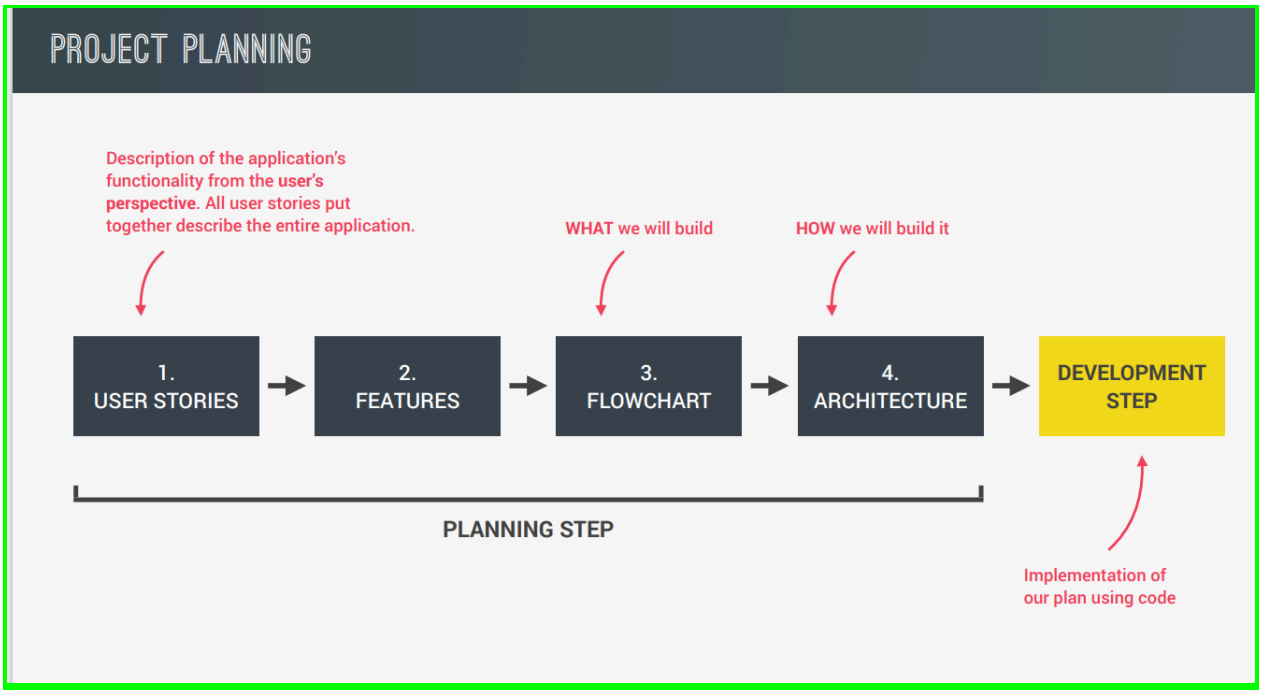
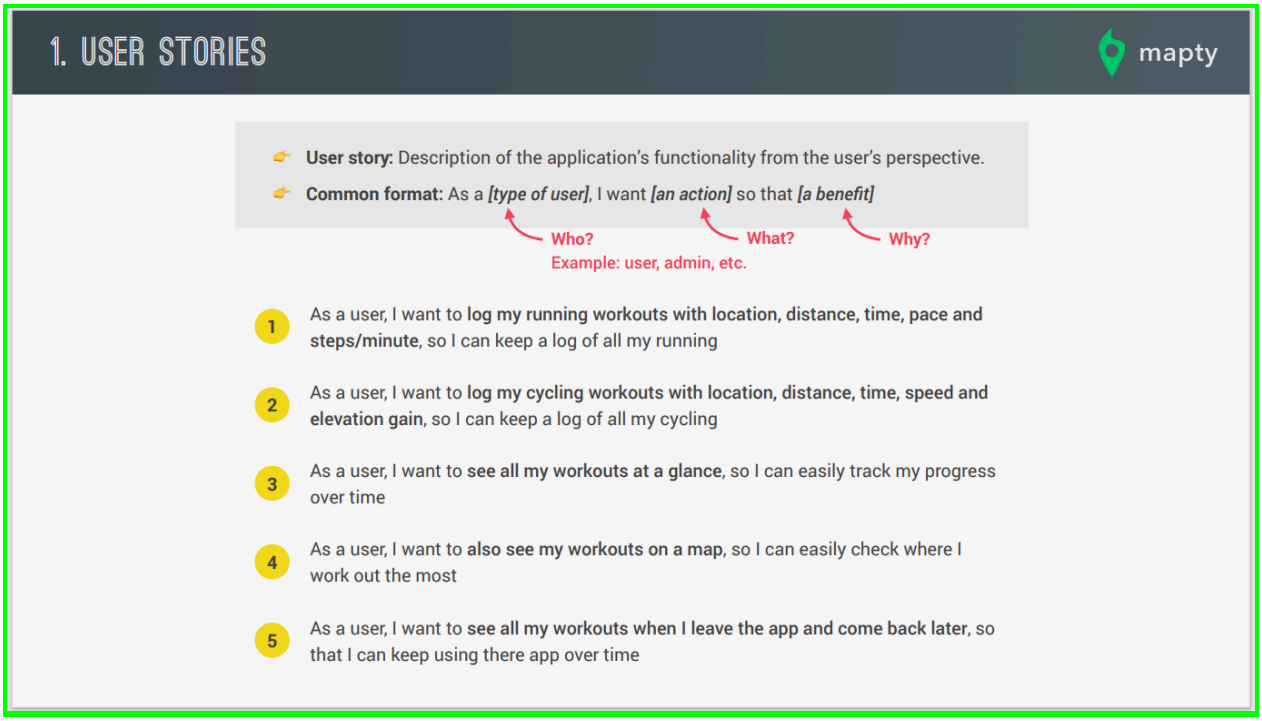
# Mapty App (OOP,Geolocation ,External Libraries(Leaflet Library,OpenStreetMap ): and More)

