// src/app.js

const express = require('express');

const path = require('path');

const morgan = require('morgan');

const bodyParser = require('body-parser');

const connectDB = require('./database/connect');

const session = require('express-session');

const flash = require('connect-flash');

const { ensureAuthenticated } = require('./middleware/auth');

require('./database/seed.js');

const app = express();

// Conectar a la base de datos

connectDB();

// Middleware

app.use(morgan('dev'));

app.use(bodyParser.urlencoded({ extended: false }));

app.use(bodyParser.json());

// Configurar Pug

app.set('views', path.join(\_\_dirname, 'views'));

app.set('view engine', 'pug');

// Archivos estáticos

app.use(express.static(path.join(\_\_dirname, 'public')));

// Configurar sesiones

app.use(session({

  secret: 'tu\_secreto\_de\_sesion', // Cambia esto por un secreto seguro

  resave: false,

  saveUninitialized: false,

}));

// Configurar flash

app.use(flash());

// Middleware para pasar mensajes y usuario a las vistas

app.use((req, res, next) => {

  res.locals.success\_msg = req.flash('success\_msg');

  res.locals.error\_msg = req.flash('error\_msg');

  res.locals.user = req.session.user || null;

  next();

});

// Rutas

const usuarioRoutes = require('./routes/usuarios');

const tareaRoutes = require('./routes/tareas');

const authRoutes = require('./routes/auth');

app.use('/usuarios', usuarioRoutes);

app.use('/tareas', tareaRoutes);

app.use('/auth', authRoutes);

// Ruta principal (Login)

app.get('/', (req, res) => {

  res.render('login', { titulo: 'Iniciar Sesión' });

});

// Ruta del dashboard

app.get('/dashboard', ensureAuthenticated, (req, res) => {

  res.render('dashboard', { titulo: 'Panel de Administración' });

});

// Manejo de errores 404

// app.use((req, res, next) => {

//   res.status(404).send('Página no encontrada');

// });

// Iniciar el servidor

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

  console.log(`Servidor corriendo en el puerto ${PORT}`);

});

// src/controllers/authController.js

const Usuario = require('../models/Usuario');

exports.formLogin = (req, res) => {

  res.render('login', { titulo: 'Iniciar Sesión' });

};

exports.login = async (req, res) => {

  const { email, contraseña } = req.body;

  try {

    const usuario = await Usuario.findOne({ email });

    if (!usuario) {

      req.flash('error\_msg', 'Credenciales inválidas');

      return res.redirect('/auth/login');

    }

    const isMatch = await usuario.matchPassword(contraseña);

    if (!isMatch) {

      req.flash('error\_msg', 'Credenciales inválidas');

      return res.redirect('/auth/login');

    }

    // Guardar usuario en la sesión

    req.session.user = {

      id: usuario.\_id,

      nombre: usuario.nombre,

      email: usuario.email,

      rol: usuario.rol,

    };

    req.flash('success\_msg', 'Inicio de sesión exitoso');

    res.redirect('/dashboard');

  } catch (error) {

    console.error(error);

    req.flash('error\_msg', 'Error del servidor');

    res.redirect('/auth/login');

  }

};

exports.logout = (req, res) => {

    // Guardar la sesión antes de destruirla para permitir que los mensajes flash se almacenen

    req.flash('success\_msg', 'Sesión cerrada exitosamente');

    req.session.save(err => {

      if (err) {

        console.error('Error al guardar la sesión:', err);

        // Intentar destruir la sesión, aunque hubo un error al guardar

        req.session.destroy(() => {

          res.redirect('/auth/login'); // Redirigir a login sin mensaje

        });

      } else {

        req.session.destroy(err => {

          if (err) {

            console.error('Error al destruir la sesión:', err);

            return res.redirect('/dashboard'); // Redirigir a dashboard si hay error

          }

          res.clearCookie('connect.sid'); // Limpiar cookie de sesión

          res.redirect('/auth/login'); // Redirigir a login

        });

      }

    });

  };

// src/controllers/tareaController.js

const Tarea = require('../models/Tarea');

const Estado = require('../models/Estado');

const Prioridad = require('../models/Prioridad');

const Usuario = require('../models/Usuario');

//Obtener todas las tareas

exports.getTareas = async (req, res) => {

  try {

    const tareas = await Tarea.find()

      .populate('estado')

      .populate('prioridad')

      .populate('usuarioAsignado');

    console.log(tareas);

    res.render('tareas/listar', { titulo: 'Lista de Tareas', tareas });

  } catch (error) {

    console.error(error);

    res.send('Error al obtener tareas');

  }

};

// Obtener todas las tareas ordenadas por fecha

let directionSort = -1;

exports.getTareasOrdenadas = async (req, res) => {

  const { sort } = req.query

  const estados = await Estado.find();

  const prioridades = await Prioridad.find();

  const filtros = {

    estado: req.query.estado || '',

    prioridad: req.query.prioridad || '',

    fecha: req.query.fecha || ''

