UNIDAD 1

JAVASCRIPT

Ejercicio final – InmoSanvi

Desarrollo web del lado del cliente

2do curso – DAW
IES San Vicente 2025/2026
Arturo Bernal / Rosa Medina

Índice

Introducción

IMPORTANTEEs fundamental leer todo este documento, especialmente la sección "Requisitos generales y notas importantes", antes de comenzar a trabajar en el proyecto.

Este último ejercicio consistirá en una adaptación a TypeScript y también en una ampliación de las actividades que hemos estado realizando durante esta unidad (hasta la semana 4).

Te proporcionaré la estructura básica de la aplicación y tú tendrás que completar los archivos TypeScript necesarios. Esto es lo que te entrego:

- Las clases auxiliares**HTTP**,**Mi geolocalización**y**Servicio de mapas**(MapService es útil para crear y administrar mapas)
- Un archivo de configuración de eslint (formato TypeScript).
- Todo lo necesario**archivos HTML**No edites ningún HTML a menos que esté muy justificado (**ejemplo**: partes opcionales como la implementación de CropperJS)
- Nuevos iconos (colócalos en la carpeta de iconos)
- Una colección de Postman para probar la web**servicios**antes de implementarlas

Servicios web

El código fuente de los servicios web se puede encontrar en este repositorio de Github: https://github.com/arturober/inmosanvi-services.

Clona el repositorio y**siempre**Inicie los servicios antes de trabajar en el proyecto:

- Correr**npm i**(primera vez) y**npm start**
- Utilice la URL http://localhost:3000 en su cliente y Postman como base.

Configuración inicial

Sigue estos pasos para configurar el entorno de tu proyecto:

1. **Crear proyecto:** Inicializar un nuevo **Vite** proyecto utilizando el **Mecanografiado** plantilla.
2. **Activos:** Copia las imágenes de la carpeta pública de la tarea anterior a la carpeta pública de tu nuevo proyecto.
3. **Extensión de VS Code (recomendada):** Instala la extensión "Material Icon Theme" en VS Code para una mejor diferenciación visual entre tipos de archivos.
4. **Instalar dependencias:**
   - Instala ol, tailwindcss y @tailwindcss/vite como dependencias regulares.
   - Instala Prettier como un **dependencia de desarrollo** (utilice la bandera -D).
5. **Configuración de ESLint:** Instala eslint en tu proyecto (npm init). @eslint/config@latest) y reemplace el archivo eslint.config.ts predeterminado con el proporcionado. Esta configuración se utilizará para la evaluación.
6. **Configuración más bonita:** Crea un **.más bonito** Archivo en la raíz de su proyecto con la siguiente configuración. **Debes formatear tu código antes de enviarlo.**

```
{
    "coma final":"es5", "ancho de
    pestaña":2, "semi":verdadero,

    "comilla simple":FALSO,
    "Espaciado entre corchetes":
    verdadero, "arrowParens":"evitar"
}
```

7. **Archivos ignorados (opcional):** Si no te gusta el formato HTML predeterminado de Prettier, crea un **.más bonito ignorar** archivo y agregue la regla: ***.html**.
8. **Archivos del proyecto:** Copia los archivos HTML y CSS proporcionados en tu proyecto. Los archivos index.html y new_property.html son prácticamente idénticos a los de la tarea anterior; los cambios se detallarán en sus respectivas secciones.
9. **Configuración de Vite:** Crea y configura vite.config.js para que gestione correctamente todos tus archivos HTML. **Antes de enviar tu proyecto, verifica que al ejecutar el comando de compilación (npm run build) se generen todos los archivos HTML en la carpeta dist.**

Además, configura el plugin de Tailwind.: https://tailwindcss.com/docs/installation/using-vite

```
complementos:[
    viento de cola css(),
],
```

10. **Migración de código:**
    1. Elimine los archivos de muestra que se encuentran dentro de la carpeta src.
    2. Copie los archivos JavaScript de la tarea anterior en src.
    3. Cambie el nombre de todos los archivos .js para que tengan la extensión .ts.
11. **Refactorización de TypeScript:** Refactoriza el código JavaScript existente para que sea totalmente compatible con TypeScript. Se te proporcionará la clase Http junto con otros archivos necesarios del proyecto, como MapService.

Interfaces requeridas

Crea las siguientes interfaces de TypeScript para modelar tus estructuras de datos. Se recomienda organizarlas en archivos lógicos (por ejemplo, property.interfaces.ts, user.interfaces.ts).

