

# 10. Używanie pluginów JS



## Wyzwania:

- Nauczysz się korzystania z pluginów JS
- Poznasz szablony HTML (Mustache)
- Wstawisz mapę do swojego projektu
- Dowiesz się czym jest jQuery

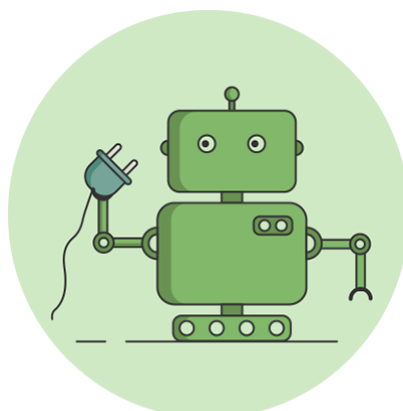
Zadanie: Generowanie slajdów karuzeli

## 10.1. Korzystanie z pluginów



W poprzednich modułach nauczyliśmy się podstaw JavaScriptu, pisząc własne skrypty. Zaczynaliśmy od bardzo prostych skryptów, które z czasem rozbudowaliśmy do w miarę skomplikowanych. To bardzo dobre podejście i ogromnie istotna umiejętność, z której będziesz korzystać codziennie — zarówno w trakcie kursu, jak i później w swojej pracy.

Kolejnym krokiem jest nauka korzystania z cudzego kodu, np.:



- znajdując przykład kodu w internecie możesz go wykorzystać, tylko częściowo go przerabiając,



- pracując w zespole, będziesz zajmować się tylko częścią kodu,
- opierając pracę na frameworku, np. React.js, korzystasz z kodu tego frameworka, napisanego przez innych programistów,
- niektórych funkcjonalności nie warto pisać samodzielnie, jeśli istnieje dobrej jakości plugin.

W każdym z tych przypadków istotne są punkty styku pomiędzy cudzym kodem, a Twoim własnym. Jeśli te punkty styku są dobrze opracowane, dużo prościej będzie bezproblemowe działanie tego połączanego kodu.

W tym module będziemy zajmować się ostatnim z wymienionych przykładów, czyli pluginami.

---

## Założenia projektu

---

Naszym celem w tym module będzie stworzenie strony, która prezentuje listę miejsc. W naszym przykładzie będzie to lista sklepów, ale najlepiej będzie, jeśli wymyślisz własne przeznaczenie tej strony, np.:

- miejsca na świecie, które chcę odwiedzić,
- najpiękniejsze budynki, jakie widziałem na własne oczy,
- moje ulubione pizzerie,
- gdzie najchętniej jeżdżę na grzyby,
- najważniejsze obrazy mojego ulubionego malarza i muzea, w których się znajdują.

Strona będzie składała się z dwóch części:

1. karuzeli, w której każdy slajd odpowiada jednej lokalizacji,
2. mapy Google, na której będą oznaczone wszystkie lokalizacje.

Oba te elementy zaimplementujemy za pomocą pluginów, a następnie połączymy ich działanie ze sobą. Przewinięcie karuzeli będzie centrować mapę na odpowiednim markerze, a kliknięcie w marker będzie przewijać karuzelę do odpowiedniego slajdu.

W ten sposób stosunkowo małym nakładem pracy będziemy mogli osiągnąć świetny efekt!

---

## Kilka słów o jQuery

---



Jeśli jeszcze nie zdarzyło Ci się spotkać z nazwą tej biblioteki, na pewno wielokrotnie będziesz mieć z nią styczność. Jest to bardzo popularna biblioteka, używana przez bardzo wiele stron. Wiele pluginów jest też opartych o jQuery. Dlaczego w takim razie nie wykorzystujemy jQuery od początku, i w tym projekcie również nie będziemy korzystać z tej biblioteki?

Zanim odpowiemy na to pytanie, musimy wyjaśnić, czym jest jQuery. Jest to biblioteka zawierająca narzędzia ułatwiające pisanie skryptów, zwłaszcza w zakresie dotyczącym elementów DOM, np. znajdowanie i modyfikacja elementów na stronie, oraz przypinanie akcji do eventów.

Przez lata jQuery cieszyło się olbrzymią popularnością, ponieważ ułatwiało pisanie kodu działającego w różnych przeglądarkach. Np. silnik JavaScript w Internet Explorer 7 różnił się znacząco od silnika, na którym oparto Firefoksa, do tego stopnia, że nawet wyszukanie elementów z pewną klasą wymagało zupełnie innego kodu.

Dziś jednak sytuacja wygląda zupełnie inaczej i zdecydowaną większość funkcjonalności dostarczanej przez jQuery można łatwo osiągnąć za pomocą tzw. **vanilla JS**, czyli czystego JavaScriptu – tzn. bez wykorzystania jQuery czy innych bibliotek.

Co więcej, znajomość czystego JavaScriptu ułatwi Ci zaznajomienie się z jQuery, podczas gdy dużo trudniej przejść na vanilla JS, jeśli od początku korzystało się z jQuery.

W związku z tym poruszymy jeszcze temat jQuery, aby zapoznać Cię z jego podstawowymi założeniami i przygotować do pracy na kodzie wykorzystującym tę bibliotekę, ale w tym projekcie, podobnie jak w większości innych, nie będziemy z niej korzystać.

---

## Dokumentacja pluginu

---

Przed decyzją o wykorzystaniu pluginu warto sprawdzić, jakiej jakości jest jego dokumentacja. Często właśnie od tego zależy, czy plugin będzie ogromnie przydatny, czy zupełnie bezużyteczny.

Pamiętaj jednak, że są różne style pisania dokumentacji i niektóre mogą w pierwszej chwili wydawać się skomplikowane. Warto poświęcić im chwilę i zwrócić uwagę m.in. na objętość dokumentacji.



Jako przykład możemy wykorzystać dokumentację karuzeli [Flickity](#), którą będziemy za chwilę implementować. W menu możemy zobaczyć dość standardowy podział na sekcje:

- Getting started, zamieszczone na stronie głównej, zawiera podstawowe informacje niezbędne do uruchomienia pluginu,
- Options, czyli opcje, które możemy ustawić w momencie uruchamiania pluginu (np. czy karuzela ma samodzielnie przewijać się do kolejnego slajdu co kilka sekund),
- API, czyli w jaki sposób możemy wpływać na plugin już po jego uruchomieniu (np. w jaki sposób przewinąć karuzelę do 5. slajdu za pomocą customowego guzika czy linka),
- Events, czyli jakie zdarzenia są generowane przez plugin i w jaki sposób ich nasłuchiwać (np. jak możemy wywołać naszą funkcję w momencie przejścia na inny slajd).