  };

  try {

    const tareas = await Tarea.find()

    .populate('estado')

    .populate('prioridad')

    .populate('usuarioAsignado')

    .sort({[sort]: (directionSort \* -1)});

    directionSort \*= -1;

    res.render('tareas/listar', { titulo: 'Lista de Tareas', tareas, estados, prioridades, filtros  });

  } catch (error) {

    console.error(error);

    res.send('Error al obtener tareas');

  }

};

// Formulario para crear tarea

exports.formCrearTarea = async (req, res) => {

  try {

    const estados = await Estado.find();

    const prioridades = await Prioridad.find();

    const usuarios = await Usuario.find();

    res.render('tareas/crear', {

      titulo: 'Crear Tarea',

      estados,

      prioridades,

      usuarios,

    });

  } catch (error) {

    console.error(error);

    res.send('Error al cargar formulario');

  }

};

// Crear nueva tarea

exports.crearTarea = async (req, res) => {

  const { area, titulo, descripcion, estado, prioridad, usuarioAsignado, fechaVencimiento } = req.body;

  try {

    const nuevaTarea = new Tarea({

      area,

      titulo,

      descripcion,

      estado: estado || null,

      prioridad: prioridad || null,

      usuarioAsignado: usuarioAsignado || null,

      fechaVencimiento,

    });

    await nuevaTarea.save();

    res.redirect('/tareas');

  } catch (error) {

    console.error(error);

    res.send('Error al crear tarea');

  }

};

// Formulario para editar tarea

exports.formEditarTarea = async (req, res) => {

  const { id } = req.params;

  try {

    const tarea = await Tarea.findById(id);

    const estados = await Estado.find();

    const prioridades = await Prioridad.find();

    const usuarios = await Usuario.find();

    res.render('tareas/editar', {

      titulo: 'Editar Tarea',

      tarea,

      estados,

      prioridades,

      usuarios,

    });

  } catch (error) {

    console.error(error);

    res.send('Error al obtener tarea');

  }

};

// Editar tarea

exports.editarTarea = async (req, res) => {

  const { id } = req.params;

  const { area, titulo, descripcion, estado, prioridad, usuarioAsignado, fechaVencimiento } = req.body;

  try {

    await Tarea.findByIdAndUpdate(id, {

      area,

      titulo,

      descripcion,

      estado: estado || null,

      prioridad: prioridad || null,

      usuarioAsignado: usuarioAsignado || null,

      fechaVencimiento,

    });

    res.redirect('/tareas');

  } catch (error) {

    console.error(error);

    res.send('Error al editar tarea');

  }

};

// Eliminar tarea

exports.eliminarTarea = async (req, res) => {

  const { id } = req.params;

  try {

    await Tarea.findByIdAndDelete(id);

    res.redirect('/tareas');

  } catch (error) {

    console.error(error);

    res.send('Error al eliminar tarea');

  }

};

exports.getTareas = async (req, res) => {

    try {

      const { estado, prioridad, fecha } = req.query; // Obtener filtros de la URL

      // Construir el objeto de consulta

      let query = {};

      if (estado && estado !== '') {

        query.estado = estado;

      }

      if (prioridad && prioridad !== '') {

        query.prioridad = prioridad;

      }

      if (fecha && fecha !== '') {

        // Suponiendo que deseas filtrar tareas con fecha de vencimiento igual a la fecha seleccionada

        // Puedes ajustar esto según tus necesidades (por ejemplo, mayor o menor que una fecha)

        // Aquí filtramos tareas cuya fechaVencimiento es el mismo día

        const selectedDate = new Date(fecha);

        const start = new Date(selectedDate.setUTCHours(0, 0, 0, 0));

        const end = new Date(selectedDate.setUTCHours(23, 59, 59, 999));

        query.fechaVencimiento = { $gte: start, $lte: end };

      }

      // Obtener las tareas filtradas

      const tareas = await Tarea.find(query)

        .populate('estado prioridad usuarioAsignado')

        .sort({ fechaVencimiento: 1 }); // Opcional: ordenar por fecha de vencimiento

      // Obtener todos los estados y prioridades para los filtros

      const Estado = require('../models/Estado');

      const Prioridad = require('../models/Prioridad');

      const estados = await Estado.find();

      const prioridades = await Prioridad.find();

      res.render('tareas/listar', {

        titulo: 'Lista de Tareas',

        tareas,

        estados,

        prioridades,

        filtros: { estado, prioridad, fecha }

      });

    } catch (error) {

      console.error(error);

      req.flash('error\_msg', 'Error al obtener tareas');

      res.redirect('/dashboard');

    }

  };

// src/controllers/tareaController.js

// src/controllers/usuarioController.js

const Usuario = require('../models/Usuario');

// Obtener todos los usuarios

exports.getUsuarios = async (req, res) => {

  try {

    const usuarios = await Usuario.find();

    res.render('usuarios/listar', { titulo: 'Lista de Usuarios', usuarios });