- **Coordenadas**Representa coordenadas geográficas con latitud y latitud. longitud.
- **Datos de inicio de sesión**Representa los datos enviados durante el inicio de sesión (correo electrónico y contraseña).
- **Datos de registro**Representa los datos enviados durante el registro. Debe **extender**Datos de inicio de sesión y agregue los campos necesarios.
- **Usuario**Representa los datos de usuario devueltos por el servidor para un perfil. debería**Extiende RegisterData pero omite el campo de contraseña.**(Por ejemplo, extiende Omit<RegisterData, "password">) y agrega cualquier otro campo requerido de la respuesta del servidor.
  - **Perfil de usuario**,**Avatar de usuario**,**Contraseña de usuario**((para la página de perfil)
- **Provincia**Representa los datos de la provincia tal como los devuelve el servidor.
- **Ciudad**Representa los datos de la ciudad tal como los devuelve el servidor.**Nota:**La provincia La propiedad puede ser un número (ID) o un objeto completo de Provincia. Utilice un tipo unión (número | Provincia) para gestionar esto.
- **Insertar propiedad**Representa los datos de una nueva propiedad inmobiliaria que es enviado al servidor (solicitud POST).
- **Propiedad**Representa una propiedad inmobiliaria tal como la devuelve el servidor. debería**extender PropertyInsert**Omitir los campos que solo se utilizan para la inserción (p. ej., townId mediante Omit<PropertyInsert, "townId">) y agregar los campos adicionales. Si algunos campos solo aparecen ocasionalmente en la respuesta, marcarlos como**opcional** (usando ?).
- **PropertiesResponse y SinglePropertyResponse**Representar la totalidad Respuestas del servidor para una lista de propiedades y una sola propiedad, respectivamente.
- **Respuesta de usuario único**Representa la respuesta del servidor al obtener un Datos de un solo usuario.
- **Respuesta de token**Representa la respuesta del servidor tras un inicio de sesión exitoso. que contiene el token de autenticación.
- **Actualización de avatar y actualización de contraseña**Interfaces para editar el avatar del usuario y contraseña. Incluya también una interfaz para la respuesta del servidor.
- **((Opcional) CalificaciónInsertar y Calificación**Para la función de comentarios opcional, Crea una interfaz para enviar una nueva calificación (RatingInsert) y otra para un objeto de comentario completo (Rating).
- **((Opcional) SingleRatingResponse y RatingsResponse**Interfaces para Las respuestas del servidor al obtener una o varias calificaciones.

Puedes crear cualquier otra interfaz que consideres necesaria.

Página de inicio de sesión (login.html)

Esta página**iniciar sesión.html**Tendrá un formulario de inicio de sesión. En este formulario, el usuario ingresará su correo electrónico y contraseña.

El servicio web al que se debe llamar para iniciar sesión es: POST →http://SERVIDOR/autenticación/inicio de sesión

Ejemplo de solicitud de datos:

```
{
    "correo electrónico":"test@test.com ",
    "contraseña":"1234",
}
```

Respuesta: Devuelve un token de autenticación que debe almacenar (localStorage):

```
{
  "token de acceso":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NywiaWF0IjoxNjAzODIxNTcxLCJleHAiOjE2MzUzNTc1NzF9.XK1RY-5C4UvhQlb7d8ack2718lKrx71va1ukURz7_NI"
}
```

Cualquier error dará como resultado un estado diferente de 200, generalmente**401**((no autorizado). Uso**atrapar**para capturarlo. Contendrá el siguiente JSON:

```
{
  "estado":401,
  "error":"Correo electrónico o contraseña incorrectos
}
```

Cuando el inicio de sesión sea exitoso, guarde el**simbólico**dentro de un**Almacenamiento local** variable (use la clave '**simbólico**(La clase HTTP está configurada para obtener este valor). Necesitará este token en otros servicios web; de lo contrario, devolverán un error de autenticación (401). La clase HTTP lo hará automáticamente siempre que haya almacenado el token previamente.

Una vez que el inicio de sesión se haya realizado correctamente, se le redirigirá a**index.html**.

Si se produce un error, muéstralo al usuario (puedes usar una alerta normal o, opcionalmente, una biblioteca como...).alerta dulce , Por ejemplo).

En página carga, controlar si el usuario es ya Registrado en (llamar AuthService.checkToken). Si es así, rediríjalos a index.html.