## Project Planning



## 2. FEATURES

mapty

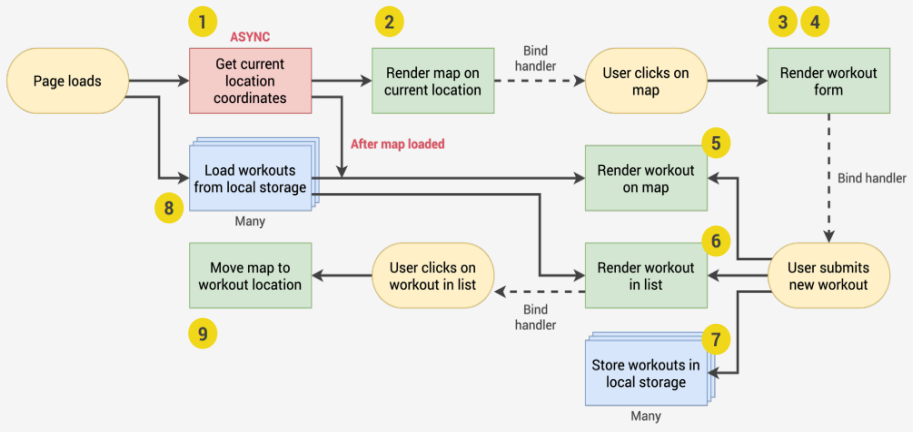
| USER STORIES | FEATURES |
|---|---|
| 1 Log my **running** workouts with location, distance, time, pace and steps/minute | Map where user clicks to add new workout (best way to get location coordinates)<br>Geolocation to display map at current location (more user friendly)<br>Form to input distance, time, pace, steps/minute |
| 2 Log my **cycling** workouts with location, distance, time, speed and elevation gain | Form to input distance, time, speed, elevation gain |
| 3 See all my workouts at a glance | Display all workouts in a list |
| 4 See my workouts on a map | Display all workouts on the map |
| 5 See all my workouts when I leave the app and come back later | Store workout data in the browser using local storage API<br>On page load, read the saved data from local storage and display |

## 3. FLOWCHART

mapty

### FEATURES

1. Geolocation to display map at current location
2. Map where user clicks to add new workout
3. Form to input distance, time, pace, steps/minute
4. Form to input distance, time, speed, elevation gain
5. Display workouts in a list
6. Display workouts on the map
7. Store workout data in the browser
8. On page load, read the saved data and display
9. Move map to workout location on click — Added later




☝ In the real-world, you don't have to come with the final flowchart right in the planning phase. It's normal that it changes throughout implementation!

**What is Geolocation API**

The **Geolocation API** is a web API that allows web applications to access the geographical location of a device (like a computer or smartphone) through the browser or application. It can be used to determine the user's latitude, longitude, altitude, speed, and heading, based on various location sources such as GPS, IP address, Wi-Fi, or Bluetooth.

Here's how it generally works:

- The user is prompted to grant permission to share their location with the application or website.
- Once granted, the API uses the available hardware (like GPS) or other methods (IP address, Wi-Fi) to get the coordinates.
- The API provides these coordinates (latitude, longitude) and other relevant details, which can be used to display maps, local information, or offer location-based services.

**Key Methods in Geolocation API:**

**getCurrentPosition()**: Retrieves the current location of the device.

- Used to retrieve the current geographical position of the device.
- This method takes a success callback function (which is called when the location is successfully fetched) and an optional error callback function (for handling errors).

  navigator.geolocation.getCurrentPosition(successCallback, errorCallback, options);

  **Parameters**:

  a. successCallback: A function that is called with a position object when the location is successfully retrieved.
  b. errorCallback: A function that is called with an error if the location cannot be determined.
  c. options (optional): A set of options to configure the location request (e.g., accuracy, timeout, etc.).

**watchPosition()**: Continuously tracks the device's location as it changes.Returns the current position of the user and continues to return updated position as the user moves (like the GPS in a car).

- Used to continuously monitor the device's position as it changes over time. It returns a watch ID that can be used to stop the tracking with clearWatch().

var watchID = navigator.geolocation.watchPosition(successCallback, errorCallback, options);

- **Parameters**:
  - successCallback: A function that is called whenever the position changes.
  - errorCallback: A function that is called when there's an error retrieving the position.
  - options (optional): A set of options to configure the location request (similar to getCurrentPosition()).

```
var watchID = navigator.geolocation.watchPosition(function(position) {

  console.log("Updated Latitude: " + position.coords.latitude);

  console.log("Updated Longitude: " + position.coords.longitude);

}, function(error) {

  console.log("Error: " + error.message);

});
```

**clearWatch()**: Stops tracking the device's location. Used to stop tracking the device's position that was started with watchPosition().

navigator.geolocation.clearWatch(watchID);

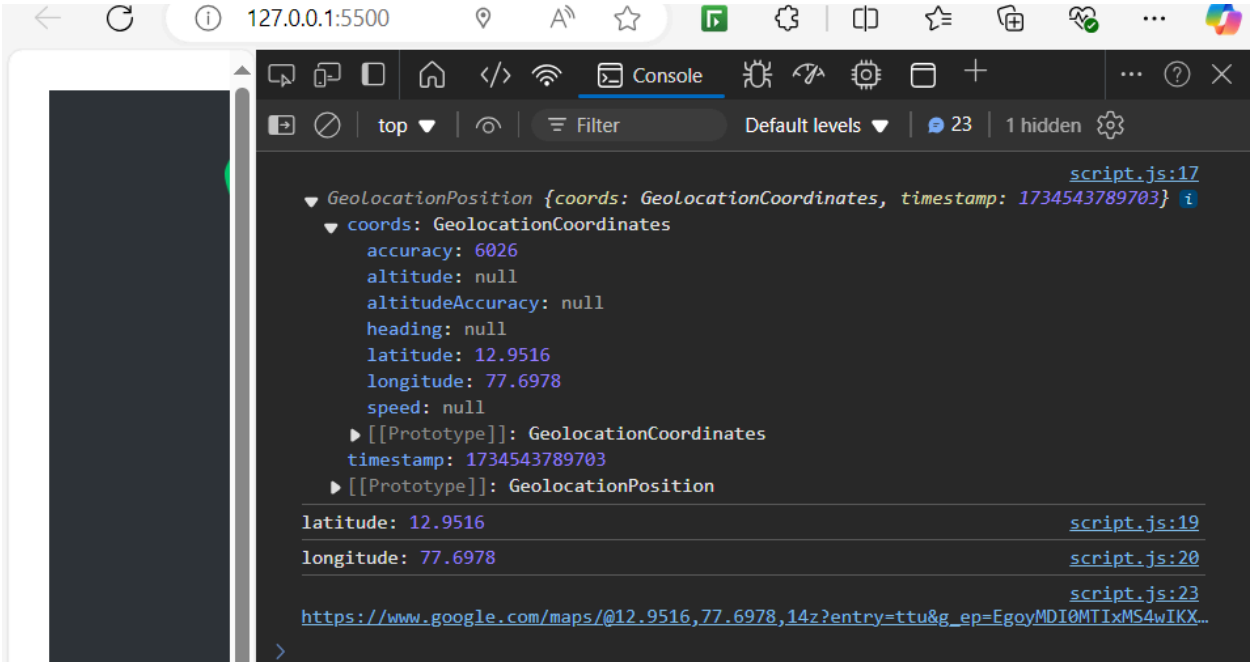
- **Parameters**:
  - watchID: The ID returned by watchPosition(), which you use to stop watching the location.
- **Example**:
  navigator.geolocation.clearWatch(watchID);

---

**EX1:**

```
if (navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(
```

```
    function (position) {
      console.log(position); // Logs the entire position object
      const { latitude, longitude } = position.coords; // Destructuring to get latitude and longitude
      console.log('latitude:', latitude); // Logs the latitude
      console.log('longitude:', longitude); // Logs the longitude

      // Generates a Google Maps link to view the location
      console.log(

`https://www.google.com/maps/@${latitude},${longitude},14z?entry=ttu&g_ep=EgoyMDI0MTIxMS4wIKXMDSoASAFQAw%3D%3D`
      );
    },
    function (error) {
      console.log('Error: ' + error.message); // Logs the error message if geolocation fails
      alert("Couldn't get your location"); // Alerts the user if location access fails
    }
  );
} else {
  console.log('Geolocation is not supported by this browser.'); // Logs if geolocation is not supported
}
```




Displaying a Map Using **Leaflet Library**


**Leaflet Library :** Leaflet is the leading open-source JavaScript library for mobile-friendly interactive maps. Leaflet is designed with *simplicity*, *performance* and *usability* in mind. It works efficiently across all major desktop and mobile platforms, can be extended with lots of plugins, has a beautiful, easy to use and well-documented API and a simple, readable source code that is a joy to contribute to.