  } catch (error) {

    console.error(error);

    res.send('Error al obtener usuarios');

  }

};

// Formulario para crear usuario

exports.formCrearUsuario = (req, res) => {

  res.render('usuarios/crear', { titulo: 'Crear Usuario' });

};

// Crear nuevo usuario

exports.crearUsuario = async (req, res) => {

  const { nombre, email, contraseña, rol, departamento } = req.body;

  try {

    const nuevoUsuario = new Usuario({

      nombre,

      email,

      contraseña,

      rol,

      departamento,

    });

    await nuevoUsuario.save();

    req.flash('success\_msg', 'Usuario creado exitosamente');

    res.redirect('/usuarios');

  } catch (error) {

    console.error(error);

    req.flash('error\_msg', 'Error al crear usuario');

    res.redirect('/usuarios/crear');

  }

};

// Formulario para editar usuario

exports.formEditarUsuario = async (req, res) => {

  const { id } = req.params;

  try {

    const usuario = await Usuario.findById(id);

    res.render('usuarios/editar', { titulo: 'Editar Usuario', usuario });

  } catch (error) {

    console.error(error);

    req.flash('error\_msg', 'Error al obtener usuario');

    res.redirect('/usuarios');

  }

};

// Editar usuario

exports.editarUsuario = async (req, res) => {

  const { id } = req.params;

  const { nombre, email, contraseña, rol, departamento } = req.body;

  try {

    const usuario = await Usuario.findById(id);

    if (!usuario) {

      req.flash('error\_msg', 'Usuario no encontrado');

      return res.redirect('/usuarios');

    }

    usuario.nombre = nombre || usuario.nombre;

    usuario.email = email || usuario.email;

    usuario.rol = rol || usuario.rol;

    usuario.departamento = departamento || usuario.departamento;

    if (contraseña) {

      usuario.contraseña = contraseña; // Se encriptará automáticamente

    }

    await usuario.save();

    req.flash('success\_msg', 'Usuario actualizado exitosamente');

    res.redirect('/usuarios');

  } catch (error) {

    console.error(error);

    req.flash('error\_msg', 'Error al editar usuario');

    res.redirect(`/usuarios/editar/${id}`);

  }

};

// Eliminar usuario

exports.eliminarUsuario = async (req, res) => {

  const { id } = req.params;

  try {

    await Usuario.findByIdAndDelete(id);

    req.flash('success\_msg', 'Usuario eliminado exitosamente');

    res.redirect('/usuarios');

  } catch (error) {

    console.error(error);

    req.flash('error\_msg', 'Error al eliminar usuario');

    res.redirect('/usuarios');

  }

};

// src/database/connect.js

const mongoose = require('mongoose');

require('dotenv').config(); // Cargar las variables de entorno

const connectDB = async () => {

  try {

    await mongoose.connect(process.env.MONGODB\_URI, {

      useNewUrlParser: true,

      useUnifiedTopology: true,

    });

    console.log('Conectado a MongoDB Atlas');

  } catch (error) {

    console.error('Error al conectar a MongoDB Atlas:', error);

    process.exit(1);

  }

};

module.exports = connectDB;

// src/database/seed.js

const mongoose = require('mongoose');

const connectDB = require('./connect');

const Estado = require('../models/Estado');

const Prioridad = require('../models/Prioridad');

const Usuario = require('../models/Usuario');

const bcrypt = require('bcryptjs');

const seedData = async () => {

  await connectDB();

  const admExist = async () => await Usuario.findOne({rol: 'administrador'});

  const initDB = async () => {

    const inicializedDB = await admExist()

    console.log('admExist: ', inicializedDB );

    if (!inicializedDB) {

      // Limpiar colecciones existentes

      await Estado.deleteMany({});

      await Prioridad.deleteMany({});

      await Usuario.deleteMany({});

      // Insertar Estados

      const estados = await Estado.insertMany([

        { nombre: 'Pendiente' },

        { nombre: 'En Progreso' },

        { nombre: 'Completada' },

      ]);

      // Insertar Prioridades

      const prioridades = await Prioridad.insertMany([

        { nombre: 'Baja' },

        { nombre: 'Media' },

        { nombre: 'Alta' },

      ]);

      // Crear un administrador

      const salt = await bcrypt.genSalt(10);

      const hashedPassword = await bcrypt.hash('admin123', salt);

      const admin = new Usuario({

        nombre: 'Administrador',

        email: 'admin@example.com',

        contraseña: 'admin123', // Contraseña en texto plano,

        rol: 'administrador',

        departamento: 'Ventas',

      });

      await admin.save();

      // Crear un empleado

      const salt2 = await bcrypt.genSalt(10);

      const hashedPassword2 = await bcrypt.hash('123', salt2);

      const empleado = new Usuario({

        nombre: 'Ernesto Garcia',

        email: 'egarcia@gmail.com',

        contraseña: '123', // Contraseña en texto plano,

        rol: 'empleado',

        departamento: 'Produccion',

      });

      await empleado.save();

      console.log('Datos de Estados, Prioridades y Administrador insertados');