Página de registro (registrés.html)

En esta página encontrará un formulario para registrar un nuevo usuario. En este formulario deberá enviar la siguiente información:

- **nombre**
- **correo electrónico**
- **contraseña**
- **avatar**→ Foto del usuario enviada en formato base64.

Validación de contraseña:

- Agrega un evento de entrada a ambos campos de contraseña y verifica si sus valores coinciden. De no ser así, establece un error de validación personalizado en el segundo campo.
- Crea una función independiente para esta comprobación y llámala desde los eventos de entrada de ambos campos de contraseña. El error siempre debe estar asociado al segundo campo.
- **Recuerda borrar el error**(setCustomValidity('')) cuando coincidan, de lo contrario no se podrá enviar el formulario.

Vista previa de la imagen: Implemente la misma funcionalidad de vista previa de imágenes que en el formulario "Agregar nueva propiedad".

Enviar: POST →http://SERVIDOR/autenticación/registro .

Este servicio devolverá un estado 200 (OK) que contiene el objeto de usuario insertado o un error, generalmente estado 400 (Solicitud incorrecta), como este:


```
{
  "statusCode":400,
  "mensaje":[
    "El nombre no debe estar vacío", "El correo electrónico
    no debe estar vacío", "El correo electrónico debe ser
    un correo electrónico", "La contraseña no debe estar
    vacía", "La contraseña debe ser una cadena de
    caracteres"
  ],
  "error":"Solicitud incorrecta"
}
```


Si recibes algún mensaje de error, muéstralo todo al usuario. Si todo ha ido bien, redirige a la página de inicio de sesión .

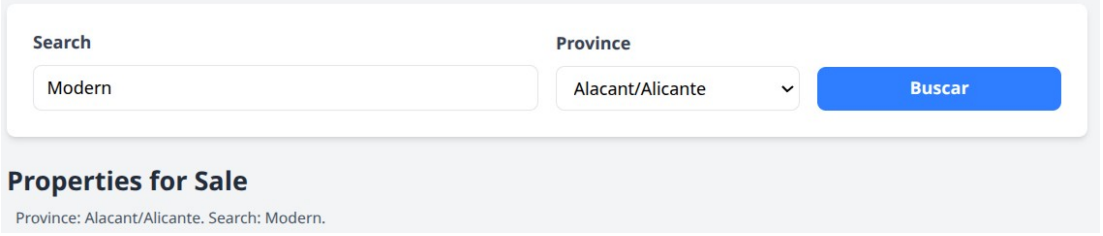
Página de propiedades (index.html)

En esta página se mostrarán las propiedades, al igual que en los ejercicios anteriores. Llamar a GET → http://SERVIDOR/propiedades Para conseguirlas. Estas son las principales diferencias:

- **Visualización de datos:**Las descripciones de las propiedades deben**no**Ya no se mostrarán en las tarjetas; se mostrarán en la página de detalles. La imagen y el título están dentro. **campo de golf**que debería ir a la página de detalles. No olvides configurar el**href** atributo con el id (ej.: property-detail.html?id=5)
- **Control de la propiedad:**El botón "Eliminar" en una tarjeta de propiedad debería**solo ser visible**Si la propiedad pertenece al usuario que ha iniciado sesión, el objeto de propiedad incluirá un**mío: booleano**campo cuando te autenticas.
- **Confirmación de eliminación:**Antes de eliminar una propiedad, utilice el cuadro de diálogo confirm() para pedir confirmación al usuario.
- **Paginación:**El servidor devuelve las propiedades en páginas de 12. Implemente el botón "Cargar más" para obtener la página siguiente.
- **Filtración:**
  - Implemente la barra de búsqueda de texto y el menú desplegable de provincias. Al hacer clic en el botón "Filtrar", se deben actualizar las variables de estado globales y mostrar la página 1 con los nuevos parámetros del filtro.
  - Al hacer clic en "Cargar más" debería incrementarse el número de página actual y mostrarse el siguiente conjunto de resultados.**agregando**agregarlos a la lista existente. La lista solo debe borrarse cuando se aplique un nuevo filtro (es decir, al cargar la página 1).
  - Utilice URLSearchParams para construir la URL de la solicitud con parámetros de búsqueda (página, provincia, búsqueda y, opcionalmente, vendedor). Los parámetros numéricos como página o provincia pueden enviarse como "0" para indicar "sin filtro" si no se permite una cadena vacía.

```
constante parámetros = nuevo Parámetros de búsqueda de URL({ página: Cadena(página), provincia, buscar });
espera de vuelta esto.#http.conseguir(`${SERVIDOR}/¿propiedades?${parámetros.()}`);
```

