Vamos a realizar una página web para encontrar pareja. Será una web con pocas funcionalidades, más bien un sistema de mensajería aunque sí que pedirá preferencias y mostrará personas por afinidad. También deberemos idear algún sistema en el que si dos personas muestran interés se establezcan como amigos y puedan verse el uno al otro cuando ingresen en el sistema.

La aplicación constará de:

- Una página de inicio que permitirá entrar en el sistema. En esta página de inicio se permitirá también enlazar a la página de registro y a la página de restaurar contraseña. La página de inicio tendrá una Integración con reCaptcha de Google en el login (aleatorio, es decir, solo tendremos que realizarlo de vez en cuando, al azar).
- La página de registro permitirá dar los datos personales (incluyendo *email* que servirá como usuario de inicio de sesión). Se deberá implementar para este formulario un captcha matemático en JS.
- Una vez dentro del sistema como usuario estándar tendremos las siguientes funcionalidades:
    - Ver usuarios de mi preferencia lo más afines posible a mis gustos.
    - Permitir establecer qué usuarios me gustan (o dejan de gustar) con un simple click. En caso de que sea recíproco, nos convertiremos automáticamente en amigos.
    - Deberíamos poder listar los amigos que tenemos. El formato será libre.
    - Enviar mensajes a otros usuarios (amigos o no).
    - Ver si algunos de los usuarios que me gustan (y yo les gusto) están en línea.
    - La primera vez que entramos al sistema nos pedirá que rellenemos un formulario con nuestras preferencias. Estas preferencias se podrán editar siempre, así como nuestro perfil (foto, nick, clave de acceso…).
- Como administrador podemos, con un CRUD, gestionar a los usuarios: altas, bajas, modificaciones, activaciones… los usuarios estarán desactivados por defecto hasta que los active el administrador. También se podrán crear a otros administradores (o quitarles los privilegios a los actuales).
- Los usuarios podrán enviar mensajes a otros usuarios y ver los mensajes que otros usuarios les han enviado. A estos mensajes se les podrán adjuntar archivos, lo obligatorio es que se pueda adjuntar al menos uno; como optativo, pero muy valorado, es que permita más de un archivo adjunto.
- Las preferencias que un usuario podrá poner son las siguientes:
    - Relación seria → indicarán si su finalidad es una relación seria o esporádica.
    - Deportivos → valor numérico que indica, de 0 a 100, su gusto por los deportes.
    - Artísticos → valor que indica su inquietud artística.
    - Políticos → interés por la política, valorando de 0 a 100.
    - Tiene/Quiere hijos o no → Indicando si tiene y/o si quiere hijos.

  - Interés en: hombres, mujeres o ambos.

  Todos estos valores se tendrán en cuenta a la hora de mostrar a posibles parejas candidatas en la pantalla principal del usuario.
- Los usuarios verán, por lo tanto, a los candidatos más afines a sus propios valores. La forma de hacer esta parte es libre, se valora el algoritmo ideado.
- Cuando un usuario se conecte se habilitará, de algún modo, un sistema que indique los amigos que tiene y que están conectados en ese momento, así como el número total de usuarios conectados (amigos o no).
- Los formularios en general serán validados con html5/API de JS personalizando los mensajes de error y el css asociado.
- Las páginas de la aplicación deberán ser coherentes en estilo, con cabecera y pie de página. En la cabecera sería bastante útil y bien valorado que siempre esté la información básica del usuario conectado y las opciones que permitan editar el perfil de los mismos, tal y como se indicó anteriormente. En el footer podemos poner el nombre del programa y del creador (vuestro nombre).

Se tendrá en cuenta:

- Programación clara y ajustada al MVC.
- Diseño de la BD que recoja toda la información de forma correcta.
- Que lo programado se ajuste a lo pedido.
- Una planificación de tareas coherente.
- Que los sprints de desarrollo cumplan lo planificado.
- El proyecto deberá estar sincronizado con un repositorio Git y tener una rama, mínimo, de desarrollo. Después de cada tarea planificada se debería hacer un merge a máster (estaría bien que se usaran los board de GitKraken investigando su sincronización con GitHub).

ANEXO: METODOLOGÍA SCRUM

SCRUM es una metodología ágil en la que se aplican regularmente un conjunto de buenas prácticas para trabajar en equipo y obtener el mejor resultado en el proyecto.

En SCRUM se realizan entregas parciales y regulares del producto final, priorizándolas según el beneficio que aportan al receptor del proyecto (cliente).

Los puntos destacables de esta metodología de trabajo son:

ROLES:

- **Product Owner:** se encarga de analizar qué es lo que quiere el cliente final en su proyecto y lo detalla en una lista priorizada (**product backlog).**
- **Scrum Master:** organiza y facilita el trabajo. Defiende la metodología a llevar a cabo ante el product owner.
- **Team:** equipo de desarrolladores. Analizan las tareas del **product backlog**, detallan los ítems que van a contener, su prioridad y el tiempo que les va a llevar realizarlas.

Una persona puede tener más de un rol. Por ejemplo, el scrum master puede formar parte del equipo de desarrollo.

FUNCIONAMIENTO:

El trabajo se organiza en **sprints**. Un sprint tiene un tiempo acotado (por ejemplo 2 semanas) y en él se incluyen las tareas (historias) establecidas en el **product backlog.**

El sprint comienza con una planificación **(planning)**: se reúne el **team, product owner y scrum master.** Se analiza cada tarea establecida por el product owner en el backlog. Se prioriza, determina la dificultad técnica por el equipo de desarrollo y se establece el tiempo que tardará en completarse. Una vez analizada, se decide si esa tarea **"cabe"** en dicho sprint, en cuyo caso, el team se **COMPROMETE** a realizarla en dicho sprint.

Cuando finaliza el sprint, se realiza una **retrospectiva** analizando lo que se ha hecho bien y lo que se ha hecho mal, para proponer mejoras al respecto. Después se realiza una **demo** de las historias del sprint que se han **completado,** ante el **product owner (**y a veces ante el cliente final).

Cada día del sprint se hará una pequeña reunión denominada **daily** (tiene que ser corta (10') y se obliga a estar de pie para que no se alargue), en la que cada miembro del equipo de desarrollo cuenta en lo que ha trabajado, las dificultades y si necesita ayuda por parte del resto del equipo.

Cuando comienza un sprint, existe un tablero donde se encuentran todas las tareas sin asignar pendientes del sprint (**backlog**), y que cada desarrollador irá escogiendo (teniendo en cuenta al equipo) y asignando según sus preferencias, ritmo de trabajo, etc.