Travel Agency App

```
https://api-de-travels.herokuapp.com/destinations
```

1. Crear las clases necesarias
    1. Una para la App
    2. Otra para cada Destination
2. En el `main.js` creamos una nueva instancia de la App
3. Añadimos las propiedades básicas a la class Destination:
    1. id
    2. city
    3. country
    4. photo
    5. visited
    6. favorite
4. Vamos a crear el markup de cada Destination dentro de la propia class
    1. Creamos un `<article>` en global
    2. En un método aparte, le damos las classes, el contenido, etc
    3. El icono de corazón tiene que estar relleno o no dependiendo de su propiedad favorite
5. Crear el array vacío que guardará todas las destinaciones -> `allDests`
6. `getAllDestinations()`
    1. Hacemos un `fetch` con todas las destinaciones
    2. Coger el array que hemos recibido y MODIFICARLO creando un nuevo array que contendrá los mismo objetos pero convertidos a `Destination`
    3. Hacemos que este array sea nuestro `allDests`
    4. `console.log(allDests)` debería dar un array con 8 objetos de tipo Destination
    5. Una vez tengamos nuestro array `allDests` correcto, llamamos a la función `printDestinations()` que crearemos ahora
7. `printDestinations(array)`
    1. Cazamos la caja donde iran todos los destinations
    2. Que borre todos los destinations que hay en pantalla
    3. Que simplemente coja el array que le pasemos y lo recorra con un bucle y imprimimos en consola cada objeto.

4. En lugar de imprimir en consola nada, cogemos cada objeto, que solo por ser de tipo Destination ya tiene un `<article>` asociado, y hacemos un `append` dentro de la lista de destinaciones.

8. Añadir un nuevo `Destination`

    1. Cazamos el formulario y escuchamos su evento submit -> `handleAddDestination()`
    2. Evitamos comportamiento por defecto
    3. Guardamos valores de los inputs. Si algun campo está vacío le ponemos algún error visible. (Tenemos ya una función que hace eso y que la podemos meter en `helpers`)
    4. Creamos la función `addDestination()` que cogerá los valores de los inputs y enviará los datos a la API

        1. Creamos un objeto con los nuevos datos para enviarselos al servidor de la API
        2. Creamos el objeto que contendrá las opciones de la petición. como `method` usaremos 'POST', como `headers` pondremos un objeto con la propiedad `'Content-Type'` y con el valor `'application/json;'` y por último pondremos un `body` que será el objeto de la nueva destinación que vamos a crear convertida a JSON utilizando `JSON.stringify()`

9. `resetForm()`

    1. Creamos la función
    2. hacemos reset del formulario vaciando los elementos
    3. Usando `this.form.elements` que nos da un array con todos los elementos de un formulario, aprovechamos para recorrerlo y quitarle a cada uno la clase `is-valid` o la que sea que hayamos puesto al validar el formulario.

10. `changeFavorite()`

    1. Creamos la función dentro de `App`, que es la que se encarga de hacer las peticiones al servidor
    2. Pero para poder escuchar el clic al corazón que está creado dentro de `Destination` tenemos 2 opciones:

        1. Hacemos que el botón del corazón esté disponible globalmente dentro de `Destination` para poder acceder a él desde `App`.
        2. cuando creamos un `new Destination()` podemos pasarle la referencia a la función `this.changeFavorite()` para que lo reciba el constructor y poder ejecutarlo desde la propia `Destination`, pero esto generaría un problema con el `this` y tendriamos que "bindear" el `this` a la clase `App`.

    3. Haciendo la primera versión que es la más facil, simplemente creamos un elemento `<i>` global dentro de `Destination`

4. Luego generamos en una función aparte llamada `createHeartMarkup()` el contenido, clases, estilos que hagan falta para que ese botón funcione.
5. Lo metemos dentro del markup de nuestro `Destination` en el lugar que le tocaba.
6. En el punto dentro de nuestra `App` en el que podemos acceder a los `new Destination()` podemos ahora escuchar el click a su botón heart porque ya lo tienen en global, así que dentro de `getAllDestinations()` justo después de generar una nueva instancia de cada objeto, añadimos el "listener" del click y llamamos a la función `this.changeFavorite()`
7. A la función le pasamos el objeto entero para tener toda la info del objeto al que le queremos cambiar su estado. `this.changeFavorite(newDest)`
8. Hacer la petición al servidor con el método `PATCH` utilizando la URL '/destinations/:id', por ejemplo `..../destinations/3`
9. Como objeto a enviar será simplemente con las propiedades que queramos cambiar, en este caso simplemente habrá que enviar un `body` con el objeto `{ favorite: lo contrario de lo que haya }`
10. Volvemos a imprimir todas las destinaciones si todo ha ido bien

11. `deleteDestination()`

    1. Hacemos exactamente lo mismo que con el botón del corazón hasta que podamos ver en consola el objeto al que hemos hecho click cuando le damos a la basura.
    2. Una vez todo hecho, haremos la petición al servidor con el método `DELETE` simplemente diciendo la URL del elemento que queremos borrar -> `.../destinations/5`
    3. Solo con hacer el `fetch` en la URL que toca y con el método `DELETE` ya nos borra el elemento.
    4. Después solo tendremos que volver a imprimir las destinaciones.

12. Editar la destinación

    1. Primero hacemos lo mismo con el icono de editar, igual que con el corazón y con la papelera.
    2. Creamos una función que se encargará de mostrar un pop-up con un formulario para editar. Ya que estamos usando bootstrap, vamos a utilizar el `Modal` que nos viene.
    3. Primero creamos el markup del modal según su documentación, podemos hacerlo en la misma función o en otra por separado.
    4. Tenemos que crear una instancia nueva del objeto `Modal` que nos trae bootstrap. Si hemos instalado bootstrap como paquete npm, podemos importar arriba el Modal. `import {Modal} from 'bootstrap'`, y si no lo hemos instalado, cogemos el `<script>` que nos da bootstrap y lo ponemos en el `index.html`
    5. Creamos la instancia del modal con la forma que escojáis y le pasáis como parámetro el objeto que hemos creado como markup del modal `const bsModal = new Modal(myModalHTML)`
    6. Este objeto nuevo nos viene con unos métodos `.show()` y `.hide()` para mostrar y esconder el modal.

7. Así que llamamos a `bsModal.show()` y nos abre el modal.
8. Escuchamos el botón de actualizar y cuando hagamos clic volvemos a capturar los valores del formulario de eición. Podemos volver a revisar que no estén vacíos y cuando todo esté correcto, enviar los 3 valores a una nueva función llamada `updateDestination()` que enviará la info a la API. También tenemos que pasarle todo el objeto Destination, ya que necesitaremos el `id`, el `favorite` y el `visited`
9. Por último, gracias a que bootstrap ha creado unos eventos que solo los Modales tienen, podemos escuchar el evento 'hidden.bs.modal' y borrar el modal del DOM cuando se haya escondido

```
myModalHTML.addEventListener('hidden.bs.modal', () ⇒
myModalHTML.remove())
```

13. Actualizar la destinación

    1. Igual que con los demás métodos, necesitamos el objeto que vamos a enviar a la API con los datos actualizados.
    2. Creamos un objeto de opciones con el método `PUT`, los headers de siempre y el body haciendo un `JSON.stringify()` con el objeto a enviar.
    3. Revisamos que todo haya ido bien, si no mandamos un error y si todo va bien volvemos a pedir las destinaciones.

A partir de aquí, podéis añadir lo que queráis para complicar la cosa.