- Estado de la interfaz de usuario:
  - El botón "Cargar más" debería ser**oculto**((añadiendo la clase oculta) cuando la respuesta del servidor incluye**más: falso**en la respuesta. Debería mostrarse de nuevo cuando**más: verdadero**.
  - Rellene el menú desplegable de provincias obteniendo las provincias del servidor.
  - Conservar el estado de búsqueda actual (página, provincia, texto) en variables globales. Mostrar un mensaje de retroalimentación al usuario debajo de la barra de búsqueda indicando los filtros activos. Ejemplo:

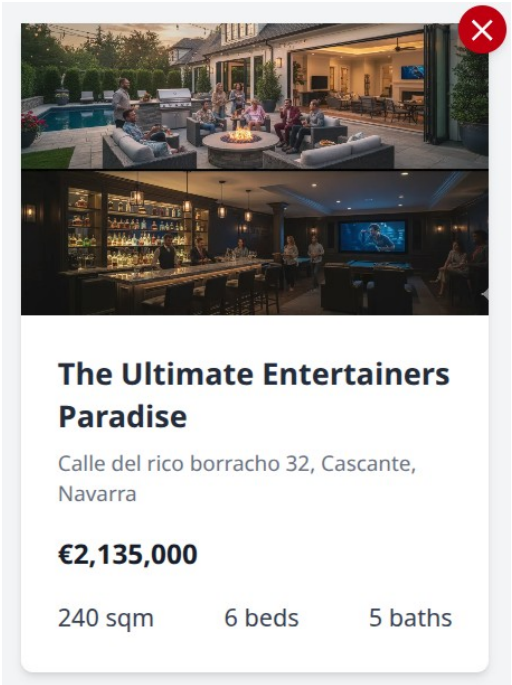


- **Autenticación:**
  - Al cargar la página, verifique si el usuario ha iniciado sesión utilizando AuthService.checktoken().
  - Si ha iniciado sesión, oculte el elemento.**.enlace de inicio de sesión**en el menú superior y mostrar el**.enlace de nueva propiedad**,**.enlace de perfil**, y**.enlace de cierre de sesión**elementos.
  - El**.logout-link**El botón debe tener un evento de clic que llame a **ServicioAutenticación.cerrarSesión()**y redirige al usuario a la página de inicio de sesión.
- **((Opcional) Filtrar por vendedor:**Si la URL contiene un parámetro de consulta de vendedor (p. ej., index.html?seller=5), inclúyalo en su solicitud a la API para obtener solo las propiedades de ese usuario. Actualice el mensaje de retroalimentación del filtro para incluir el nombre del vendedor.

Ejemplo de un objeto de propiedad que el servidor devuelve cuando inicias sesión:


```
{
    "identificación":17,
    "DIRECCIÓN":"Calle del rico borracho 32", "título":"El paraíso
    definitivo para los artistas,
    "descripción":"Organice reuniones inolvidables en esta casa diseñada para el entretenimiento. La sala de estar de concepto abierto se conecta fluidamente con un amplio patio exterior con barbacoa y fogón integrados.[1] Un sótano terminado con bar y sala de cine en casa ofrece espacio adicional para la diversión y el ocio. Con amplio estacionamiento y una distribución acogedora, esta casa está lista para su próxima celebración.,

    "metros cuadrados":  240,
    "numRooms":  6,
    "numBaños":  5,
    "precio":2135000,
    "Calificación total":  5,
    "foto principal":  "http://localhost:3000/img/properties/1760885974657.jpg",
    "creadoEn":  "2025-10-19T14:59:34.657Z", "venta",
    "estado":
    "ciudad": {
        "identificación":  1310689,
        "nombre":  "Cascante",
        "longitud":  - 1.67903193,
        "latitud":  41.99909834,
        "provincia":  {
            "identificación":31,
            "nombre":"Navarra"
        }
    },
    "vendedor":1,
    "mío":verdadero
}
```

Nueva página de propiedad (new-property.html)

Esta página contendrá un formulario para crear una nueva propiedad. Será similar a lo que hemos hecho en ejercicios anteriores, pero con las siguientes características:

- Esta página es solo para usuarios autenticados. Utilice `AuthService.checktoken()` al cargar la página. Si la verificación falla (en el bloque `catch`), redirija al usuario a `index.html`.
- **Cerrar sesión:**Implemente la funcionalidad del botón de cierre de sesión como se describe en el index.html sección.
- **Opcional (funciones de IA)**Agregar la funcionalidad necesaria al**#botón de traducción**y**#botón generar**elementos (clic):
  - El botón de traducción traducirá el texto de la descripción al inglés.**Detectar** primero el idioma actual y**solo traducir**cuando no esté ya en inglés (generará un error).
  - El botón "Generar" creará un título para la propiedad a partir de la descripción. Puede proporcionar un contexto como "Un título atractivo para vender rápidamente una propiedad inmobiliaria en el mercado". El título se generará solo si la descripción tiene al menos 20 caracteres; de lo contrario, se mostrará un mensaje de error al usuario (alerta).
  - Esta es una nueva API, así que es**No compatible con TypeScript**Aún así, puedes declarar las clases de la API como `any` y deshabilitar ESLint para este fragmento de código para evitar errores. También podrías colocar estas funciones en un archivo aparte y exportarlas.

```
/* eslint-disable */
declararconstanteDetector de idioma:cualquier;
declararconstanteTraductor:cualquier; declarar
constanteResumen:cualquier;

asíncronofunccióndetectarIdioma(texto:cadena):Promesa<cadena>{...}

asíncronofunccióntraducir:Promesa<cadena>(...}

asíncronofunccióresumir(texto:cadena):Promesa<cadena>{...} /*
eslint-enable */
```

PrecauciónEn el formulario hay un campo llamado «título». Esto dará problemas con TypeScript, por ejemplo, al escribir:**formulario.título**, interpretará que te refieres al atributo title del elemento <form>. Para evitarlo, puedes usar un objeto FormData para analizar el formulario o a través de la colección elements → **propertyForm.elements.namedItem("título")**.

Cuando todo vaya bien, redirige a index.html. Si algo falla, muestra un error en una alerta (o con la librería sweetalert2).

Página de detalles de la propiedad (property-detail.html)

Al hacer clic en la imagen o el título de una propiedad, debería redirigirnos aquí. Esta página muestra todos los detalles de una propiedad. Llamar a GET →http://SERVIDOR/propiedades/:id a Obtén el objeto de propiedad con toda su información.

- **Cargar datos:**
  - Obtén el ID de la propiedad de la cadena de consulta de la URL (p. ej., property-detail.html?id=16). Puedes usar location.search y URLSearchParams para extraer el ID.
    - Si no hay ningún id, redirige a index.html.
  - Obtén los datos de la propiedad del servidor y completa los elementos HTML correspondientes (#título-de-la-propiedad, #dirección-de-la-propiedad, etc.).**no**Utiliza una plantilla; manipula directamente los elementos del DOM.
  - El avatar y el nombre del vendedor también son enlaces a su página de perfil. Actualícelos con el ID de usuario correspondiente (ej.: profile.html?id=4).
- **Integración de mapas:**Una vez cargados los datos de la propiedad, inicialice el mapa centrado en la ubicación de la propiedad (latitud y longitud).
- **Después**Cargando los datos de la propiedad (no antes), compruebe si el usuario ha iniciado sesión.
  - Si has iniciado sesión**y**El objeto de propiedad cargado (almacenado en una variable global) tiene el atributo**calificado: falso**, muestra el formulario para añadir un nuevo comentario, actualiza el menú superior y añade la funcionalidad de clic al botón de cierre de sesión como hiciste en elindex.html página.
- **Opcional (Valoraciones de los usuarios)**:
  - **Calificaciones por estrellas:**Las calificaciones se representan con asteriscos (★ para relleno, ☆ para vacío). Crea una función auxiliar que reciba una calificación numérica y devuelva una cadena con la combinación correcta de asteriscos.
  - Recupera y muestra los comentarios existentes de la propiedad. Utiliza la plantilla proporcionada y**anteponer**Cada comentario se añade al contenedor para mostrar primero los más recientes.
  - La clasificación por estrellas se gestiona mediante CSS (no te preocupes por ello). Tu código solo necesita leer la calificación seleccionada y el texto del comentario, enviarlos al servidor y, una vez enviada la información correctamente, añadir el nuevo comentario al principio de la lista.
    - Junto con la calificación ingresada, el servidor devolverá la nueva calificación total de la propiedad (actualizándola en la página).
- **Calculadora de hipotecas:**Utilice este formulario para calcular la cuota mensual que el usuario pagaría bajo ciertas condiciones al obtener un préstamo hipotecario:
  - El primer campo de entrada (de solo lectura) debe rellenarse previamente con el precio del inmueble.
  - El resultado se mostrará en el**#pago mensual**párrafo. Elimine la clase oculta del párrafo.**#resultado hipotecario**elemento primero.