**Leaflet** is an open-source JavaScript library used for building interactive maps. It is designed to be lightweight, simple, and easy to use, while providing powerful functionality for working with maps in web applications. Leaflet is widely used for displaying maps, adding markers, overlays, and handling geospatial data in a user-friendly way.

**Key Features of Leaflet:**

1. **Lightweight**: It's relatively small in size, making it suitable for both mobile and desktop applications.
2. **Interactive Maps**: You can easily create interactive maps with zoom, pan, and layers.
3. **Customizable**: Leaflet supports customization of maps, markers, pop ups, and other map elements.
4. **Easy to Use**: The library is designed to have a simple API that's easy to learn and integrate into web applications.
5. **Extensibility**: It provides plugins for adding advanced features like heatmaps, vector layers, and more.

**Using Leaflet library code:**

i) Navigate **Download** option in the official site page of https://leafletjs.com/download.html

**Copy below piece of code and paste it in our index.html**

**Using a Hosted Version of Leaflet**

The latest stable Leaflet release is available on several CDN's — to start using it straight away, place this in the head of your HTML code:

```
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css"
integrity="sha256-p4NxAoJBhIIN+hmNHrzRCf9tD/miZyoHS5obTRR9BMY=" crossorigin=""
/>

<script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"
integrity="sha256-20nQCchB9co0qIjJZRGuk2/Z9VM+kNiyxNV1lvTlZBo="
crossorigin=""></script>
```

```
<link
  rel="stylesheet"
  href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css"
  integrity="sha256-p4NxAoJBhIIN+hmNHrzRCf9tD/miZyoHS5obTRR9BMY="
  crossorigin=""
/>
<script
  defer
  src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"
  integrity="sha256-20nQCchB9co0qIjJZRGuk2/Z9VM+kNiyxNV1lvTlZBo="
  crossorigin=""
></script>
```

ii) Navigate **Overview** option in the official site page of https://leafletjs.com/download.html

Copy below piece of code inside the successCallback function in getCurrentPosition .

For this need a empty div tag `<div id="map"></div>` in index .html as below we are using map.

we create a map in the 'map' div, add tiles of our choice, and then add a marker with some text in a popup:

```
var map = L.map('map').setView([51.505, -0.09], 13);

L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {

   attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'

}).addTo(map);

L.marker([51.5, -0.09]).addTo(map)

   .bindPopup('A pretty CSS popup.<br> Easily customizable.')

   .openPopup();
```

```
if (navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(
    function (position) {
      console.log(position); // Logs the entire position object
      const { latitude, longitude } = position.coords; // Destructuring to get latitude and longitude
      console.log('latitude:', latitude); // Logs the latitude
      console.log('longitude:', longitude); // Logs the longitude
      const coords = [latitude, longitude];
      const map = L.map('map').setView(coords, 13);

      L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution:
          '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors',
      }).addTo(map);

      L.marker(coords)
        .addTo(map)
        .bindPopup('A pretty CSS popup.<br> Easily customizable.')
        .openPopup();
```

## Code explanation

**L** is the global namespace object provided by the **Leaflet** library.

It contains all the methods and classes that Leaflet offers, such as:

- **L.map**: For creating a map.
- **L.tileLayer**: For adding tile layers to the map.
- **L.marker**: For adding markers to the map.

In short, L is the entry point for using Leaflet's API.

**const coords = [latitude, longitude];**

- Creates an array of coords with two elements: latitude and longitude.
- These are the geographic coordinates (latitude and longitude) where the map will be centered.
- Example: If latitude = 51.505 and longitude = -0.09, then coords will be [51.505, -0.09]

**const map = L.map('map').setView(coords, 13);**

- **L.map('map'):** Creates a Leaflet map object.

'map' is the id of the HTML element where the map will be rendered. Example:

- **.setView(coords, 13):** Centers the map on the coordinates provided in coords ([latitude, longitude]). 13 is the zoom level of the map.
  - Higher values (e.g., 16) zoom in closer.
  - Lower values (e.g., 5) zoom out for a wider view.

**L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {...})**

- **L.tileLayer()**: Adds a tile layer to the map. A tile layer provides the graphical map elements (roads, buildings, terrain, etc.).
- **URL Template (https://tile.openstreetmap.org/{z}/{x}/{y}.png)**:
  - Specifies where to fetch the tiles from.
  - {z}: Zoom level.
  - {x} and {y}: Tile coordinates based on the location and zoom level.
  - In this case, the tiles are fetched from **OpenStreetMap**.
- **Attribution**: Adds text at the bottom-right of the map to credit the tile source (OpenStreetMap in this case).
  © OpenStreetMap contributors

OpenStreetMap is the free wiki world map. OpenStreetMap is an initiative to create and provide free geographic data, such as roads, buildings, addresses, shops and businesses, points of interest, railways, trails, transit, land use and natural features, and much more.

some alternative tile providers for OpenStreetMap:

https://tile.openstreetmap.fr/hot/{z}/{x}/{y}.png refers to a specific tile server providing map tiles for the **Humanitarian OpenStreetMap Team (HOT)** style

1. **Carto**:
    - Light: https://{s}.basemaps.cartocdn.com/light_all/{z}/{x}/{y}{r}.png
    - Dark: https://{s}.basemaps.cartocdn.com/dark_all/{z}/{x}/{y}{r}.png
2. **Stamen**:
    - Toner: https://stamen-tiles.a.ssl.fastly.net/toner/{z}/{x}/{y}.png
    - Watercolor: https://stamen-tiles.a.ssl.fastly.net/watercolor/{z}/{x}/{y}.png
    - Terrain: https://stamen-tiles.a.ssl.fastly.net/terrain/{z}/{x}/{y}.png
3. **Esri**:
    - Satellite: https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/tile/{z}/{y}/{x}
4. **Thunderforest**:
    - Cycle: https://{s}.tile.thunderforest.com/cycle/{z}/{x}/{y}.png?apikey=YOUR_API_KEY
5. **Mapbox**:
    - Streets: https://api.mapbox.com/styles/v1/mapbox/streets-v11/tiles/{z}/{x}/{y}?access_token=YOUR_ACCESS_TOKEN
6. **OpenTopoMap**:
    - https://{s}.tile.opentopomap.org/{z}/{x}/{y}.png

Replace the URL in L.tileLayer and include the required attribution.