Oprócz tego autorzy Flickity umieścili na swojej stronie również przykładowe style karuzeli (sekcja Style), dodatkowe informacje włącznie z dodatkowymi przykładami zastosowania, oraz informacje dotyczące licencji pluginu.

Zwróć uwagę, że z tego pluginu można korzystać zarówno z pomocą biblioteki jQuery, jak i bez niej (vanilla JS). W całej dokumentacji przykłady kodu są podane w obu wersjach, przy czym czasami wersja vanilla JS jest tylko podlinkowana jako "vanilla JS demo".

Dzięki tak świetnej dokumentacji, nie powinniśmy mieć problemów z wykorzystaniem tego pluginu!

---

## Zadanie: Podłączamy karuzelę Flickity

---

Twoim zadaniem jest skorzystanie z dokumentacji pluginu [Flickity](#) do zaimplementowania karuzeli w nowym projekcie. Nie będziemy Ci w tym pomagać i omawiać fragmentów kodu, ponieważ przykłady znajdziesz w dokumentacji. Zamiast tego, pokrótce opiszemy krok po kroku jak zrealizować to zadanie.

Pod instrukcją zamieściliśmy też kilka wyjaśnień dotyczących zagadnień poruszonych w instrukcji.

1. Stwórz nowe repozytorium na GitHubie i sklonuj je na swój komputer.
2. Przenieś do niego plik `package.json` z poprzedniego projektu (lub z ostatniego zadania w module 7).



3. Otwórz terminal w katalogu projektu i uruchom `npm run init-project` , a następnie `npm run watch` .
4. Zapisz pierwszy commit, wyślij na zdalne repozytorium i sprawdź przez stronę GitHuba czy w projekcie znajdują się pliki. Od tej pory samodzielnie decyduj o tym jak często zapisywać commity i wysyłać je na zdalne repozytorium. Pamiętaj o nazywaniu commitów zgodnie z dobrymi praktykami, opisanymi w module 7!
5. W pliku `index.html` stwórz podstawowy kod html (doctype, html, head, title, body, etc.).
6. W pliku `index.html` , w `<head>` podłącz kolejno:
  1. pobrany plik `Normalize.css` umieszczony w katalogu `vendor` ,
  2. pobrany plik `flickity.min.css` umieszczony w katalogu `vendor` ,
  3. plik `css/style.css` , który generuje się z naszego `sass/style.scss` .
7. W pliku `index.html` , tuż przed zamknięciem `</body>` , podłącz kolejno:
  1. pobrany plik `flickity.pkgd.min.js` umieszczony w katalogu `vendor` ,
  2. nasz plik `js/script.js` .
8. Zaimplementuj karuzelę Flickity wedle instrukcji z sekcji `Getting started`. Umieść w niej co najmniej 5 slajdów. Wykonując instrukcję z dokumentacji pamiętaj, że pliki css i js zostały już podłączone w powyższych punktach, oraz chcemy inicjować karuzelę (w wersji vanilla JS) w pliku `js/script.js` .
9. Chcemy, aby nasza karuzela zajmowała 100% szerokości okna (przykłady w sekcji `Style` dokumentacji pluginu), ale tekst zawarty w niej ma mieć ograniczoną szerokość. Wysokość karuzeli powinna wynosić 50% wysokości okna, ale nie mniej niż 250px. Dodatkowo:
  1. W każdym slajdzie wstaw div o klasie `container` i dodaj dla niego style, które ograniczą jego szerokość (np. do `1200px` ) i wyśrodkują go w poziomie. Dodaj przykładowy tekst wewnątrz containera każdego slajdu.
  2. Dla każdego slajdu nadaj jakieś tło obrazkowe dodając atrybut `style` w kodzie HTML. Jest to wyjątek, kiedy style nadajemy w kodzie HTML, który wynika z faktu, że to w kodzie HTML chcemy podawać url do obrazka wyświetlanego w slajdzie. Użyjemy jednak do tego właściwości `background-image` i żadnych innych stylów nie wpisujemy w kodzie HTML, aby nie nadpisać żadnych stylów z pliku `style.css` .
  3. Dla wszystkich slajdów nadaj styl `background-size: cover;` .

10. Zmień opcje pluginu, aby:



1. nie wyświetlały się kropki pod karuzelą,
  2. karuzela zmieniała url strony dodając hash, dzięki czemu po odświeżeniu strony będzie widoczny ostatnio oglądany slajd (opcja "hash" z [Options](#) w dokumentacji).
11. Korzystając z sekcji [API](#) w dokumentacji, dodaj w rogu karuzeli (używając `position: absolute`) guzik "Restart", który przewija karuzelę do pierwszego slajdu.
12. Użyj pierwszego przykładu dla opcji `scroll` z sekcji [Events](#) w dokumentacji, aby dodać płynnie przewijany pasek postępu pod karuzelą.

## Kilka wyjaśnień

- Wszystkie pliki pluginów i bibliotek, o ile nie będziemy ich modyfikować, umieszczamy w katalogu `vendor` (można w podkatalogach). Dzięki temu będziemy pamiętać, że te pliki nie były modyfikowane. Przypominamy też, że wszystkie pliki z katalogu `css` są kasowane, aby znajdowały się tam wyłącznie pliki wygenerowane z plików w katalogu `sass`.
- Style zawsze podłączamy w `<head>`. Kolejność plików ma znaczenie, ponieważ chcemy mieć możliwość nadpisywania stylu np. pluginu bez niepotrzebnego zwiększania szczegółowości selektora.
- Skrypty podłączamy zawsze na samym końcu `<body>`, tzn. tuż przed `</body>`.
- [Normalize.css](#) służy do nadania podstawowych, domyślnych stylów dla elementów HTML (np. nagłówków, paragrafów, itd.). Dzięki temu zostaną zniwelowane różnice w domyślnych stylach, które występują w różnych przeglądarkach. Podłączamy go zawsze jako pierwszy styl.
- Jeśli plugin lub biblioteka oferuje zminifikowane wersje plików, korzystamy z nich. I tak nie będziemy ich modyfikować, a zajmą trochę mniej miejsca.
- Wiele pluginów i bibliotek (w tym Flickity) oferuje pliki dostępne przez CDN, czyli podłączane na stronie za pomocą linka do zewnętrznego serwera. Dzięki takiemu podejściu jest szansa, że użytkownik spotkał się z tym samym plikiem podłączonym do innej strony w internecie, i wchodząc na naszą stronę nie będzie musiał pobierać tego pliku, tylko skorzysta z wersji znajdującej się w cache przeglądarki. Wymaga to jednak ciągłego utrzymywania strony, ponieważ linki do CDN mogą się zmieniać, stare wersje plików mogą być usuwane lub zmieniane na nowe wersje, niekoniecznie kompatybilne z naszymi skryptami. Dlatego bezpieczniej jest podłączać pliki, które zostały pobrane i znajdują się w naszym repozytorium.