- Implemente la lógica de cálculo de la hipoteca: Después de restar el pago inicial del precio total, aplique la fórmula proporcionada al monto restante.

$$M = P\frac{r(1+r)^n}{(1+r)^n - 1}$$

Definitions of Variables

- M = Total monthly payment
- P = Principal loan amount (the initial amount borrowed)
- r = Monthly interest rate (decimal)
  - *Note: This is your annual interest rate divided by 12. For example, if your annual rate is 5%, $r = 0.05/12 = 0.0041667$.*
- n = Total number of payments (months)
  - *Note: If your loan is for 25 years, $n = 25 \times 12 = 300$.*

Página de perfil de usuario (profile.html)

Cuando hagamos clic en el enlace del icono de perfil, o en el avatar o el nombre de un usuario, iremos aquí.

Esta página muestra los datos del perfil del usuario. Puede mostrar el perfil del usuario actual o el de otro usuario si se proporciona un ID en la URL.

- Esta página es solo para usuarios autenticados. Llama al método `checktoken` al inicio. Si la llamada a `checktoken` falla (captura), redirige a `index.html`.
- Esta página mostrará información del usuario. Puede recibir un ID de usuario en la URL (profile.html?id=3). Si no recibe ningún ID, mostrará el perfil del usuario conectado. Llamada: GET →http://SERVIDOR/usuarios/yo O si recibe un ID, llame en su lugar a: OBTENER →http://SERVIDOR/usuarios/:id .
- Rellene los campos de la página y del formulario "Editar perfil" con el nombre, el correo electrónico y el avatar del usuario.

  La respuesta del servidor incluirá un campo booleano llamado**a mí**. Ejemplo:

```
{
    "usuario": {
        "identificación":48,
        "nombre":"Usuario de prueba, "correo
        electrónico":"test@test.com ",
        "avatar":"http://SERVIDOR/img/usuarios/1634033447718.jpg", "a mí":
        verdadero
    }
}
```

  - Si**a mí**es falso (estás viendo el perfil de otra persona), debes**esconder**Los botones "Editar perfil" y "Cambiar contraseña" ocultan la superposición de cambio de avatar (.avatar-image-overlay), y **desactivar**el campo de entrada de archivos (deshabilitado = verdadero).
- **Actualización de avatar:**
  - La imagen del usuario es una <label> para un input[type=file] oculto.
  - Cuando se selecciona un archivo, conviértalo a un**Base64**cadena e inmediatamente llamar al servidor para actualizar el avatar.
  - Si la operación se realiza correctamente, actualiza la imagen del avatar en la página sin recargarla por completo.
- **Alternancia de formulario:**
  - Cuando el usuario haga clic en "Editar perfil" o "Cambiar contraseña", oculte el botón y muestre el formulario correspondiente.
  - Si el formulario se envía correctamente**o**El usuario hace clic en "Cancelar", oculta el formulario y vuelve a mostrar el botón.
- **Actualizaciones de datos:**Cuando los datos del perfil del usuario se actualicen correctamente, refleje esos cambios en la página inmediatamente sin recargarla.
- **Comentarios de los usuarios**Proporcionar información al usuario (alertas, por ejemplo) cuando se produzca un error o la información se haya actualizado correctamente.
- **((Opcional) Enlace para ver propiedades:**Establezca el href del botón "Ver propiedades del usuario" en index.html?seller=USER_ID, donde USER_ID es el ID del usuario cuyo perfil se está visualizando.

Requisitos generales y notas importantes

- **Control de versiones (importante):**
  - Debes crear un**PRIVADO**Repositorio de GitHub para tu proyecto.
  - Comparta el repositorio con su profesor (los permisos de solo lectura son suficientes).
  - Debes hacer**Al menos 2 contribuciones significativas por semana**, a partir de la semana del 10 de noviembre.