—----------

- **.addTo(map)**: Adds this tile layer to the map object so that the map is visible.

---

**L.marker(coords):** Creates a marker (an icon) at the location specified by coords ([latitude, longitude]).

---

**.addTo(map) :** Add the marker to the map.

---

**.bindPopup('A pretty CSS popup.<br> Easily customizable.')**

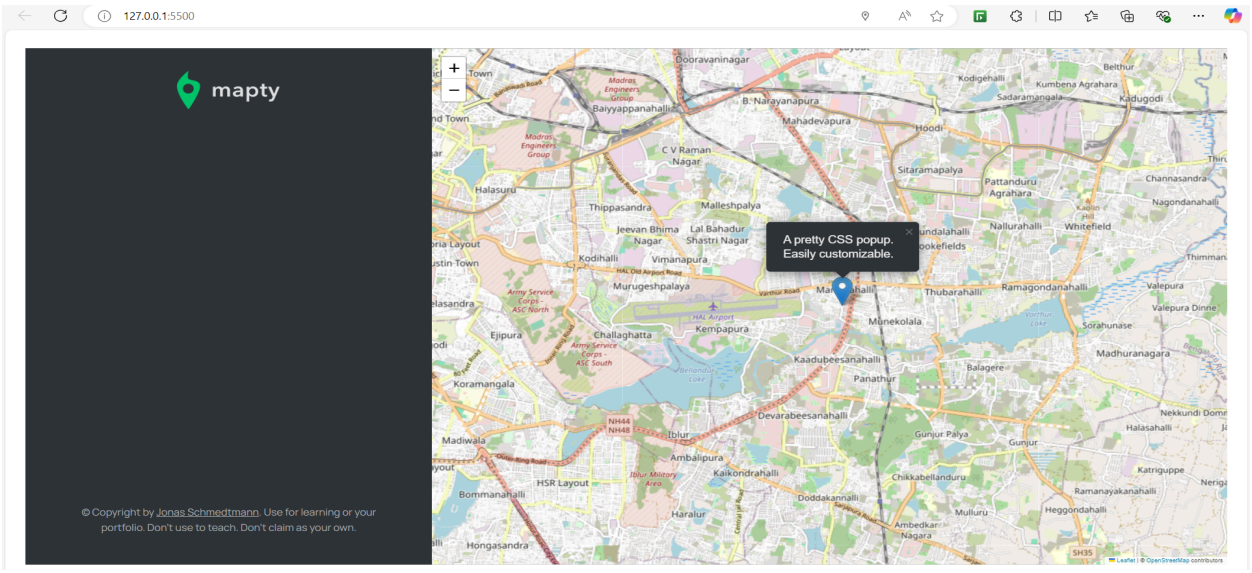
- Attach a popup to the marker.
- The popup will display the HTML content provided as a string ('A pretty CSS popup.<br> Easily customizable.'). <br> is used to create a line break in the text.

---

**.openPopup():** Automatically opens the popup when the map loads, so the message is visible immediately.

---

After the map is loaded according to our current location as we modified the above code to our coordinates.




## Displaying a Map Marker

Navigate **Docs** option in the official site page of [https://leafletjs.com/download.html](https://leafletjs.com/download.html)

Click on the Marker option under **UI Layers** and will get info regarding map markers **.**

**Ex:** Usage example

**L.marker([50.5, 30.5]).addTo(map);**

**JavaScript mapping library like Leaflet, the map.on method is used to attach event listeners to a map object. This allows you to respond to user interactions or other map events.**

**Basic Syntax:**

```
map.on(eventType, function(event) {

    // Your event handling code

});
```

```
if (navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(
    function (position) {
      console.log(position); // Logs the entire position object
      const { latitude, longitude } = position.coords; // Destructuring to get latitude and longitude
      console.log('latitude:', latitude); // Logs the latitude
      console.log('longitude:', longitude); // Logs the longitude
      const coords = [latitude, longitude];
      const map = L.map('map').setView(coords, 13);

      L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution:
          '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors',
      }).addTo(map);

      map.on('click', function (mapEvent) {
        console.log(mapEvent);
        const { lat, lng } = mapEvent.latlng;
        console.log(lat, lng);
        L.marker([lat, lng])
          .addTo(map)
          .bindPopup(
            L.popup({
              minWidth: 250,
              minWidth: 100,
              autoClose: false,
              closeOnClick: false,
              className: 'running-popup',
            })
          )
          .setPopupContent('workout')
          .openPopup();
      });
      // Generates a Google Maps link to view the location
```

This code snippet adds an interactive click event to a Leaflet map. When a user clicks on the map, it:

1. Logs the event object to the console.
2. Extracts the latitude and longitude of the click location.

3. Creates a marker at the click location.
4. Adds a custom-styled popup to the marker with the content "workout."

Here’s a detailed breakdown of what happens:

Event Listener:

```
map.on('click', function (mapEvent) { ... });
```

The map.on('click') method listens for click events on the map. The mapEvent object contains details about the event, including the click location.

Extracting Latitude and Longitude:

```
const { lat, lng } = mapEvent.latlng;
```

The latlng property of the mapEvent object contains the latitude and longitude where the map was clicked.

Adding a Marker:

```
L.marker([lat, lng])

  .addTo(map);
```

A marker is created at the specified [lat, lng] coordinates and added to the map.

Binding a Popup:

```
.bindPopup(

  L.popup({

    minWidth: 250,

    minWidth: 100,

    autoClose: false,

    closeOnClick: false,

    className: 'running-popup',
```

```
    })

);
```

