**FONTEND:**

1. D:\-chemistry\src\services\user.service.ts

//1 (user)

import { IBaseSearch, ResponseProps } from '@/types/base';

import { IUser } from '@/types/user';

import axios from 'axios';

const prefix = '/api/user';

export const createUser = async (request: IUser): Promise<ResponseProps> => {

  const response = await axios.post(`${prefix}/create`, request);

  return response.data;

};

export const updateUser = async (request: IUser): Promise<ResponseProps> => {

  const response = await axios.post(`${prefix}/update`, request);

  return response.data;

};

export const searchUser = async (

  request: IBaseSearch

): Promise<ResponseProps<IUser[]>> => {

  const response = await axios?.post(`${prefix}/search`, request);

  return response.data;

};

export const deleteUser = async (data: {

  UserId: string;

}): Promise<ResponseProps> => {

  const response = await axios?.post(`${prefix}/delete`, data);

  return response.data;

};

export const login = async (data: { UserName: string, PassWork: string }): Promise<any> => {

  const response = await axios?.post(`${prefix}/login`, data);

  return response.data;

};

**BACKEND**

1. D:\-chemistry\src\helpers\repositories\person.repository.ts

//1

import { IPerson } from './../../types/person.d';

import { db\_Provider } from '@/app/api/Api\_Provider';

import { IBaseSearch } from '@/types/base';

export const createPerson = async (model: IPerson): Promise<any> => {

  try {

    const sql = 'CALL AddPerson(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)';

    return await db\_Provider(

      sql,

      [

        model.id,

        model.user\_id,

        model.name,

        model.status,

        model.birth\_hour,

        model.birth\_minute,

        model.birth\_day,

        model.birth\_month,

        model.birth\_year,

        model.death\_hour,

        model.death\_minute,

        model.death\_day,

        model.death\_month,

        model.death\_year,

        model.gender,

        model.description,

        model.generation,

        model.photo,

        model.parent\_id,

        model.created\_user,

      ],

      true

    );

  } catch (error: any) {

    throw new Error(error.message);

  }

};

export const updatePerson = async (model: IPerson): Promise<any> => {

  try {

    const sql = 'CALL UpdatePerson(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)';

    return await db\_Provider(

      sql,

      [

        model.id,

        model.name,

        model.status,

        model.birth\_hour,

        model.birth\_minute,

        model.birth\_day,

        model.birth\_month,

        model.birth\_year,

        model.death\_hour,

        model.death\_minute,

        model.death\_day,

        model.death\_month,

        model.death\_year,

        model.gender,

        model.description,

        model.generation,

        model.photo,

        model.parent\_id,

        model.lu\_user,

      ],

      true

    );

  } catch (error: any) {

    throw new Error(error.message);

  }

};

// Xóa người dùng (mềm hoặc cứng tùy thủ tục)

export const DeletePerson = async (userId: string): Promise<any> => {

  try {

    const sql = 'CALL DeletePerson(?)';

    return await db\_Provider(sql, [userId], true);

  } catch (error: any) {

    throw new Error(error.message);

  }

};

// Tìm kiếm người dùng có phân trang

export const searchPerson = async (model: IBaseSearch): Promise<any> => {

  try {

    const sql = 'CALL GetPersonsByPageOrder(?,?,?,?)'; // cần tạo thủ tục

    const searchContent = model.search\_content || null;

    const results = await db\_Provider(sql, [

      model.page\_index ?? 0,

      model.page\_size ?? 10,

      model.order\_type ?? 'ASC',

      searchContent,

    ]);

    return results;

  } catch (error: any) {

    throw new Error(error.message);

  }

};

// lẤY THÔNg tin người dùng theo ID

export const getPersonById = async (id: string): Promise<any> => {

  try {

    if (!id) {

      throw new Error('ID người dùng không được để trống');

    }

    const sql = 'CALL Get\_Persons\_ByUserId(?)';

    const results = await db\_Provider(sql, [id]);

    console.log('getPersonById results:', results);

    return results;

  } catch (error: any) {

    throw new Error(error.message);

  }

};