- Los métodos de la clase Http utilizan tipos genéricos <T>. Esto significa que al llamar a un método, debe especificarse el tipo de la respuesta. En los métodos POST y PUT, también debe indicarse el tipo de los datos enviados al servidor. Existen interfaces para cada respuesta.**src/interfaces/responses.ts**archivo. Ejemplo:

  esto.#http.get<**PropiedadesRespuesta**>(`${SERVER}/properties`)

  esto.#http.post<**SinglePropertyResponse, PropertyInsert**>(`${SERVER}/properties`, propiedad)

- Utilice clases para encapsular la funcionalidad. Por ejemplo, la**Servicios de propiedades** La clase tendrá los métodos necesarios para llamar a servicios web que incluyen / **propiedades**/. Una clase llamada**Servicio de usuario**accederá a la /**usuarios**/ servicios y una clase llamada**Servicio de autenticación**accederá a la /**autorización**/ servicios.
- Por defecto, todos los elementos HTML se devuelven del DOM como HTMLElement genérico. Tendrás que convertirlos al elemento correcto si quieres usar propiedades específicas:

```
dejar imagen = documento.obtenerElementoPorId("imagen") como Elemento de imagen HTML; imagen.fuente
= ...;// DE ACUERDO
dejar valor = (forma.fecha como Elemento de entrada HTML).valor;// DE ACUERDO
```

- **Organización del código:**
  - Coloca todas las clases en una carpeta src/classes.
  - Coloca todas las interfaces en una carpeta src/interfaces. No las incluyas todas en un solo archivo; crea archivos temáticos (por ejemplo, user.interfaces.ts, property.interfaces.ts).
- **Importaciones:**Con Vite, ya no es necesario incluir las extensiones de archivo en las rutas de importación.
- **Seguridad de tipo: Evite el uso de cualquier**La configuración de ESLint proporcionada impondrá un tipado estricto.
- **Características de JS/TS:**Si desea utilizar características de lenguaje modernas, puede editar tsconfig.json y cambiar el destino de ES2022 a ES2024 o ESNext.
- **Async/Await:**Cuando se llama a una función asíncrona desde el ámbito superior sin usar `.then()` o `.catch()`, ESLint mostrará una advertencia. Para solucionarlo, simplemente añade un `.catch(console.error)` para registrar cualquier posible error.
- **HTML:**No modifique los archivos HTML proporcionados a menos que sea absolutamente necesario, y solo después de consultar con el profesor.
- **Manejo de errores:**Debes notificar al usuario (al menos con una alerta) cuando falle el envío de un formulario. En la página de perfil, también debes proporcionar un mensaje de éxito cuando...

La actualización se ha realizado correctamente.

- **Scripts de NPM:**
  - Cambia el nombre del script de desarrollo para empezar.
  - Crea un script de prueba que ejecute ESLint en todos los archivos de la carpeta src.
  - Agrega los siguientes scripts a tu archivo package.json:

```
"prearranque":""npm run format && npm run test",
"formato":"prettier --write --cache --parser typescript"src/**/*.ts""
```

Clases

Estas son las clases y métodos recomendados que podrías implementar. Por supuesto, **Puedes añadir más funcionalidades,**o hacer las cosas de otra manera (no tienes que usar **asíncrono**Por ejemplo):

```
exportar clase Servicio de autenticación {
    ...
    asíncrono acceso(Inicio de sesión de usuario):(Inicio de sesión de usuario):Promesa<
    vacío>{...} asíncrono registro(información de usuario):Usuario):Promesa<vacío>{...}
    asíncrono token de verificación():Promesa<vacío>{...}
    cerrar sesión():vacío{...}// Simplemente elimina el token del almacenamiento local
}

exportar clase Servicio de usuario {
    ...
    asíncrono obtenerPerfil(identificación¿:número):Promesa<Usuario>{...}
    asíncrono guardarPerfil(nombre:cadena,correo electrónico:cadena):Promesa<vacío>{...
    } asíncrono guardarAvatar(avatar:cadena):Promesa<cadena>{...} asíncrono
    guardarContraseña(contraseña:cadena):Promesa<vacío>{...}
}

exportar clase Servicio de provincias {
  ...
  asíncrono obtener provincias():Promesa<Provincia[]>{... } asíncrono
  obtenerCiudades(idProvincia:número):Promesa<Ciudad[]>{...}
}


exportar clase Servicio de propiedades {
    ...
    asíncrono obtenerPropiedades(urlSearch:Parámetros de búsqueda de URL):Promesa<PropiedadesRespuesta>{...} asíncrono
     obtenerPropiedadPorId(identificación:número):Promesa<Propiedad>{...} asíncrono insertarPropiedad(propiedad:Inserción de
    propiedad):Promesa<Propiedad>{...} asíncrono eliminarPropiedad(identificación:número):Promesa<vacío>{...}

    asíncrono agregar calificación(identificación:número,clasificación:(Insertar calificación):Promesa<Clasificación>{... }// Opcional
    asíncrono obtener calificaciones(identificación:número):Promesa<Clasificación[]>{... }// Opcional
}
```