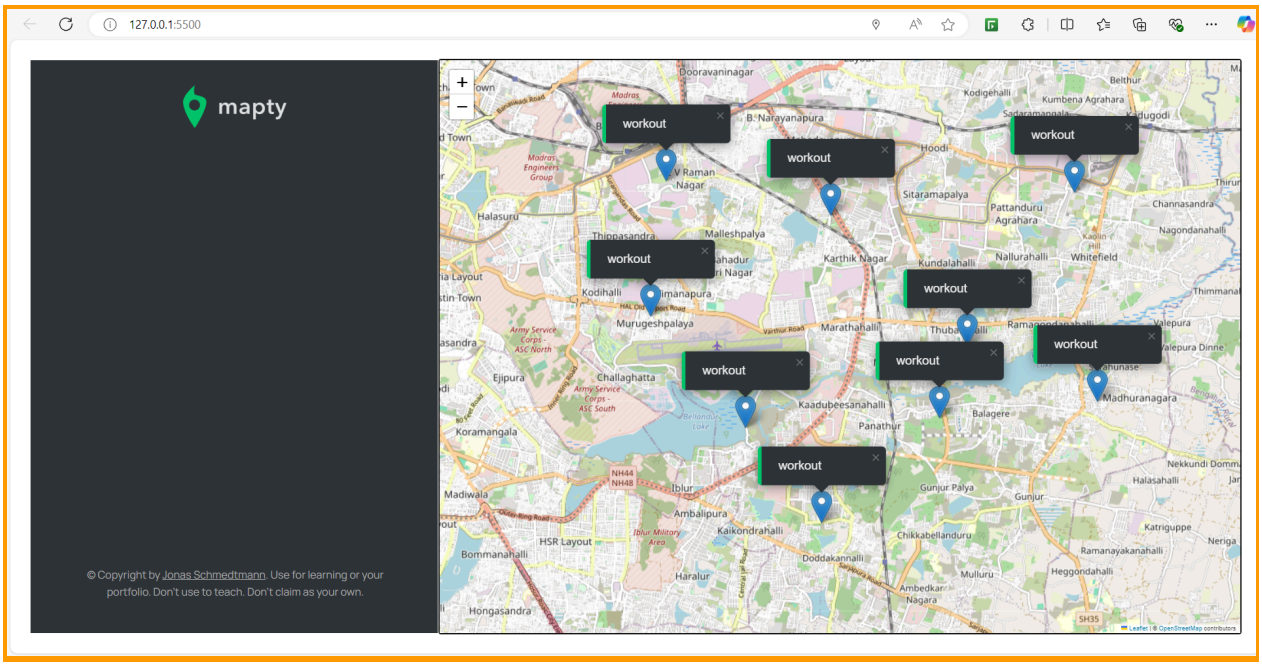
A popup is associated with the marker. The popup:

- Has a minimum width of 250.
- Does not automatically close when another popup opens (autoClose: false).
- Stays open even if the map is clicked elsewhere (closeOnClick: false).
- Uses a custom class running-popup for additional styling.

Setting Content and Opening the Popup:

```
.setPopupContent('workout')

.openPopup();
```

The popup displays the text "workout" and automatically opens upon marker creation.




//

We can call any line code in constructor and if you want to execute the method once the page load then you declare the methods in constructor also as one instance the instance is created then **Constructor**:

- The **constructor** is a special method in a class that is automatically called when a new instance of the class is created.
- You can define any logic inside the constructor, including calling other methods or performing actions like initializing properties.

**Methods in the Constructor**:

- You **can call methods from the constructor**, but it's not always ideal to execute certain methods directly in the constructor, especially if they are related to page load or asynchronous operations.
- However, calling methods in the constructor is **valid** and ensures they are executed as soon as the object is instantiated.

```
if (navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(
    function (position) {
      console.log(position); // Logs the entire position object
      const { latitude, longitude } = position.coords; // Destructuring to get latitude and longitude
      console.log('latitude:', latitude); // Logs the latitude
      console.log('longitude:', longitude); // Logs the longitude
      const coords = [latitude, longitude];
      map = L.map('map').setView(coords, 13);

      L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution:
          '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors',
      }).addTo(map);
      // Generates a Google Maps link to view the location
      console.log(
        `https://www.google.com/maps/@${latitude},${longitude},14z?entry=ttu&g_ep=EgoyMDI0MTIxMS4wIKXMDSoASAFQAw%3D%3D`
      );
      map.on('click', function (mapE) {
        mapEvent = mapE;
        form.classList.remove('hidden');
        inputDistance.focus();
      });
    },
    function (error) {
      console.log('Error: ' + error.message); // Logs the error message if geolocation fails
      alert("Couldn't get your location"); // Alerts the user if location access fails
    }
  );
} else {
  console.log('Geolocation is not supported by this browser.'); // Logs if geolocation is not supported
}
```

```
form.addEventListener('submit', function (e) {
  e.preventDefault();
  //clear input fields
  inputDistance.value =
    inputDuration.value =
    inputCadence.value =
    inputElevation.value =
      '';

  //disply marker
  console.log(mapEvent);
  const { lat, lng } = mapEvent.latlng;
  console.log(lat, lng);
  L.marker([lat, lng])
    .addTo(map)
    .bindPopup(
      L.popup({
        minWidth: 250,
        minWidth: 100,
        autoClose: false,
        closeOnClick: false,
        className: 'running-popup',
      })
    )
    .setPopupContent('workout')
    .openPopup();
});

//

inputType.addEventListener('change', function () {
  inputElevation.closest('.form__row').classList.toggle('form__row--hidden');
  inputCadence.closest('.form__row').classList.toggle('form__row--hidden');
});
```

This code snippet is adding interactivity to the workout form, allowing the user to switch between different input types (e.g., running vs. cycling) and toggle visibility of related input fields. Here's a detailed explanation:

Event Listener:
inputType.addEventListener('change', function () { ... });

- Listens for the change event on the inputType element (likely a dropdown or select menu).
- Triggered when the user selects a different workout type (e.g., "Running" or "Cycling").

Toggling Visibility:

inputElevation.closest('.form__row').classList.toggle('form__row--hidden');

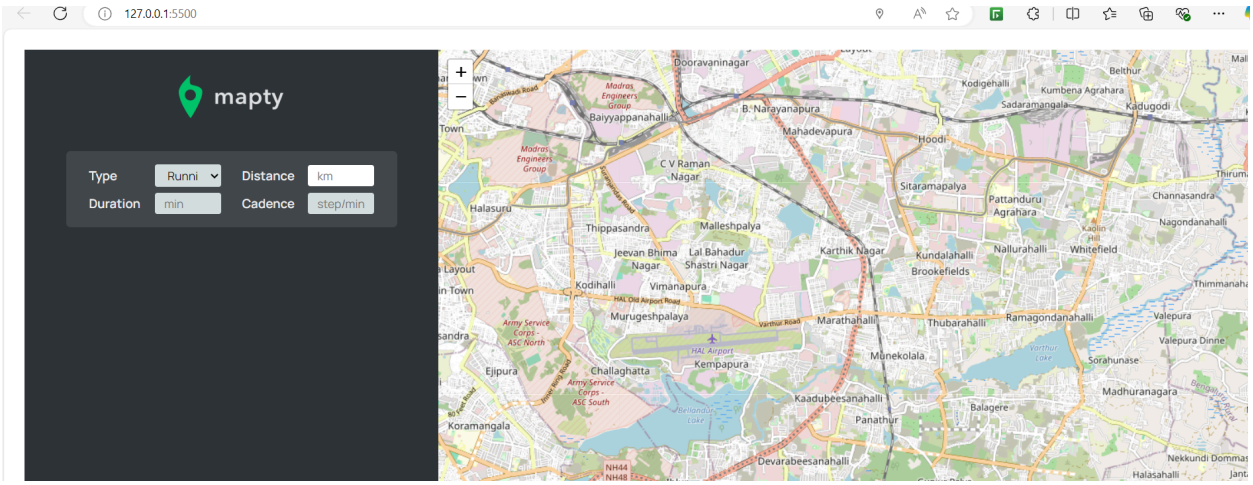
inputCadence.closest('.form__row').classList.toggle('form__row--hidden');

- The .closest('.form__row') method finds the closest parent element with the class form__row for inputElevation and inputCadence.