## Podsumowanie



Jeśli wszystko poszło zgodnie z planem, widzisz teraz na swojej stronie estetyczną karuzelę. Każdy ze slajdów składa się z dużego zdjęcia w tle, na którym znajduje się tekst opisujący dany slajd. To świetny początek naszego projektu!

Pamiętaj, aby sprawdzić czy Twoja karuzela działa poprawnie w różnych szerokościach okna przeglądarki. Jeśli nie, dodaj odpowiednie style za pomocą Media Queries.

Po skończeniu zadania, opublikuj projekt za pomocą GitHub Pages i wyślij do mentora link do repozytorium zawierającego kod projektu.

Podgląd zadania

<https://0na.github.io/Flic>

Wyślij link ✓

## 10.2. Szablony HTML



Do tej pory, kiedy potrzebowaliśmy za pomocą JSa dodać nowy element lub tekst na stronie, wpisywaliśmy go w kodzie JS. Nie jest to jednak dobra praktyka, ponieważ jeśli kiedyś będzie potrzebna zmiana kodu HTML, będzie trudniej go zmienić, jeśli będzie zaszyty w kodzie JS. Co więcej, gdybyśmy chcieli np. zmienić wersję językową strony, zmiany będzie wymagał nie tylko plik `index.html`, ale również `js/script.js`.

Rozwiązaniem tych problemów jest wykorzystanie szablonów HTML.

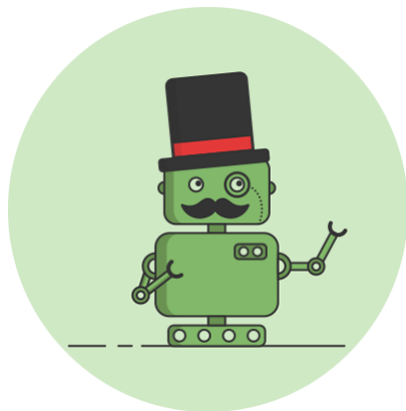
---

## Szablony Mustache.js

---

Jest wiele pluginów umożliwiających wykorzystanie szablonów kodu HTML. Jednym z prostszych i bardziej niezawodnych jest `Mustache.js`, z którego za chwilę skorzystamv





Jak możesz zobaczyć na stronie tego pluginu, najprostszym przykładem szablonu jest:

```
<script id="template" type="x-tmpl-mustache">
Hello {{ name }}!

</script>
```

W pierwszej chwili może Cię wprowadzać w błąd zastosowanie tagu `<script>`, co sugerowałoby, że znajduje się w nim kod JS. Musisz jednak wiedzieć, że przeglądarka interpretuje zawartość tagu `<script>` jako kod JS tylko jeśli nie podano atrybutu `type`, lub jego wartość to `text/javascript`. Właśnie dlatego w tym przypadku podajemy atrybut `type="x-tmpl-mustache"`, aby przeglądarka nie interpretowała zawartości tego tagu.

Dzięki takiemu podejściu możemy w kodzie HTML umieścić ten szablon i nie zostanie on wyświetlony na stronie, ale będziemy mieli do niego dostęp w naszym skrypcie. Wykorzystamy do tego `id` tagu `<script>`.

W zawartości tego tagu została wpisana nazwa zmiennej `name`, zamknięta w podwójnym nawiasie klamrowym. Jak za chwilę zobaczysz, cały fragment `{{ name }}` zostanie podmieniony na wartość, którą podamy w momencie parsowania szablonu.

Przykłady zawarte w dokumentacji pluginu często korzystają z jQuery, dlatego sami omówimy sobie implementację szablonów Mustache.js.





HTML SCSS JS Result 

```
1 'use strict';
2 (function(){
3     // Pierwszym
    krokiem jest
    znalezienie
    elementu
    <script> po id
    (w naszym
    wypadku
    "template-
    hello"). Od razu
    po znalezieniu
    tego elementu
    pobierzemy jego
    zawartość za
    pomocą
    właściwości
    innerHTML
4
5     var
    templateHello =
    document.getElem
    entById('templat
    e-
    hello').innerHTM
    L;
6
7     // Następnie
    opcjonalnie
    możemy wykonać
```

**Templates tests:**  
  
Hello John Smith!  
  
Hello Maria Jones!  
  
Hello Tom Newman!

Resources 1x 0.5x 0.25x Rerun

Weźmy teraz inny przypadek — często będziemy spotykać się z sytuacją, w której będziemy dostawać tablicę zawierającą wiele elementów.

Mogą to być np. nazwy produktów, które chcemy mieć zapisane w pliku HTML, aby kod z pliku `script.js` mógł działać tak samo, niezależnie od wersji językowej strony.

HTMLSCSSJSResult

1 'use strict';

2 (function(){

3 /\* W kodzie

HTML tego

przykładu

dodaliśmy też

dwa szablony:

4 - template-

product-list,

który jest

wrapperem listy

produktów, oraz

5 - template-

product-item,

który jest

szablonem

pojedynczej

pozycji na

liście.

6

7 Oprócz tego

dodaliśmy też

tablicę,

zapisaną w

zmiennej

productsData,

zawierającą

kilka obiektów.

Każdy z nich

reprezentuje

Templates

tests:

Product list

- Buy Football for \$10!
- Buy Volleyball for \$8!
- Buy Basketball for \$12!

Resources1x0.5x0.25xRerun

## Zadanie: Generowanie slajdów karuzeli

Czas użyć szablonów HTML w naszym projekcie!

Stwórz w swoim repozytorium nowy branch, który po skończeniu zadania prześlesz swojemu mentorowi.

Zacznij od podłączenia pliku js pluginu [Mustache.js](#). Pamiętaj, aby umieścić go tuż nad odwołaniem do `js/script.js`.

