return View(viewModel);

}

int libraryId = \_sessionService.GetCurrentLibraryId();

bool usernameUsed = \_db.LibraryUsers

.Any(x => x.Username == viewModel.Username && x.LibraryId == libraryId);

if (usernameUsed)

{

ModelState.AddModelError(nameof(LibraryCreateViewModel.Username), "This username is taken.");

return View(viewModel);

}

var user = \_mapper.Map<LibraryUser>(viewModel);

user.LibraryId = libraryId;

// Create the user

\_db.LibraryUsers.Add(user);

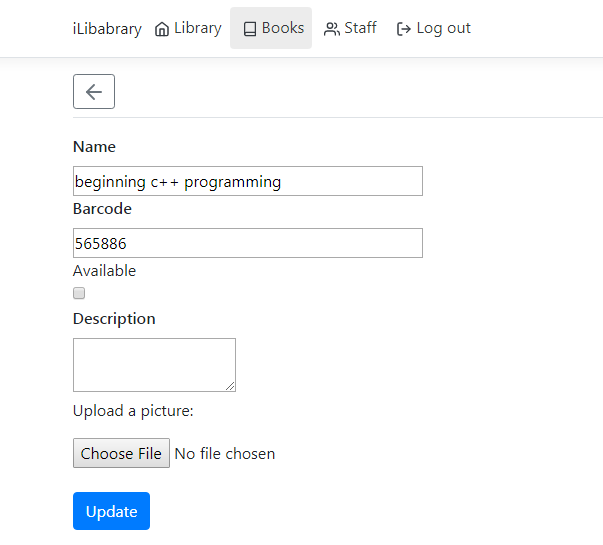
\_db.SaveChanges();

TempData["account\_created"] = true;

return RedirectToAction("Create");

}

**Add new book**

****

[HttpPost("create")]

public async Task<IActionResult> Create(BookCreateViewModel viewModel)

{

if (!await \_sessionService.IsAuthentificateAsync())

{

return RedirectToAction("Login", "Library");

}

if (!ModelState.IsValid)

{

return View(viewModel);

}

// verify the barcode

if (!string.IsNullOrWhiteSpace(viewModel.Barcode))

{

var barcodeExist = \_db.Books.Any(x => x.Barcode.ToLower() == viewModel.Barcode && x.LibraryId == \_sessionService.GetCurrentLibraryId());

if (barcodeExist)

{

ModelState.AddModelError(nameof(BookCreateViewModel.Barcode), "This barcode is already assigned to another book");

return View(viewModel);

}

}

var book = \_mapper.Map<Book>(viewModel);

// Add tags

if (!string.IsNullOrWhiteSpace(viewModel.BookTags))

{

book.Tags = viewModel.BookTags.Split(";").Select(x => new BookTag

{

Name = x

}).ToList();

}

// Add images

if (viewModel.Files != null && viewModel.Files.Any())

{

book.Images = viewModel.Files.Select(file =>

{

string savePath = \_fileService.SaveFile(file);

return new BookPicture

{

FullPath = savePath,

Size = file.Length

};

}).ToList();

}

book.LibraryId = \_sessionService.GetCurrentLibraryId();

\_db.Add(book);

await \_db.SaveChangesAsync();

return RedirectToAction("Index");

}