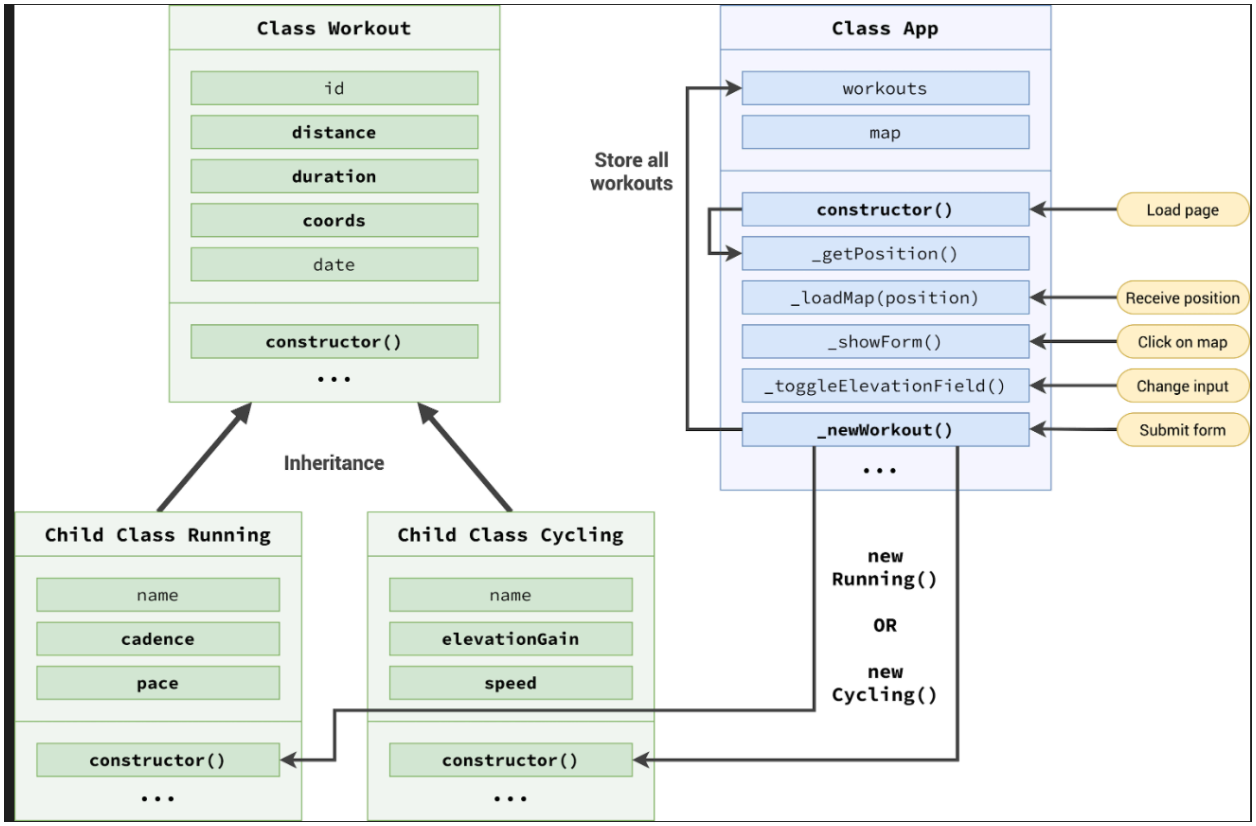
- The classList.toggle() method adds or removes the form__row--hidden class to/from these rows:
  - inputElevation is for activities like Cycling (elevation gain).
  - inputCadence is for activities like Running (cadence or steps per minute).

Behavior

- When the workout type changes:
  - If Running is selected, the Elevation field is hidden, and the Cadence field is shown.
  - If Cycling is selected, the Cadence field is hidden, and the Elevation field is shown.




## Architecture




//hiding the form after 1 sec we display the original property so use setTimeout().

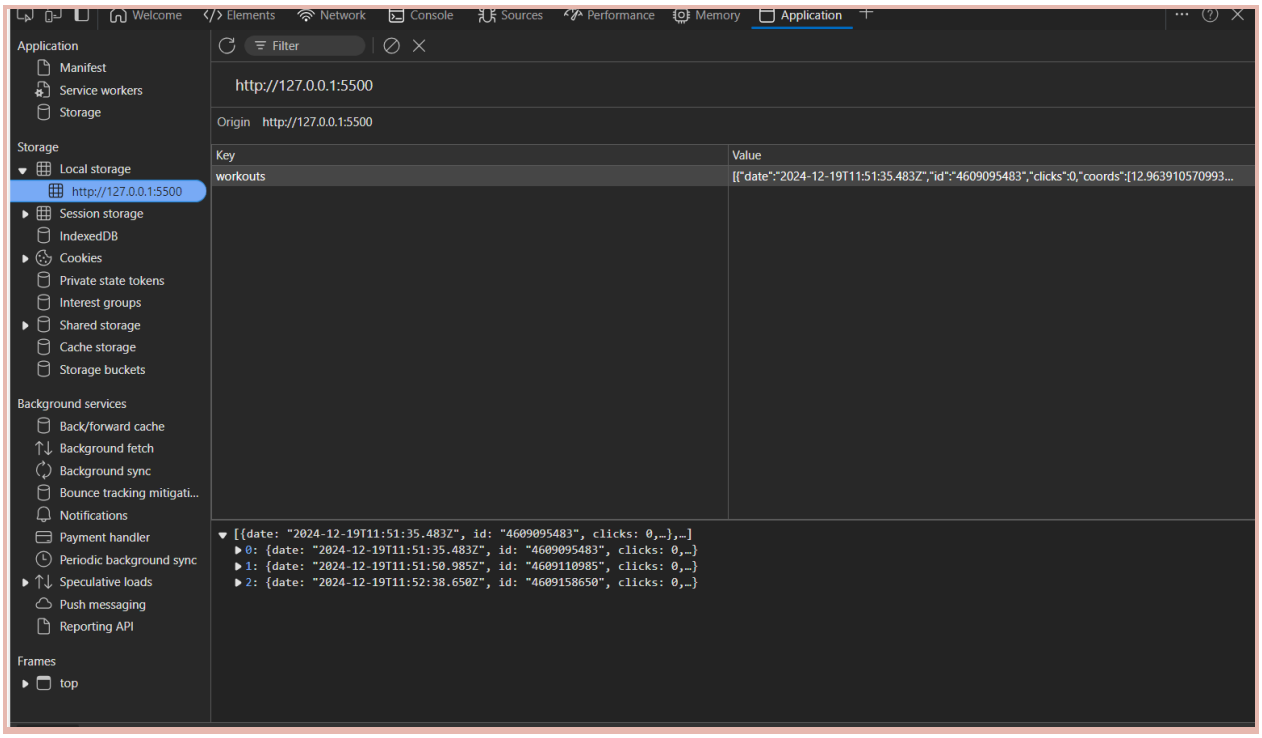
//using public interface
workout.click(); to calculate how many clicks happen in overall function.