Następnie stwórz w pliku `index.html` zmienną z tablicą zawierającą kilka obiektów. Każdy z tych obiektów będzie reprezentował jeden slajd, i będzie zawierał właściwości `image`, `title` i `description`. Ostatnia z nich ma zawierać tagi HTML, czyli może mieć np. wartość:



```
"<p>Lorem ipsum.</p> <img src='http://via.placeholder.com/400x200' al
```

Później, na podstawie kodu pojedynczego slajdu, stwórz szablon slajdu. W divie `.container` każdego slajdu wstaw nagłówek, zawierający pole `title`, a pod nagłówkiem dodaj pole `description`. Pamiętaj, że to pole będzie dodawało kod HTML, a nie tekst!

Po stworzeniu szablonu możesz skasować slajdy dodane wcześniej do karuzeli w kodzie HTML. Nie kasuj jednak elementu, w którym znajdowały się slajdy – będziemy do niego dodawać slajdy wygenerowane na podstawie szablonu i tablicy z danymi.

Na końcu w kodzie JS dopisz pętlę (koniecznie przed zainicjowaniem karuzeli Flickity), która wygeneruje kod wszystkich slajdów. Zwróć uwagę, że będzie to prostsze zastosowanie niż we wcześniejszym przykładzie, ponieważ nie musimy używać drugiego szablonu (wrappera listy).

Jeśli masz problem z tym punktem, wróć do tamtego przykładu i zamiast korzystać z szablonu `template-product-list`, umieść w bezpośrednio w kodzie nagłówków oraz `<ul></ul>`, nadaj jakieś `id` dla `<ul>` i zmień kod JS w taki sposób, aby zawartość zmiennej `listItems` była wstawiana do tego `<ul>`.

Jeżeli wszystko pójdzie zgodnie z planem, a kod JS tego zadania będzie umieszczony przed zainicjowaniem pluginu Flickity, na Twojej stronie pojawi się jest karuzela ze slajdami wygenerowanymi za pomocą szablonów.

Podgląd zadania

<https://0na.github.io/Flic>

Wyślij link ✓

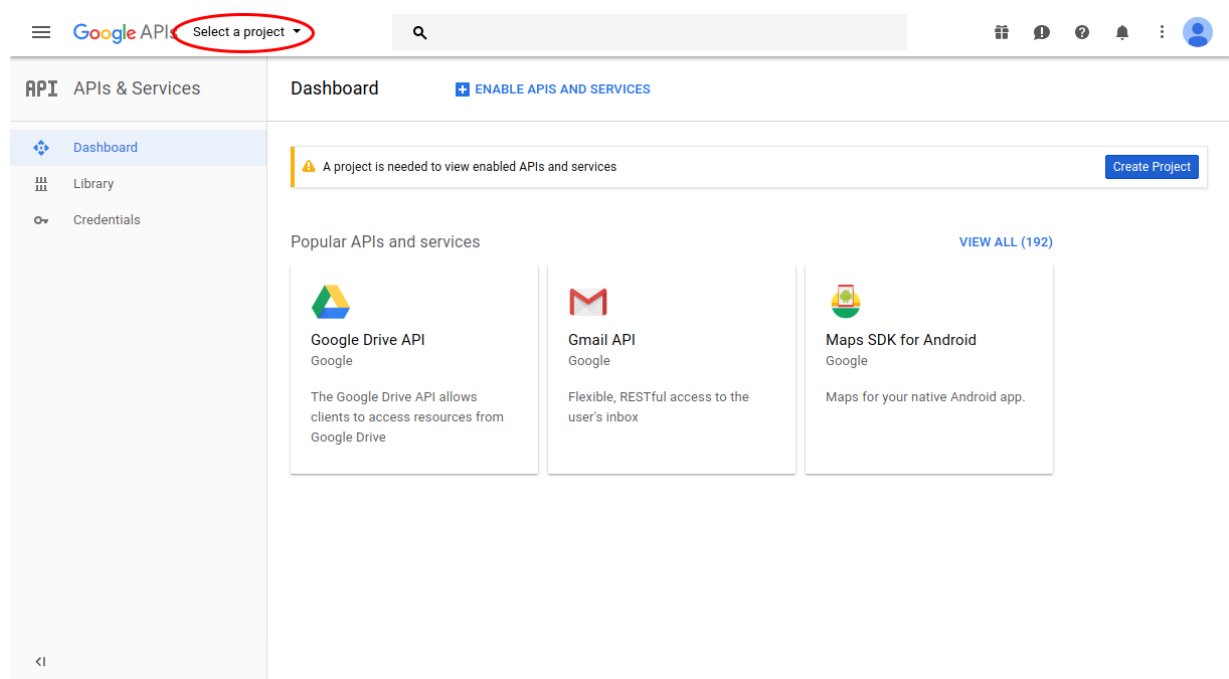
## 10.3. Google Maps



Wstawienie na swojej stronie Mapy Google często przysparza młodych web developerów o ból głowy. Pierwszy problem zaczyna się jeszcze zanim dotkniemy choćby jednej linii kodu - najpierw musimy wygenerować specjalny klucz umożliwiający korzystanie z map.

