Place Keeper

ES6 & HTML5

General

Our app consists of three main sections:

- **<section class="home">** – the App's home page with navigation buttons to the show the other sections.
- **<section class="user-pref">** – displays a `<form>` for collecting user preferences. These preferences determine how various parts of the app are displayed.
- **<section class="map">** – displays a list of places saved by the user and a map.

Guidelines

Remember to use the <section, nav, main, aside, header, footer> semantic elements

Use ES6 throughout your code: destructering, arrow functions, default parameter values, let, const, etc.

Use the MVC pattern to shape your app, you should have the following services:

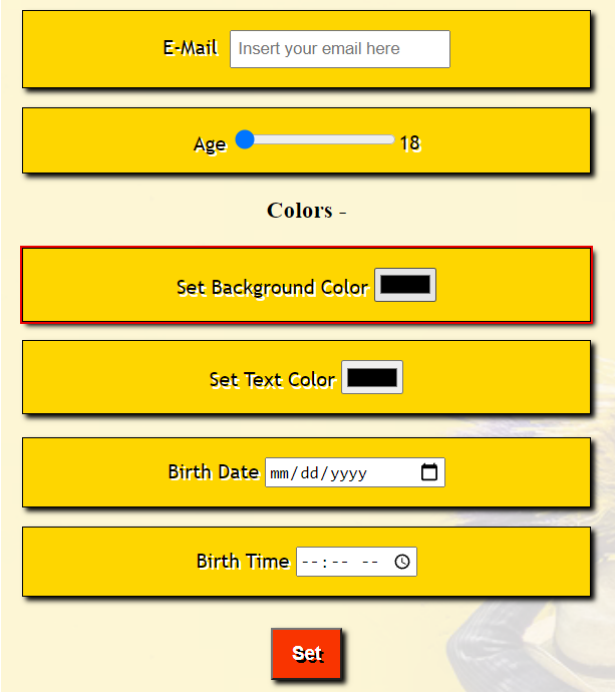
- `util.service` – *general utility functions.*
- `user.service` – *manages saving and reading the user's prefs* ***synchronously***
- `place.service` – *manages the place entity CRUDL* ***asynchronously*** *using the* ***async*-local-storage**
- `app.controller` – *connects the view to the services, handles our map and is the only script loaded to our document. Don't forget *type="module"**

Home section

This is a simple home page with some graphics and a welcome message, something like: *Find your way back to your best places*

Add navigation buttons to the other (two) pages: user-pref and map
These buttons will hide the home section and show the chosen one.

User Preferences section



Here we will use a <form> to get the user settings and save them to localStorage. (No need to use async-local-storage here) We save it in the storage so that when a user revisits our page we will save his preferences. (or when refreshing)

The **user** object will finally look like that

The user data model :

```
const user = {

    email : '',
    txtColor : '',
    bgColor : '',
    age : '',
    birthDate: '',
    birthTime: ''

}
```

Master tip: it's best to **start simple** with the first two properties – `email` and `txtColor`

The application should use the colors provided by the user and show the home section accordingly.

Step 1 - Colors

Use HTML5 color <input> to let the user set its background and text color of the pages.
TIP: use: *userService.save(userData)*

Step 2 – Date and Time

Use HTML5 *date* and *time* <input>s to let the user set his exact birth time, In the homepage render the user's birthtime

Step 3 – Wrap in a form

Put those inputs in a <form>, and on submit, use a service to keep them in a localstorage object: userData

TIP: you will need event.preventDefault in the onsubmit event handler.