//eventhough we reload the page we shouldn't lose data. For that

## Local storage API :

**Local Storage** in JavaScript is a built-in feature of modern web browsers that provides a way to store data in the browser on the user's computer. This storage is accessible via the **window.localStorage** object and allows you to save key-value pairs as strings. Unlike cookies, local storage data is not sent to the server with every HTTP request, making it a more efficient option for client-side storage.

Inspect the browser -> Application->Storage->local storage( data will be there ink key-value pair)




**localStorage Methods**

The localStorage offers some methods to use it. We will discuss all these localStorage methods with examples. Before that, a basic overview of these methods are as follows:

| | |
|---|---|
| setItem() | This method is used to add the data through key and value to localStorage. |
| getItem() | It is used to fetch or retrieve the value from the storage using the key. |
| removeItem() | It removes an item from storage by using the key. |
| clear() | It is used to clear all the storage |

**1. Setting Data**

You can use the setItem() method to store data in local storage.

```
localStorage.setItem('key', 'value');
```

Example:

```
localStorage.setItem('username', 'JohnDoe');
```

**2. Retrieving Data**

The getItem() method is used to retrieve data by its key.

```
const value = localStorage.getItem('key');
```

Example:

```
const username = localStorage.getItem('username');

console.log(username); // Outputs: JohnDoe
```

**3. Removing Data**

The removeItem() method removes a specific key and its value.

```
localStorage.removeItem('key');
```

Example:

```
localStorage.removeItem('username');
```

**4. Clearing All Data**

The clear() method removes all data stored in local storage for that origin.

```
localStorage.clear();
```

**5. Checking Keys**

The key() method retrieves the key at a specific index in local storage.

```
const keyName = localStorage.key(index);
```

Example:

```
const firstKey = localStorage.key(0);

console.log(firstKey);
```

**6. Length**

You can determine the number of stored items using the length property.

```
console.log(localStorage.length);
```

**Advantage of localStorage**

The localStorage has come with several advantages. First and essential advantage of localStorage is that it can store temporary but useful data in the browser, which remains even after the browser window closes. Below is a list of some advantages:

- The data collected by localStorage is stored in the browser. You can store 5 MB data in the browser.
- There is no expiry date of data stored by localStorage.
- You can remove all the localStorage items by a single line code, i.e., clear().
- The localStorage data persists even after closing the browser window, like items in a shopping cart.
- It also has advantages over cookies because it can store more data than cookies.

**Limitation of localStorage**

As the localStorage allows to store temporary, local data, which remains even after closing the browser window, but it also has few limitations. Below are some limitations of localStorage are given:

- Do not store sensitive information like username and password in localStorage.
- localStorage has no data protection and can be accessed using any code. So, it is quite insecure.
- You can store only a maximum of 5MB data on the browser using localStorage.
- localStorage stores the information only on browsers, not in server-based databases.
- localStorage is synchronous, which means that each operation executes one after another.

---

**1. JSON.stringify()**

The JSON.stringify() method is used to **convert a JavaScript object or value into a JSON string**. This is useful for storing or transmitting data in string format, such as saving it to local storage or sending it in an API request.

**Syntax**

JSON.stringify(value, replacer, space);

- **value**: The JavaScript object or value to be converted.
- **replacer** *(optional)*: A function or array to transform the result.
- **space** *(optional)*: Adds indentation, whitespace, or line breaks for readability.

**Example**

```
const user = { name: "John", age: 30, hobbies: ["reading", "gaming"] };

// Convert object to JSON string
const jsonString = JSON.stringify(user);
console.log(jsonString);
// Output: '{"name":"John","age":30,"hobbies":["reading","gaming"]}'
```

**With Formatting (Readable JSON)**

```
const formattedString = JSON.stringify(user, null, 2);
console.log(formattedString);
Output:
{
  "name": "John",
  "age": 30,
  "hobbies": [
    "reading",
    "gaming"
  ]
}
```

---

**2. JSON.parse()**

The JSON.parse() method is used to **convert a JSON string back into a JavaScript object or value**. This is commonly used to retrieve data stored as a string (e.g., in local storage or received from an API).

**Syntax**

JSON.parse(text, reviver);

- **text**: The JSON string to parse.
- **reviver** *(optional)*: A function that transforms the result.

**Example**

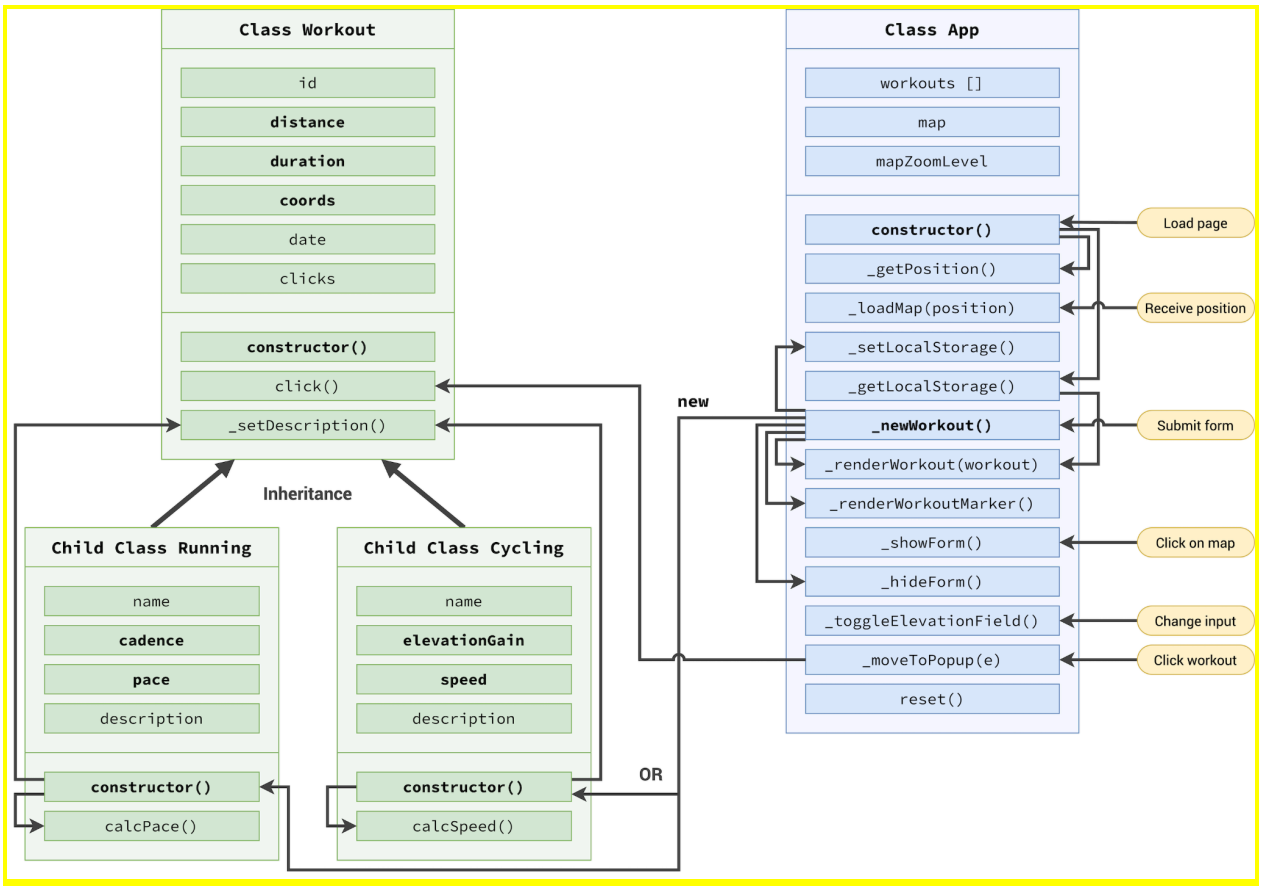
```
const jsonString = '{"name":"John","age":30,"hobbies":["reading","gaming"]}';

// Convert JSON string back to object
const user = JSON.parse(jsonString);
console.log(user);
// Output: { name: 'John', age: 30, hobbies: [ 'reading', 'gaming' ] }
```