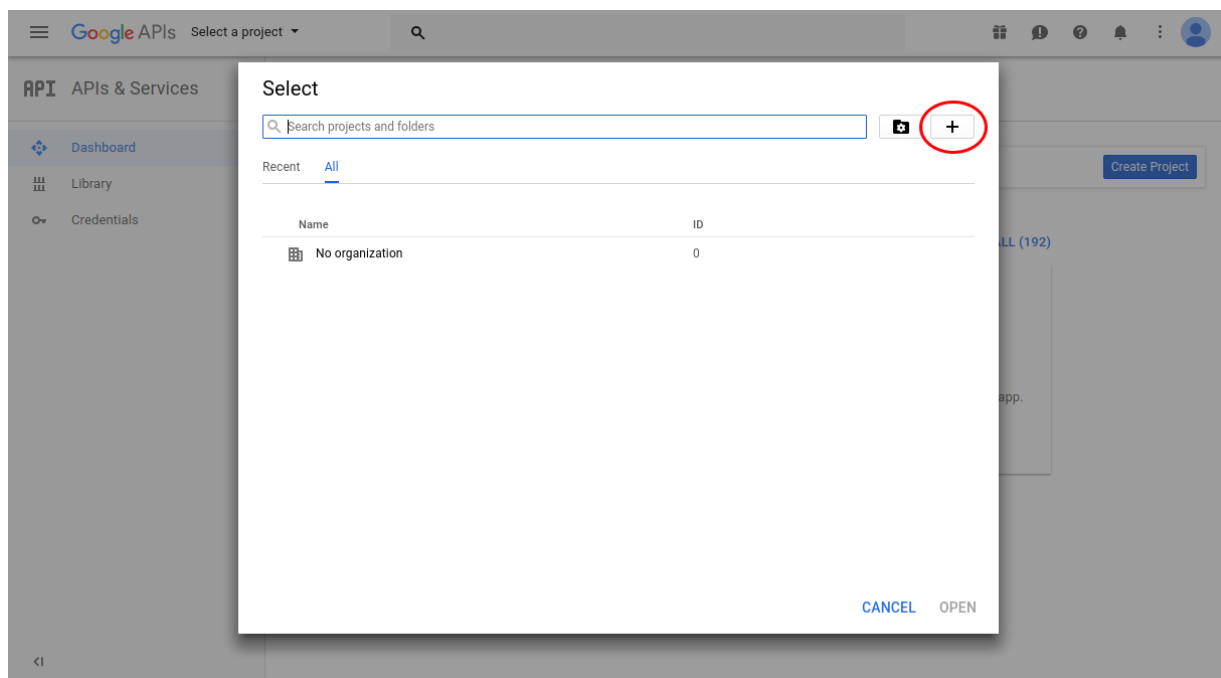
## Uzyskanie API Key

Przejdź do [Google API Console](#) i zaloguj się swoim kontem Google. Jeśli nie masz konta Google, załóż je teraz. Po zalogowaniu kliknij na dropdown *Select a project* u góry strony.

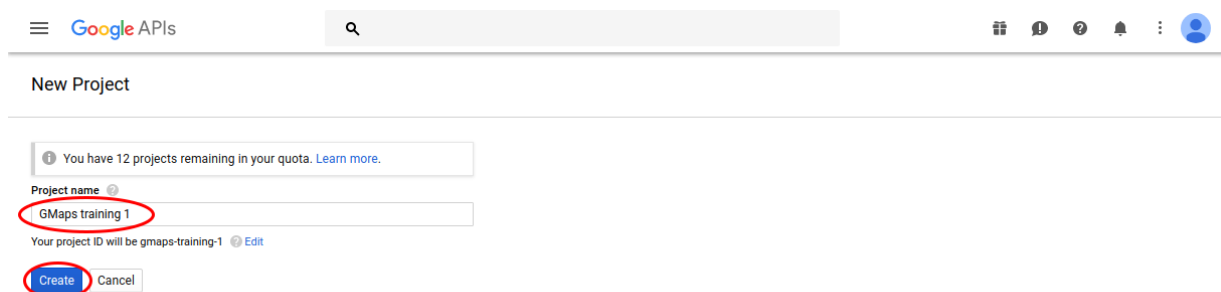


Następnie kliknij guzik z plusem.