    } else {

     console.log('DB ya inicializada');

    }

    //mongoose.disconnect();

  }

  initDB();

};

seedData();

// src/middleware/auth.js

module.exports = {

    ensureAuthenticated: (req, res, next) => {

      if (req.session.user) {

        return next();

      }

      req.flash('error\_msg', 'Por favor, inicia sesión para continuar');

      res.redirect('/auth/login');

    },

    ensureAdmin: (req, res, next) => {

      if (req.session.user && req.session.user.rol === 'administrador') {

        return next();

      }

      req.flash('error\_msg', 'No tienes permisos para realizar esta acción');

      res.redirect('/dashboard');

    }

  };

// src/middleware/errorHandler.js

module.exports = (err, req, res, next) => {

    console.error(err.stack);

    res.status(500).send('Algo salió mal!');

  };

// src/models/Estado.js

const mongoose = require('mongoose');

const EstadoSchema = new mongoose.Schema({

  nombre: {

    type: String,

    enum: ['Pendiente', 'En Progreso', 'Completada'],

    required: true,

  },

});

module.exports = mongoose.model('Estado', EstadoSchema);

// src/models/Prioridad.js

const mongoose = require('mongoose');

const PrioridadSchema = new mongoose.Schema({

  nombre: {

    type: String,

    enum: ['Baja', 'Media', 'Alta'],

    required: true,

  },

});

module.exports = mongoose.model('Prioridad', PrioridadSchema);

// src/models/Tarea.js

const mongoose = require('mongoose');

const TareaSchema = new mongoose.Schema({

  area: {

    type: String,

    enum: ['Compras', 'Ventas', 'Producción', 'Inventario'],

    required: true,

  },

  titulo: {

    type: String,

    required: true,

  },

  descripcion: {

    type: String,

  },

  estado: {

    type: mongoose.Schema.Types.ObjectId,

    ref: 'Estado',

    default: null,

  },

  prioridad: {

    type: mongoose.Schema.Types.ObjectId,

    ref: 'Prioridad',

    default: null,

  },

  usuarioAsignado: {

    type: mongoose.Schema.Types.ObjectId,

    ref: 'Usuario',

    default: null,

  },

  fechaCreacion: {

    type: Date,

    default: Date.now,

  },

  fechaVencimiento: {

    type: Date,

  },

});

module.exports = mongoose.model('Tarea', TareaSchema);

// src/models/Usuario.js

const mongoose = require('mongoose');

const bcrypt = require('bcryptjs');

const UsuarioSchema = new mongoose.Schema({

  nombre: {

    type: String,

    required: true,

  },

  email: {

    type: String,

    required: true,

    unique: true,

    lowercase: true,

  },

  contraseña: {

    type: String,

    required: true,

  },

  rol: {

    type: String,

    enum: ['administrador', 'empleado'],

    default: 'empleado',

  },

  departamento: {

    type: String,

    enum: ['Compras', 'Ventas', 'Produccion', 'Inventario'],

    required: true,

  },

});

// Encriptar contraseña antes de guardar

UsuarioSchema.pre('save', async function (next) {

  if (!this.isModified('contraseña')) return next();

  const salt = await bcrypt.genSalt(10);

  this.contraseña = await bcrypt.hash(this.contraseña, salt);

  next();

});

// Método para comparar contraseña

UsuarioSchema.methods.matchPassword = async function (enteredPassword) {

  return await bcrypt.compare(enteredPassword, this.contraseña);

};

module.exports = mongoose.model('Usuario', UsuarioSchema);

/\* public/css/styles.css \*/

/\* Reset básico para eliminar márgenes y padding predeterminados \*/

\* {

    margin: 0;

    padding: 0;

    box-sizing: border-box;

  }

  /\* Estilos del contenedor principal \*/

  .container {

    max-width: 1200px;

    margin: 0 auto;

    padding: 20px;

    font-family: Arial, sans-serif;

    background-color: #f5f5f5;

    min-height: 100vh;

  }

  /\* Estilos de navegación \*/

  nav ul {

    list-style: none;

    display: flex;

    gap: 15px;

    margin-bottom: 20px;

  }

  nav ul li a {

    text-decoration: none;

    color: #333;

    padding: 8px 16px;

    border: 2px solid #333;

    border-radius: 4px;

    transition: background-color 0.3s, color 0.3s;

  }

  nav ul li a:hover {

    background-color: #333;

    color: #fff;

  }

  /\* Estilos para encabezados \*/

  h1 {

    margin-bottom: 20px;

    color: #333;

  }

  /\* Estilos para párrafos de mensajes \*/

  p {

    margin-bottom: 15px;

    font-size: 1rem;

  }

  /\* Estilos de formularios \*/

  form {

    background-color: #fff;

    padding: 20px;

    border-radius: 6px;