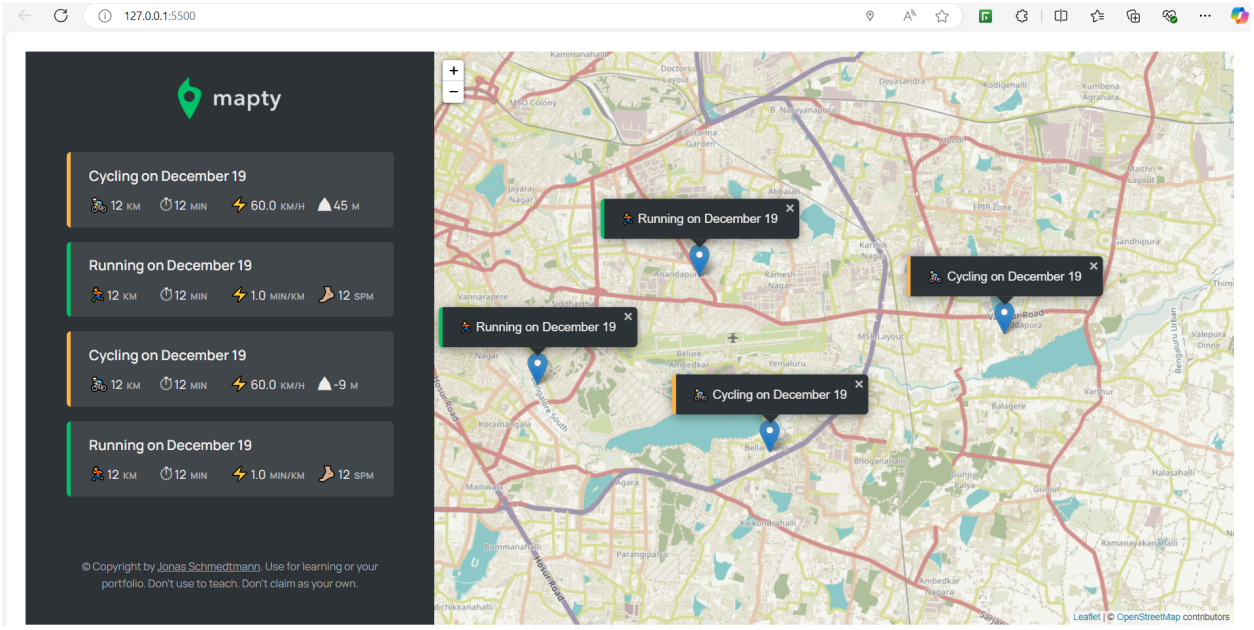
```
  _setLocalStorage() {
    localStorage.setItem('workouts', JSON.stringify(this.#workouts));
  }

  _getLocalStorage() {
    const data = JSON.parse(localStorage.getItem('workouts'));

    if (!data) return;

    this.#workouts = data;

    this.#workouts.forEach(work => {
      this._renderWorkout(work);
    });
  }

  reset() {
    localStorage.removeItem('workouts');
    location.reload();
  }
}

const app = new App();
//app.reset();It will use removeItem in reset() method  and delete data from local storage.
```

```
_loadMap(position) {
  const { latitude } = position.coords;
  const { longitude } = position.coords;
  // console.log(`https://www.google.pt/maps/@${latitude},${longitude}`);

  const coords = [latitude, longitude];

  this.#map = L.map('map').setView(coords, this.#mapZoomLevel);

  L.tileLayer('https://{s}.tile.openstreetmap.fr/hot/{z}/{x}/{y}.png', {
    attribution:
      '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors',
  }).addTo(this.#map);

  // Handling clicks on map
  this.#map.on('click', this._showForm.bind(this));

  this.#workouts.forEach(work => {
    this._renderWorkoutMarker(work);
  });
}
```

The data will be get and visible from local storage during browser loading itself
As we mentioned in `_loadMap(position) method`
and `_loadMap(position) method` is called from the constructor.. so once an instance is created then immediately the constructor is called.

**Finally**


Class Workout
id
distance
duration
coords
date
clicks
constructor()
click()
_setDescription()
Inheritance
Child Class Running
name
cadence
pace
description
constructor()
calcPace()
Child Class Cycling
name
elevationGain
speed
description
constructor()
calcSpeed()
OR
new
Class App
workouts []
map
mapZoomLevel
constructor()
_getPosition()
_loadMap(position)
_setLocalStorage()
_getLocalStorage()
_newWorkout()
_renderWorkout(workout)
_renderWorkoutMarker()
_showForm()
_hideForm()
_toggleElevationField()
_moveToPopup(e)
reset()
Load page
Receive position
Submit form
Click on map
Change input
Click workout


127.0.0.1:5500
mapty
Cycling on December 19
12 KM 12 MIN 60.0 KM/H 45 M
Running on December 19
12 KM 12 MIN 1.0 MIN/KM 12 SPM
Cycling on December 19
12 KM 12 MIN 60.0 KM/H -9 M
Running on December 19
12 KM 12 MIN 1.0 MIN/KM 12 SPM
© Copyright by Jonas Schmedtmann. Use for learning or your portfolio. Don't use to teach. Don't claim as your own.
Running on December 19
Cycling on December 19
Running on December 19
Cycling on December 19
Leaflet | © OpenStreetMap contributors