2. D:\-chemistry\src\helpers\services\user.service.ts

//2

import { IUser } from '@/types/user';

import { randomUUID } from 'crypto';

import bcrypt from 'bcryptjs';

import {

  authenticate,

  createUser,

  deleteUser,

  searchUsers,

  updateUser,

} from '../repositories/user.repository';

import { IBaseSearch } from '@/types/base';

import { isDisposableEmail } from '@/services/utils.services';

import { decrypt } from '@/libs/access';

const BCRYPT\_ROUNDS = parseInt('10');

export const createUserService = async (model: IUser) => {

  try {

    // Validate input

    if (!model.username?.trim()) throw new Error('Tên đăng nhập không được để trống');

    if (!model.email?.trim()) throw new Error('Email không được để trống');

    if (!model.password?.trim()) throw new Error('Mật khẩu không được để trống');

    const isInvalidEmail = await isDisposableEmail(model.email);

    if (isInvalidEmail) throw new Error('Địa chỉ email không được hỗ trợ');

    // Decrypt & hash password

    const decryptedPassword = decrypt(model.password);

    console.log("mã hóa mật khẩu", decryptedPassword);

    if (!decryptedPassword) throw new Error('Mật khẩu không hợp lệ');

    const hashedPassword = await bcrypt.hash(decryptedPassword, BCRYPT\_ROUNDS);

    const userId = randomUUID();

    // Save

    const result = await createUser({

      ...model,

      id: userId,

      password: hashedPassword,

    });

    return result;

  } catch (error: any) {

    throw new Error(error.message || 'Lỗi khi tạo người dùng');

  }

};

export const updateUserService = async (model: IUser) => {

  try {

    if (!model.username?.trim()) throw new Error('Tên đăng nhập không được để trống');

    if (!model.email?.trim()) throw new Error('Email không được để trống');

    const isInvalidEmail = await isDisposableEmail(model.email);

    if (isInvalidEmail) throw new Error('Địa chỉ email không được hỗ trợ');

    const updatedModel = { ...model };

    if (model.password?.trim()) {

      const decryptedPassword = decrypt(model.password);

      if (!decryptedPassword) throw new Error('Mật khẩu không hợp lệ');

      const hashedPassword = await bcrypt.hash(decryptedPassword, BCRYPT\_ROUNDS);

      updatedModel.password = hashedPassword;

    }

    const result = await updateUser(updatedModel);

    return result;

  } catch (error: any) {

    throw new Error(error.message || 'Lỗi khi cập nhật người dùng');

  }

};

export const searchUserService = async (model: IBaseSearch) => {

  try {

    return await searchUsers(model);

  } catch (error) {

    throw new Error('Không thể tìm kiếm người dùng');

  }

};

export const deleteUserService = async (userId: string) => {

  try {

    return await deleteUser(userId);

  } catch (error) {

    throw new Error('Không thể xóa người dùng');

  }

};

export const authenticateService = async (

  username: string,

  encryptedPassword: string

) => {

  try {

    const decryptedPassword = decrypt(encryptedPassword);

    const user = await authenticate(username);

    if (!user) return null;

    console.log("user", user)

    console.log("decryptedPassword", decryptedPassword)

    console.log("user.password", user[0].password)

    const isMatch = await bcrypt.compare(decryptedPassword, user[0].password);

    if (!isMatch) return null;

    return { ...user, hashed\_password: undefined };

  } catch (error) {

    return null;

  }

};

3. D:\-chemistry\src\app\api\user\create\route.ts

//3

import { createUserService } from '@/helpers/services/user.service';

import { NextResponse } from 'next/server';

export async function POST(request: Request) {

  try {

    const model = await request.json();

    const result = await createUserService(model);

    return NextResponse.json({

      success: true,

      data: result

    }, { status: 200 });

  } catch (error: any) {

    console.error("API Error in POST /api/user/create:", error);

    return NextResponse.json(

      { success: false, message: error.message || 'Internal Server Error' },

      { status: 500 }

    );

  }

}

sửa xoá tìm kiếm nữa

viết procedure đăng nhập như nào?