    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

  }

  form label {

    display: block;

    margin-bottom: 5px;

    font-weight: bold;

    color: #555;

  }

  form input[type="text"],

  form input[type="email"],

  form input[type="password"],

  form input[type="date"],

  form select,

  form textarea {

    width: 100%;

    padding: 8px 12px;

    margin-bottom: 15px;

    border: 1px solid #ccc;

    border-radius: 4px;

  }

  form textarea {

    resize: vertical;

    height: 100px;

  }

  form button,

  form a.button-link {

    display: inline-block;

    padding: 10px 20px;

    background-color: #333;

    color: #fff;

    text-decoration: none;

    border: none;

    border-radius: 4px;

    cursor: pointer;

    transition: background-color 0.3s;

  }

  form button:hover,

  form a.button-link:hover {

    background-color: #555;

  }

  /\* Estilos de tablas \*/

  table {

    width: 100%;

    border-collapse: collapse;

    margin-top: 20px;

    background-color: #fff;

    border-radius: 6px;

    overflow: hidden;

    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

  }

  table thead {

    background-color: #333;

    color: #fff;

  }

  table th,

  table td {

    padding: 12px 15px;

    text-align: left;

    border-bottom: 1px solid #ddd;

  }

  table tbody tr:nth-child(even) {

    background-color: #f9f9f9;

  }

  table tbody tr:hover {

    background-color: #f1f1f1;

  }

  table a {

    color: #333;

    text-decoration: none;

    padding: 6px 12px;

    border: 2px solid #333;

    border-radius: 4px;

    transition: background-color 0.3s, color 0.3s;

  }

  table a:hover {

    background-color: #333;

    color: #fff;

  }

  /\* Estilos para filtros \*/

  .filters-form {

    display: flex;

    flex-wrap: wrap;

    gap: 15px;

    margin-top: 20px;

    background-color: #fff;

    padding: 15px;

    border-radius: 6px;

    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

  }

  .filters-form div {

    flex: 1 1 200px;

    display: flex;

    flex-direction: column;

  }

  .filters-form label {

    margin-bottom: 5px;

    font-weight: bold;

    color: #555;

  }

  .filters-form button,

  .filters-form a.button-link {

    width: fit-content;

    padding: 8px 16px;

    background-color: #333;

    color: #fff;

    text-decoration: none;

    border: none;

    border-radius: 4px;

    cursor: pointer;

    transition: background-color 0.3s;

  }

  .filters-form button:hover,

  .filters-form a.button-link:hover {

    background-color: #555;

  }

  /\* Estilos para enlaces transformados en botones \*/

  a.button-link {

    display: inline-block;

    text-align: center;

  }

/\* Botones Generales \*/

.btn {

    display: inline-block;

    padding: 8px 16px;

    margin: 4px 2px;

    font-size: 14px;

    cursor: pointer;

    text-decoration: none;

    border: none;

    border-radius: 4px;

    color: white;

    transition: background-color 0.3s ease;

  }

  /\* Botón Principal (Azul) \*/

  .btn-primary {

    background-color: #007BFF;

  }

  .btn-primary:hover {

    background-color: #0056b3;

  }

  /\* Botón de Peligro (Rojo) \*/

  .btn-danger {

    background-color: #DC3545;

  }

  .btn-danger:hover {

    background-color: #a71d2a;

  }

  /\* Botón Secundario (Gris) \*/

  .btn-secondary {

    background-color: #6C757D;

  }

  .btn-secondary:hover {

    background-color: #5a6268;

  }

  /\* Opcional: Botón para acciones pequeñas \*/

  .btn-sm {

    padding: 4px 8px;

    font-size: 12px;

  }

// src/routes/auth.js

const express = require('express');

const router = express.Router();

const authController = require('../controllers/authController');

// Formulario de inicio de sesión

router.get('/login', authController.formLogin);

// Procesar inicio de sesión

router.post('/login', authController.login);

// Cerrar sesión

router.get('/logout', authController.logout);

module.exports = router;

// src/routes/tareas.js

const express = require('express');

const router = express.Router();

const tareaController = require('../controllers/tareaController');

const { ensureAuthenticated, ensureAdmin } = require('../middleware/auth');

// Obtener todas las tareas

router.get('/', tareaController.getTareas);

// Obtener tareas ordenadas

router.get('/ordenadas', tareaController.getTareasOrdenadas);

// Formulario para crear tarea

router.get('/crear', tareaController.formCrearTarea);

// Todas las rutas de tareas requieren autenticación

router.use(ensureAuthenticated);

// Ruta para listar tareas con filtros

router.get('/', tareaController.getTareas);

// Crear nueva tarea

router.post('/crear', tareaController.crearTarea);

// Formulario para editar tarea

router.get('/editar/:id', tareaController.formEditarTarea);

// Editar tarea

router.post('/editar/:id', tareaController.editarTarea);

// Eliminar tarea

router.get('/eliminar/:id', tareaController.eliminarTarea);

module.exports = router;