Desglose de calificaciones

La calificación final se calculará según estos criterios (lo que ya se haya implementado en los ejercicios anteriores no se tendrá en cuenta):

- Página de inicio de sesión:**1 punto**
- Página de registro:**1 punto**
- Página principal de listados (index.html):**2,5 puntos**
- Agregar página de propiedad:**1 punto**
- Página de detalles de la propiedad:**2,5 puntos**
- Página de perfil de usuario:**2 puntos**

Aunque todo funcione correctamente, se pueden descontar puntos en cada sección si el código no se considera adecuado. Formatee y organice bien su código y evite repeticiones innecesarias.

Deducciones de puntos / Penalizaciones

- **Incumplimiento de los requisitos generales:**-De 0,25 a -0,5 puntos por regla.
- **Configuración incorrecta del repositorio o commits faltantes:**-0,25 a -0,5 puntos.
- **Advertencias de ESLint en la versión final:**-0,25 a -0,5 puntos.
- **Errores de ESLint en la versión final:**-De 0,5 a -1 punto.
- **Mala calidad del código:**Se podrá deducir hasta -1 punto por cuestiones como:
  - Código espagueti (falta de estructura, todo en un solo archivo).
  - Variables/funciones con nombres poco acertados.
  - Utilizar código básico (por ejemplo, bucles for) cuando se han enseñado alternativas más avanzadas y apropiadas (por ejemplo, filter, map).
  - Se recomienda el uso excesivo de comentarios en línea (se aconseja incluir comentarios de documentación al estilo JSDoc encima de las funciones/clases).
- **Funcionalidad:**La aplicación debe estar operativa. Si una función básica como el inicio de sesión no funciona, las funciones dependientes (como el perfil de usuario) no podrán evaluarse y se considerarán incompletas.

Contenido opcional (máximo 2 puntos extra)

Estas 3 extensiones aumentarán la calificación final del ejercicio, pero solo si la calificación final sin el contenido opcional es**7 o superior**:

- **(0,5 puntos extra)**Utilice SweetAlert o Bootstrap para mostrar comentarios al usuario en los formularios, preguntándole cuándo debe eliminar una propiedad, etc.
- **(0,5 puntos extra)**Implementa la API de IA de Chrome en la página property-detail.html para traducir la descripción de otros idiomas al inglés y generar un título a partir de la descripción.
  - Ten en cuenta que a veces el resultado puede tardar unos segundos.
- **(0,5 puntos extra)**Implemente la lógica para calificar y comentar una propiedad (property-detail.html).
- **(0,5 puntos extra)**Agrega un enlace en el perfil del usuario para filtrar las propiedades creadas por el usuario actual. Este enlace redirigirá a la página index.html, incluyendo el ID de este usuario, de la siguiente manera:**index.html?vendedor=14**
  - Cuando el ID del vendedor esté presente en la URL, muestre solo las propiedades creadas por este usuario, y**incluir el nombre**del usuario en el texto que muestra los filtros aplicados actualmente. Por ejemplo:

    Vendedor: Usuario de prueba 2. Provincia: Alicante. Búsqueda: Moderno.
- **(1 punto extra)**Aprende sobreCropperJS biblioteca para recortar imágenes y usarla para La foto de la propiedad (new-property.html) y el avatar del usuario (regístrese y actualice su perfil).
  - Descarga el**cropperjs**dependencia y su**@tipos/cropperjs**Definiciones de TypeScript. Inclúyelo en tu código con**importar Cropper desde 'cropperjs'**.
  - La imagen de la propiedad tendrá una relación de aspecto de 16:10 y un ancho de 1024 píxeles. La imagen del avatar del usuario tendrá una relación de aspecto de 1:1 y un ancho de 200 píxeles.
  - Si implementa esta función, puede modificar el HTML para incluir un área para recortar la imagen (solo visible al recortar).