1. Requerimientos funcionales

1. Pantalla de registro
    - El sistema debe mostrar una pantalla titulada “Regístrate”.
    - La pantalla debe contener los campos:
        - Nombre
        - Correo electrónico
        - Contraseña
        - Rol (Estudiante, Profesor, Administrador)
    - Debe haber un botón “Registrarse”.
    - Debe mostrarse un enlace con el texto “¿Ya tienes una cuenta? Inicia sesión”.
2. Campo Nombre
    - RF2.1 El usuario debe poder capturar su nombre completo en el campo Nombre.
    - RF2.2 Si el campo está vacío o solo contiene espacios, el sistema debe marcar el campo como inválido y mostrar el mensaje: “Campo vacío.”.
    - RF2.3 El sistema debe validar que:
        - Solo existan letras (A–Z, a–z) y espacios.
        - Cada palabra tenga al menos 2 caracteres.
        - No existan dobles espacios ni espacios al inicio o al final.
    - RF2.4 Si el valor no cumple estas condiciones, se debe mostrar el mensaje: “El nombre tiene otro caracter que no es letra, tiene menos de dos caracteres o tiene doble espacios.”
3. Campo Correo electrónico
    - RF3.1 El usuario debe poder capturar su correo en el campo Correo electrónico.
    - RF3.2 Si el campo está vacío o solo contiene espacios, el sistema debe marcar el campo como inválido y mostrar: “Campo vacío.”.

- RF3.3 El sistema debe validar el formato del correo usando la expresión: ^[a-zA-Z0-9._]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$
    - Debe contener una parte local con letras, números, puntos o guiones bajos.
    - Debe tener un @.
    - Debe tener un dominio con letras, números, puntos o guiones.
    - Debe tener una terminación de al menos 2 letras (por ejemplo .com, .mx).
- RF3.4 Si el formato es inválido, se debe mostrar “Email invalido.”.
- RF3.5 El sistema debe verificar que el correo no exista ya registrado:
    - Debe realizar una petición GET a http://localhost:3030/api/users.
    - Debe recorrer la lista de usuarios y comparar user.email con el correo capturado.
- RF3.6 Si el correo ya está registrado, se debe mostrar “El correo ya existe.”.
- RF3.7 Si el correo no está vacío, tiene formato válido y no existe en la base de datos, el campo debe considerarse válido.

4. Campo Contraseña
    - RF4.1 El usuario debe poder capturar una contraseña en el campo Contraseña.
    - RF4.2 Si el campo está vacío o solo contiene espacios, el sistema debe mostrar “Campo vacío.”.
    - RF4.3 La contraseña debe cumplir las siguientes reglas (según el código):
        - Longitud mínima 8 y máxima 20 caracteres.
        - Contener al menos una letra mayúscula (A–Z).
        - Contener al menos una letra minúscula (a–z).
        - Contener al menos un dígito (0–9).
        - Contener entre 1 y 3 caracteres especiales del conjunto [!@#$%&*-_+].

- RF4.4 Si no cumple estas reglas, se debe marcar como inválida y mostrar un mensaje de error indicando los requisitos.

5. Campo Rol
    - RF5.1 El sistema debe mostrar las opciones de rol como botones de opción (radio buttons):
        - Estudiante
        - Profesor
        - Administrador
    - RF5.2 El usuario debe poder seleccionar exactamente un rol.
6. Validación en tiempo real
    - RF6.1 El sistema debe realizar la validación de los campos mientras el usuario escribe (evento input).
    - RF6.2 Cuando un campo no cumpla las reglas, debe utilizarse setCustomValidity y reportValidity() para mostrar al usuario el mensaje correspondiente.
    - RF6.3 Cuando el campo se corrija, el mensaje de error debe eliminarse y el campo se considere válido.
7. Registro del usuario
    - RF7.1 El botón “Registrarse” solo debe permitir enviar el formulario cuando todos los campos obligatorios (Nombre, Correo electrónico, Contraseña, Rol) sean válidos.
    - RF7.2 Si algún campo es inválido, el navegador debe impedir el envío y mostrar el mensaje de error definido para el campo.

2. Requerimientos no funcionales

1. Usabilidad
    - RNF1.1 El sistema debe proporcionar mensajes de error claros y en español, indicando la causa del problema (campo vacío, formato incorrecto, contraseña débil, correo ya existente).

- RNF1.2 Las validaciones deben ocurrir mientras el usuario escribe para reducir errores al momento de enviar el formulario.

2. Rendimiento
    - RNF2.1 La verificación de correo existente debe realizarse mediante una llamada a la API (fetch a http://localhost:3030/api/users) de forma asíncrona, sin bloquear la interfaz.
    - RNF2.2 La respuesta del servidor debe ser lo suficientemente rápida para que la experiencia de escritura no se sienta lenta.
3. Seguridad
    - RNF3.1 Las contraseñas deben cumplir una política de complejidad (mayúsculas, minúsculas, dígitos y caracteres especiales).
    - RNF3.2 Debe existir una validación equivalente del lado del servidor (no solo en el cliente) para evitar registros inválidos del frontend.
    - RNF3.3 La comunicación con la API de usuarios debe realizarse sobre un canal seguro (por ejemplo, HTTPS) en un entorno real.
4. Mantenibilidad
    - RNF4.1 Las funciones de validación (checkEmail, validPassword) deben estar separadas del código de interfaz para facilitar su modificación y pruebas.
    - RNF4.2 Los mensajes de error deben centralizarse para evitar inconsistencias entre la lógica y lo que se muestra al usuario.
5. Compatibilidad
    - RNF5.1 La solución debe funcionar en navegadores modernos que soporten fetch, setCustomValidity y reportValidity.