// src/routes/usuarios.js

const express = require('express');

const router = express.Router();

const usuarioController = require('../controllers/usuarioController');

const { ensureAuthenticated, ensureAdmin } = require('../middleware/auth');

// Todas las rutas de usuarios requieren autenticación

router.use(ensureAuthenticated);

// Obtener todos los usuarios (solo administradores)

router.get('/', ensureAdmin, usuarioController.getUsuarios);

// Formulario para crear usuario (solo administradores)

router.get('/crear', ensureAdmin, usuarioController.formCrearUsuario);

// Crear nuevo usuario (solo administradores)

router.post('/crear', ensureAdmin, usuarioController.crearUsuario);

// Formulario para editar usuario (solo administradores)

router.get('/editar/:id', ensureAdmin, usuarioController.formEditarUsuario);

// Editar usuario (solo administradores)

router.post('/editar/:id', ensureAdmin, usuarioController.editarUsuario);

// Eliminar usuario (solo administradores)

router.get('/eliminar/:id', ensureAdmin, usuarioController.eliminarUsuario);

module.exports = router;

// src/test/matchPasswordTest.js

const mongoose = require('mongoose');

const connectDB = require('../database/connect');

const Usuario = require('../models/Usuario');

const testMatchPassword = async () => {

  await connectDB();

  // Buscar al administrador

  const admin = await Usuario.findOne({ email: 'admin@example.com' });

  if (!admin) {

    console.log('Administrador no encontrado');

    mongoose.disconnect();

    return;

  }

  // Contraseña correcta

  const isMatchCorrect = await admin.matchPassword('admin123');

  console.log('Contraseña correcta coincide:', isMatchCorrect); // Debería ser true

  // Contraseña incorrecta

  const isMatchIncorrect = await admin.matchPassword('wrongpassword');

  console.log('Contraseña incorrecta coincide:', isMatchIncorrect); // Debería ser false

  mongoose.disconnect();

};

testMatchPassword();

// src/utils/gestorTareas.js

function calcularPrioridad(estado, fechaVencimiento) {

    const hoy = new Date();

    const fechaTarea = new Date(fechaVencimiento);

    if (estado === 'Completada') {

      return 'Baja';

    } else if (fechaTarea < hoy) {

      return 'Alta';

    } else {

      return 'Media';

    }

  }

  module.exports = { calcularPrioridad };

// src/utils/gestorTareas.test.js

const { calcularPrioridad } = require('./gestorTareas');

describe('Función calcularPrioridad', () => {

  test('Debe devolver "Baja" si la tarea está completada', () => {

    const resultado = calcularPrioridad('Completada', '2024-10-01');

    expect(resultado).toBe('Baja');

  });

  test('Debe devolver "Alta" si la tarea está vencida', () => {

    const resultado = calcularPrioridad('Pendiente', '2022-10-01');

    expect(resultado).toBe('Alta');

  });

  test('Debe devolver "Media" si la tarea está pendiente y la fecha no ha pasado', () => {

    const fechaFutura = new Date();

    fechaFutura.setDate(fechaFutura.getDate() + 7); // 7 días en el futuro

    const resultado = calcularPrioridad('Pendiente', fechaFutura.toISOString().split('T')[0]);

    expect(resultado).toBe('Media');

  });

});

//- src/views/layout.pug

doctype html

html

  head

    title= titulo

    link(rel='stylesheet', href='/css/styles.css')

  body

    .container

      nav

        ul

          if user

            li: a.btn.btn-primary(href='/dashboard') Dashboard

            if user.rol === 'administrador'

              li: a.btn.btn-primary(href='/usuarios') Usuarios

            li: a.btn.btn-primary(href='/tareas') Tareas

            li: a.btn.btn-secondary(href='/auth/logout') Cerrar Sesión

          else

            li: a.btn.btn-primary(href='/auth/login') Iniciar Sesión

      if error\_msg

        p(style="color: red")= error\_msg

      if success\_msg

        p(style="color: green")= success\_msg

      block content

//- src/views/login.pug

extends layout

block content

  h1= titulo

  if error\_msg

    p(style="color: red")= error\_msg

  if success\_msg

    p(style="color: green")= success\_msg

  form(action='/auth/login', method='POST')

    label(for='email') Email:

    input(type='email', name='email', id='email', required)

    label(for='contraseña') Contraseña:

    input(type='password', name='contraseña', id='contraseña', required)

    button(type='submit') Iniciar Sesión

//- src/views/index.pug

extends layout

block content

  h1= titulo

  p Bienvenido al Sistema de Gestión de Tareas.

  p Utiliza la navegación para gestionar usuarios y tareas.