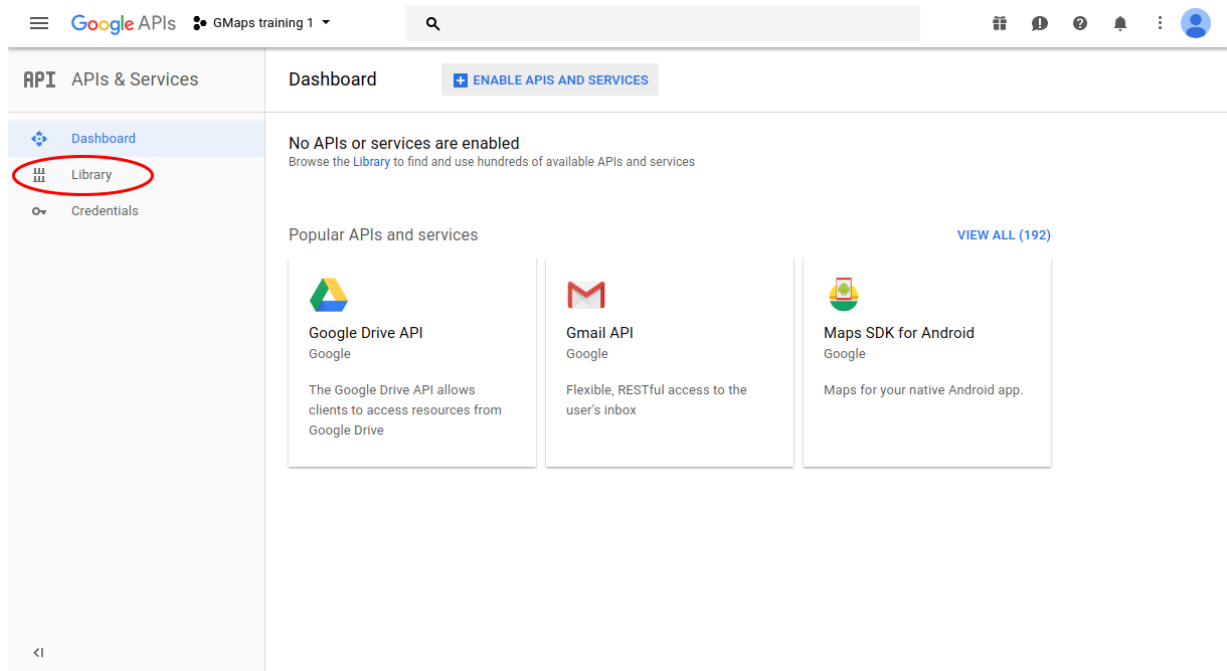




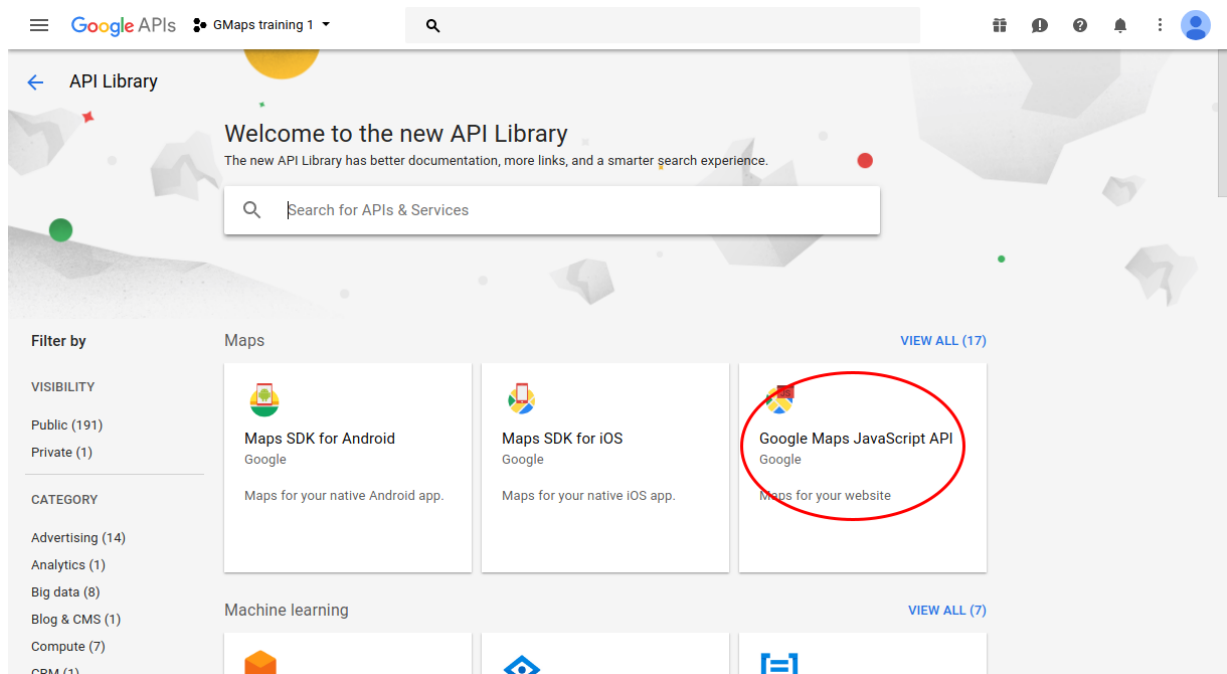
W kolejnym kroku wpisz nazwę swojego projektu, np. "GMaps training 1" i kliknij guzik *Create*.



Po powrocie do Dashboardu przejdź do *Library*.

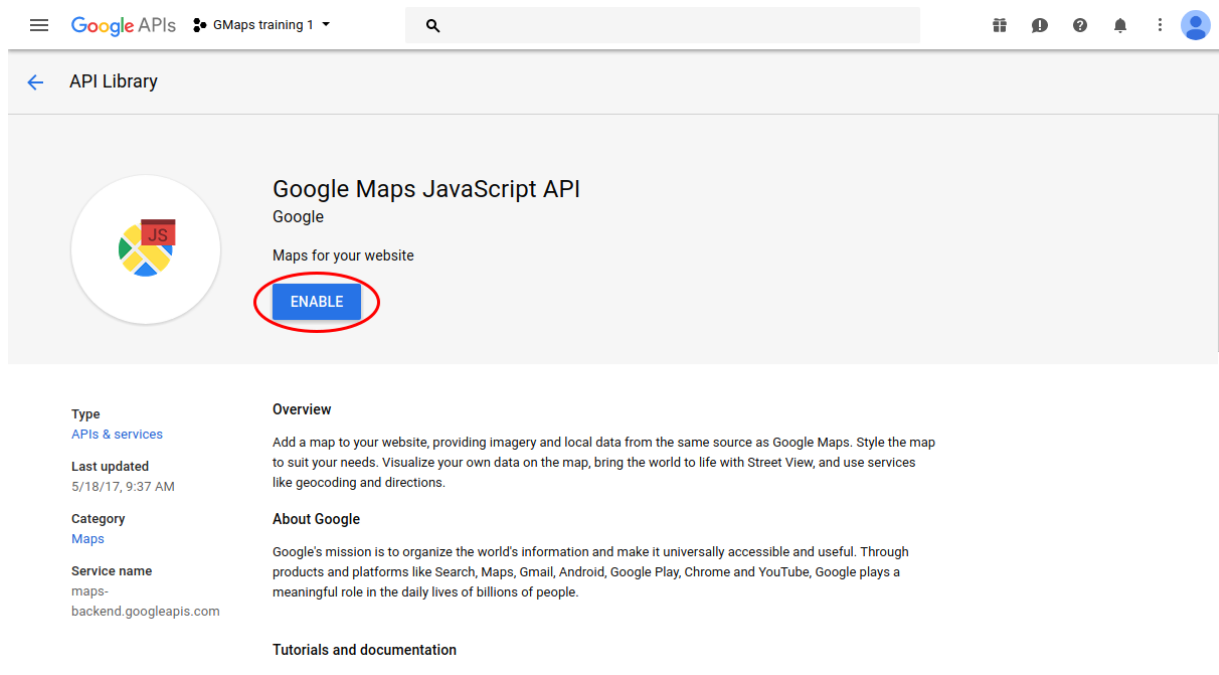


Następnie z kategorii *Maps* wybierz *Google Maps JavaScript API*.