Step 4 – Add some more inputs

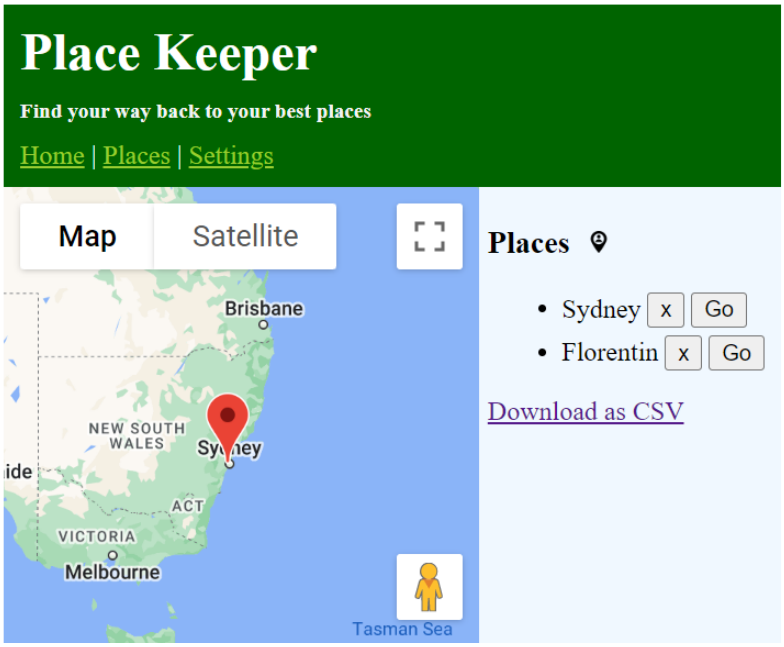
1. Add a *required* email <input>
2. Add a *range* <input> to let the user select his age: 18->120

Map section

Here we will show a **map** and allow the user to manage his places.

Tips:

- Use the 'create Google Api' doc to create an API key and secure it.
- This exercise involves self learning and reading documentation for a new API



Step 1 – places list

Show the list and allow the user to remove a place.
Use a placeService that manages the place entity, a place object looks like that:

```
{id: '1p2', lat: 32.1416, lng: 34.831213, name: 'Pukis house'}
```

- Start from rendering 2 places on the page
- Setup your controller functions
  - `function onInit() {}`
  - `function renderPlaces() {}`
  - `function onRemovePlace(placeId) {}`
- Setup the place.service
  - `function getPlaces() {}`
  - `function removePlace(placeId) {}`
  - `function addPlace(name, lat, lng, zoom) {}`
  - `function getPlaceById(placeId) {}`
  - `function _createPlace(name, lat, lng, zoom) {}`
  - `function _createPlaces() {}`
- Render the list and check that your functions work

Step 2 – Show a map

1. Generate your Google Maps API key
   (see directions in a separate doc)
2. Show a map centered at **Eilat**
   - Copy the code needed for showing a simple map
   - you can use an online tool (such as this) for getting the lat-lng for Eilat
3. When a user clicks on the map, the user is prompted to enter a name and a new place is saved to storage, here is some code to put you in the right direction (inside initMap):

```
gMap.addListener('click',async ev => {
        const name = prompt('Place name?', 'Place 1')
        const lat = ev.latLng.lat()
        const lng = ev.latLng.lng()
        await placeService.addPlace(name, lat, lng, gMap.getZoom())
        renderPlaces()
})
```

4. When a user clicks a button to go to a place, the map is moved and zoomed on the selected place

```
async function onPanToPlace(placeId) {
    const place = await placeService.getPlaceById(placeId)
    gMap.setCenter({ lat: place.lat, lng: place.lng})
    gMap.setZoom(place.zoom)
}
```

Step 3 – User location

when user clicks the  button, get his current location and center the map accordingly.

Step 4 – Markers

When the map is ready, and also when places are added / removed, we call the renderMarkers function:

```
async function renderMarkers() {
    const places = await placeService.getPlaces()
    // remove previous markers
    gMarkers.forEach(marker => marker.setMap(null))
    // every place is creating a marker
    gMarkers = places.map(place => {
        return new google.maps.Marker({
            position: place,
            map: gMap,
            title: place.name
        })
    })
}
```

Step 5 - Finalize the app

1. Add navigation buttons to all pages.
2. Let the user download a CSV of the places

Bonuses

1. Replace the prompt for new place name with a nice modal
2. In the user-prefs
    - add another input: gender, that is based on a datalist with the options: Male, Female, Other
    - Add custom validation: validate the provided user age matches the provided birth year