//- src/views/dashboard.pug

extends layout

block content

  h1= titulo

  p Bienvenido, #{user.nombre}!

  if user.rol === 'administrador'

    div

      a.btn.btn-primary(href='/usuarios') Gestionar Usuarios

      a.btn.btn-primary(href='/tareas') Gestionar Tareas

  else

    div

      a.btn.btn-primary(href='/tareas') Mis Tareas

  a.btn.btn-secondary(href='/auth/logout') Cerrar Sesión

//- src/views/tareas/crear.pug

extends ../layout

block content

  h1= titulo

  form(action='/tareas/crear', method='POST')

    label(for='titulo') Título:

    input(type='text', name='titulo', id='titulo', required)

    label(for='area') Área:

    select(name='area', id='area', required)

      option(value='Compras') Compras

      option(value='Ventas') Ventas

      option(value='Producción') Producción

      option(value='Inventario') Inventario

    label(for='descripcion') Descripción:

    textarea(name='descripcion', id='descripcion')

    label(for='estado') Estado:

    select(name='estado', id='estado')

      option(value='') Sin Estado

      each estado in estados

        option(value=estado.\_id)= estado.nombre

    label(for='prioridad') Prioridad:

    select(name='prioridad', id='prioridad')

      option(value='') Sin Prioridad

      each prioridad in prioridades

        option(value=prioridad.\_id)= prioridad.nombre

    label(for='usuarioAsignado') Usuario Asignado:

    select(name='usuarioAsignado', id='usuarioAsignado')

      option(value='') Sin Asignar

      each usuario in usuarios

        option(value=usuario.\_id)= usuario.nombre

    label(for='fechaVencimiento') Fecha de Vencimiento:

    input(type='date', name='fechaVencimiento', id='fechaVencimiento')

    button(type='submit') Crear Tarea

//- src/views/tareas/editar.pug

extends ../layout

block content

  h1= titulo

  form(action=`/tareas/editar/${tarea.\_id}`, method='POST')

    label(for='titulo') Título:

    input(type='text', name='titulo', id='titulo', value=tarea.titulo, required)

    label(for='area') Área:

    select(name='area', id='area', required)

      option(value='Compras', selected=tarea.area === 'Compras') Compras

      option(value='Ventas', selected=tarea.area === 'Ventas') Ventas

      option(value='Producción', selected=tarea.area === 'Producción') Producción

      option(value='Inventario', selected=tarea.area === 'Inventario') Inventario

    label(for='descripcion') Descripción:

    textarea(name='descripcion', id='descripcion')= tarea.descripcion

    label(for='estado') Estado:

    select(name='estado', id='estado')

      option(value='') Sin Estado

      each estado in estados

        option(

          value=estado.\_id

          selected=(tarea.estado && tarea.estado.\_id.toString() === estado.\_id.toString())

        )= estado.nombre

    label(for='prioridad') Prioridad:

    select(name='prioridad', id='prioridad')

      option(value='') Sin Prioridad

      each prioridad in prioridades

        option(

          value=prioridad.\_id

          selected=(tarea.prioridad && tarea.prioridad.\_id.toString() === prioridad.\_id.toString())

        )= prioridad.nombre

    label(for='usuarioAsignado') Usuario Asignado:

    select(name='usuarioAsignado', id='usuarioAsignado')

      option(value='') Sin Asignar

      each usuario in usuarios

        option(

          value=usuario.\_id

          selected=(tarea.usuarioAsignado && tarea.usuarioAsignado.\_id.toString() === usuario.\_id.toString())

        )= usuario.nombre

    label(for='fechaVencimiento') Fecha de Vencimiento:

    input(

      type='date'

      name='fechaVencimiento'

      id='fechaVencimiento'

      value=tarea.fechaVencimiento ? tarea.fechaVencimiento.toISOString().split('T')[0] : ''

    )

    button(type='submit') Actualizar Tarea

//- src/views/tareas/listar.pug

extends ../layout

block content

  h1= titulo

  a.btn.btn-primary(href='/tareas/crear') Crear Nueva Tarea

  table

    thead

      tr

        th(onclick='location.href="/tareas/ordenadas?sort=titulo"') Título

        th(onclick='location.href="/tareas/ordenadas?sort=area"') Área

        th(onclick='location.href="/tareas/ordenadas?sort=estado"') Estado

        th(onclick='location.href="/tareas/ordenadas?sort=prioridad"') Prioridad

        th(onclick='location.href="/tareas/ordenadas?sort=usuarioAsignado"') Usuario Asignado

        th(onclick='location.href="/tareas/ordenadas?sort=fechaVencimiento"') Fecha de Vencimiento

        th Acciones

    tbody

      each tarea in tareas

        tr

          td= tarea.titulo

          td= tarea.area

          td= tarea.estado ? tarea.estado.nombre : 'Sin Estado'

          td= tarea.prioridad ? tarea.prioridad.nombre : 'Sin Prioridad'

          td= tarea.usuarioAsignado ? tarea.usuarioAsignado.nombre : 'Sin Asignar'

          td= tarea.fechaVencimiento ? tarea.fechaVencimiento.toISOString().split('T')[0] : 'Sin Fecha'