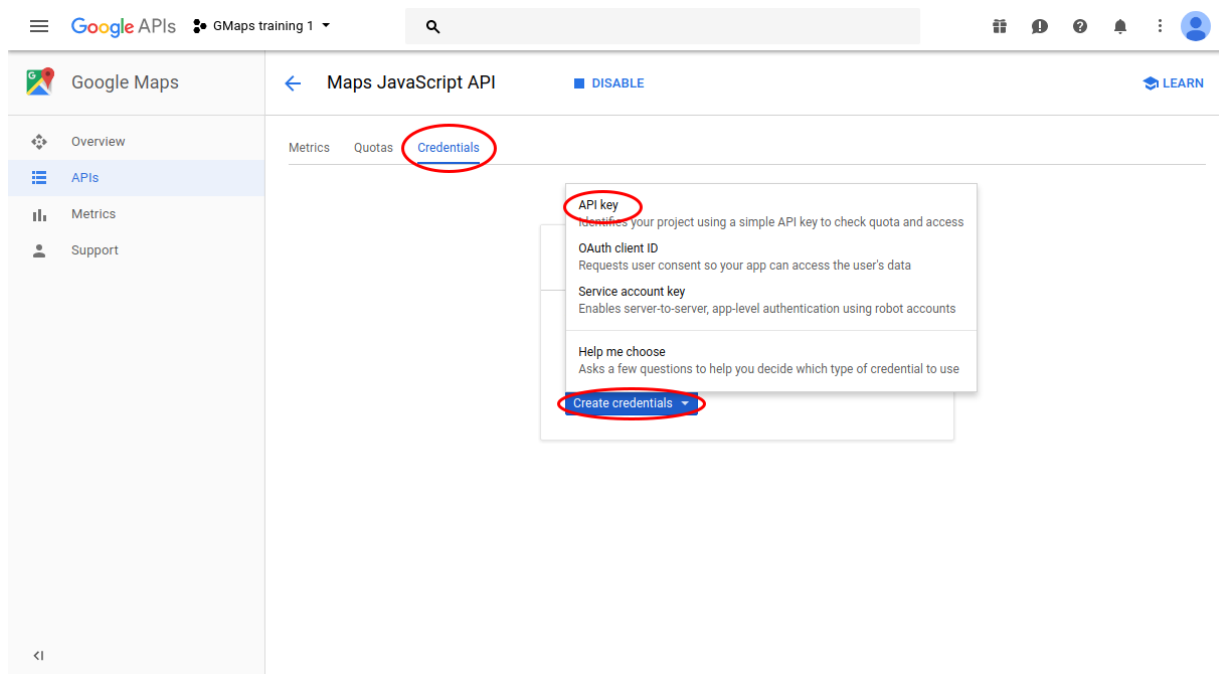


Na stronie tego API kliknij guzik *Enable*.





Po paru chwilach wczyta się strona uruchomionego API. Przejdź na niej do zakładki *Credentials*, a następnie kliknij guzik *Create credentials* i wybierz *API key*.



Następnie wyświetli się Twój klucz API do Google Maps JavaScript API. Skopiuj ten klucz, za chwilę będzie nam potrzebny.

Warto w tym momencie wspomnieć, że jest to klucz którego będziemy używać tylko do nauki, dlatego nie musimy go w żaden sposób zabezpieczać. Nawet jeśli ktoś inny z niego skorzysta, w najgorszym wypadku wyczerpie nasz limit 25 tysięcy zapytań dziennie. Jeśli jednak w przyszłości będziesz używać klucza API na publicznej, działającej stronie, możesz przejść do szczegółowej konfiguracji. Dzięki niej możesz ograniczyć wykorzystanie Twojego klucza np. do konkretnej domeny.



Mamy klucz API, możemy zacząć implementację mapy!

---

## Wstawienie mapy na stronie

---

Zamieszczenie mapy na Twojej stronie nie jest bardzo trudne - możesz zrobić to samodzielnie, z pomocą [dokumentacji](#).

Względem przykładu z dokumentacji, będziemy chcieli wprowadzić kilka zmian w naszym kodzie:

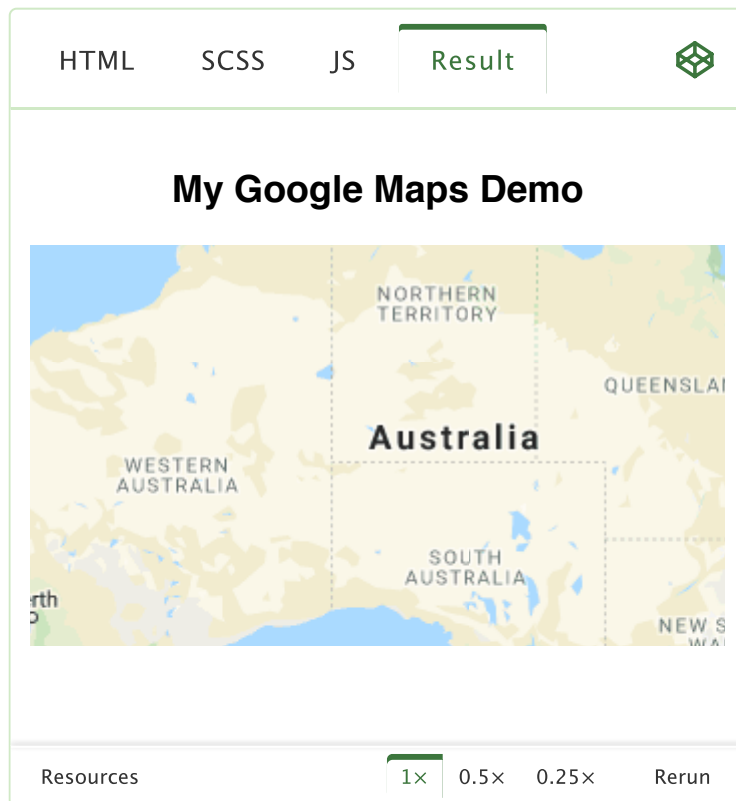
1. Funkcję `initMap` przeniesiemy do naszego kodu JS. Ważne jest jednak, aby zmienić sposób jej zadeklarowania na `window.initMap = function() {...};`. Funkcja ta musi być dostępna w zakresie globalnym, co możemy osiągnąć właśnie za pomocą zapisania jej jako właściwość obiektu `window`.
2. Style znajdujące się w tagu `<style>` przykładu przeniesiemy do naszego kodu `.scss`.
3. Jeśli implementujesz mapę w swoim projekcie, upewnij się, że poniższe odwołanie do skryptu znajduje się w kodzie HTML poniżej odwołania do `js/script.js`, aby funkcja `initMap` na pewno została stworzona zanim wczyta się skrypt z serwerów Google.

```
<script async defer src="https://maps.googleapis.com/maps/api/js?key=
```

Jeśli wszystko poszło zgodnie z planem, mapa powinna być zaimplementowana podobnie, jak na poniższym przykładzie.