          td

            a.btn.btn-primary.btn-sm(href=`/tareas/editar/${tarea.\_id}`) Editar

            a.btn.btn-danger.btn-sm(href=`/tareas/eliminar/${tarea.\_id}`) Eliminar

  // Mostrar mensajes flash si existen

  if error\_msg

    p(style="color: red")= error\_msg

  if success\_msg

    p(style="color: green")= success\_msg

  // Formulario de filtros

  form(action='/tareas', method='GET', class='filters-form')

    div

      label(for='estado') Estado:

      select(name='estado', id='estado')

        option(value='') Todos

        each est of estados

          option(value=est.\_id, selected=(filtros.estado == est.\_id.toString()))= est.nombre

    div

      label(for='prioridad') Prioridad:

      select(name='prioridad', id='prioridad')

        option(value='') Todos

        each pri in prioridades

          option(value=pri.\_id, selected=(filtros.prioridad == pri.\_id.toString()))= pri.nombre

    div

      label(for='fecha') Fecha de Vencimiento:

      input(

        type='date'

        name='fecha'

        id='fecha'

        value=filtros.fecha || ''

      )

    div

      button.btn.btn-primary(type='submit') Filtrar

      a.btn.btn-secondary(href='/tareas') Limpiar Filtros

//- src/views/usuarios/crear.pug

extends ../layout

block content

  h1= titulo

  form(action='/usuarios/crear', method='POST')

    label(for='nombre') Nombre:

    input(type='text', name='nombre', id='nombre', required)

    label(for='email') Email:

    input(type='email', name='email', id='email', required)

    label(for='contraseña') Contraseña:

    input(type='password', name='contraseña', id='contraseña', required)

    label(for='rol') Rol:

    select(name='rol', id='rol')

      option(value='empleado') Empleado

      option(value='administrador') Administrador

    label(for='departamento') Departamento:

    select(name='departamento', id='departamento', required)

      option(value='Compras') Compras

      option(value='Ventas') Ventas

      option(value='Produccion') Producción

      option(value='Inventario') Inventario

    button(type='submit') Crear Usuario

//- src/views/usuarios/editar.pug

extends ../layout

block content

  h1= titulo

  form(action=`/usuarios/editar/${usuario.\_id}`, method='POST')

    label(for='nombre') Nombre:

    input(type='text', name='nombre', id='nombre', value=usuario.nombre, required)

    label(for='email') Email:

    input(type='email', name='email', id='email', value=usuario.email, required)

    label(for='contraseña') Contraseña:

    input(type='password', name='contraseña', id='contraseña', placeholder='Ingresa nueva contraseña si deseas cambiarla')

    label(for='rol') Rol:

    select(name='rol', id='rol')

      option(value='empleado', selected=usuario.rol === 'empleado') Empleado

      option(value='administrador', selected=usuario.rol === 'administrador') Administrador

    label(for='departamento') Departamento:

    select(name='departamento', id='departamento', required)

      option(value='Compras', selected=usuario.departamento === 'Compras') Compras

      option(value='Ventas', selected=usuario.departamento === 'Ventas') Ventas

      option(value='Producción', selected=usuario.departamento === 'Producción') Producción

      option(value='Inventario', selected=usuario.departamento === 'Inventario') Inventario

    button(type='submit') Actualizar Usuario

//- src/views/usuarios/listar.pug

extends ../layout

block content

  h1= titulo

  a.btn.btn-primary(href='/usuarios/crear') Crear Nuevo Usuario

  table

    thead

      tr

        th Nombre

        th Email

        th Rol

        th Departamento

        th Acciones

    tbody

      each usuario in usuarios

        tr

          td= usuario.nombre

          td= usuario.email

          td= usuario.rol

          td= usuario.departamento

          td

            a.btn.btn-primary.btn-sm(href=`/usuarios/editar/${usuario.\_id}`) Editar

            a.btn.btn-danger.btn-sm(href=`/usuarios/eliminar/${usuario.\_id}`) Eliminar

MONGODB\_URI=mongodb+srv://christianjavierlemos:C2X35tZ4DOOo3XLI@cluster0.gk9fm.mongodb.net/sistema-gestion-tareas

{

  "name": "gestion\_tareas\_v3",

  "version": "1.0.0",

  "description": "",

  "main": "index.js",

  "scripts": {

    "start": "node src/app.js",

    "dev": "nodemon src/app.js",

    "test": "jest"

  },

  "keywords": [],

  "author": "",

  "license": "ISC",

  "dependencies": {

    "bcryptjs": "^2.4.3",

    "body-parser": "^1.20.3",

    "connect-flash": "^0.1.1",

    "dotenv": "^16.4.5",

    "express": "^4.21.1",

    "express-session": "^1.18.1",

    "mongoose": "^8.7.1",

    "morgan": "^1.10.0",

    "pug": "^3.0.3"

  },

  "devDependencies": {

    "eslint": "^9.12.0",

    "jest": "^29.7.0",

    "nodemon": "^3.1.7"

  }

}