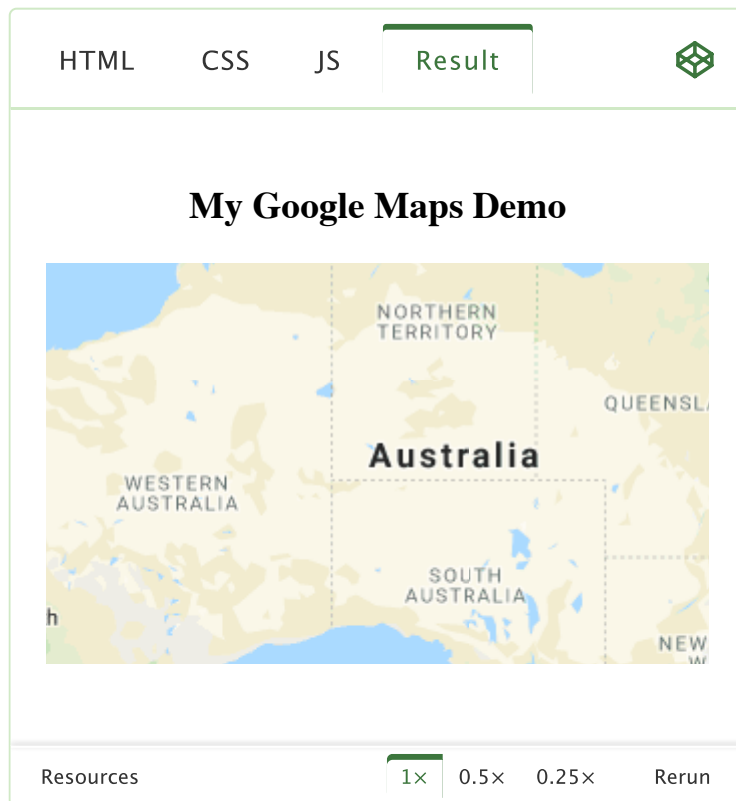


---

## Akcja na kliknięcie markera

---

Powtórzymy teraz powyższy przykład, ale dodamy w nim kilka markerów. Do każdego z nich po kliknięciu wyświetli na stronie inny tekst. Ten prosty przykład pomoże nam za chwilę zaimplementować bardzo ważną funkcjonalność w naszym projekcie!



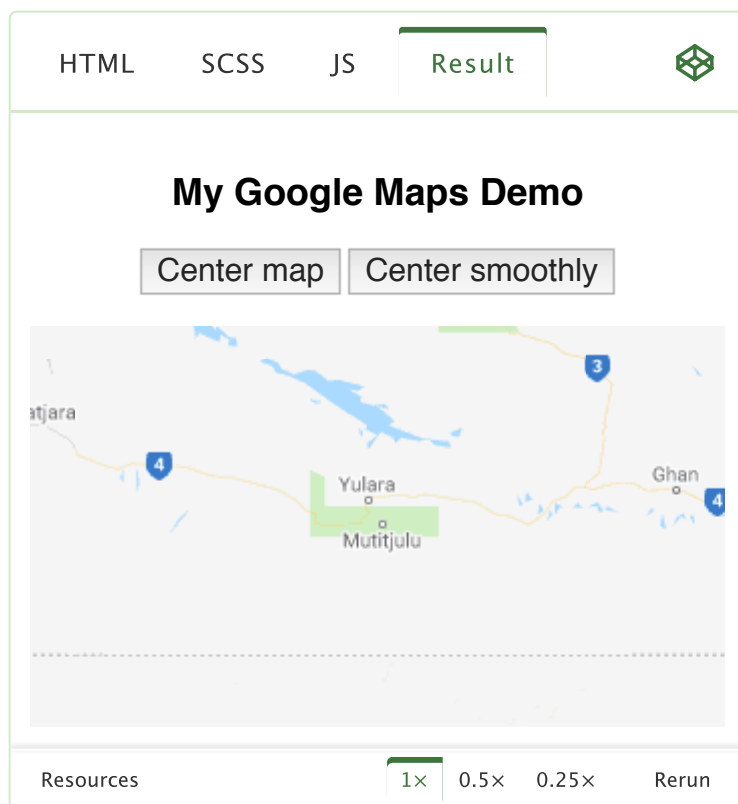
Jak widzisz, akcję na kliknięcie markera przypisujemy prawie identycznie jak do elementów DOM - używamy tylko `addListener` zamiast `addEventListener`.

---

## Przesuwanie mapy

---

Do naszego projektu będziemy też potrzebowali zmiany powiększenia i wycentrowania mapy. Na razie, w poniższym przykładzie, powiążemy te zmiany z kliknięciem guzika.



Docelowo w naszym projekcie zmiany powiększenia i wycentrowania będą wykonywane w reakcji na zmianę aktywnego slajdu w karuzeli. Korzystając z powyższego przykładu na pewno bez trudu poradzisz sobie z implementacją tej funkcjonalności naszego projektu!

---

## Zadanie: Mapa z markerami

---

Jak zwykle, stwórz nowy branch swojego projektu, aby po wykonaniu zadania przestać mentorowi link do tego brancha.

Twoim zadaniem jest zaimplementowanie mapy w Twoim projekcie:

1. Zaczynij od dodania mapy z przykładowym markerem. Ustawienia taką samą wysokości mapy, jaką wysokość ma karuzela. Mapa ma być tuż pod karuzelą. Pamiętaj o krokach, które podaliśmy przy okazji opisywania sposobu wstawiania mapy na stronę!
2. Następnie w tabeli definiującej slajdy (tej w kodzie HTML) dodaj do każdego obiektu właściwość **coords**, a jako jej wartość podaj obiekt z właściwościami **lat** i **lng**, czyli np. **coords: {lat: -25.363, lng: 131.044}**.
3. Teraz zmień początkowe wyśrodkowanie mapy na współrzędne z pierwszego slajdu.
4. Pozostaje jeszcze napisać pętlę, która zamiast przykładowego markera doda po jednym markerze dla każdego slajdu.

I to wszystko - udało Ci się zaimplementować mapę w projekcie!

## Wskazówka

Aby nauczyć się sposobu na pobieranie współrzędnych konkretnego miejsca (lub znaleźć narzędzie które to ułatwia), wpisz w Google frazę "google maps how to get coordinates". Ćwicz w ten sposób samodzielne znajdowanie rozwiązań dla napotkanych problemów - warto trenować tę umiejętność, ponieważ jest bardzo ceniona wśród developerów!

Podgląd zadania

<https://github.com/0na/>

Wyślij link ✓

## 10.4. Połączenie dwóch pluginów



Podsumujmy co dotychczas zrobiliśmy w tym module:

1. Podłączyliśmy karuzelę Flickity opierając się na dokumentacji tego pluginu. Dzięki temu znacznie łatwiej poradzisz sobie w przyszłości z czytaniem dokumentacji i implementacją innych pluginów.
2. Nauczyliśmy się korzystać z szablonów HTML za pomocą pluginu Mustache.js. Dzięki użyciu go do wygenerowania slajdów, nasz kod JS nie zawiera już fragmentów kodu HTML, ani treści wyświetlanych na stronie.
3. Poznaliśmy implementację Google Maps z klikalnymi markerami. Teraz już umiesz wstawić mapę na stronie i manipulować jej wycentrowaniem i powiększeniem za pomocą kodu JS.

Do zakończenia naszego projektu brakuje już tylko połączenia karuzeli i mapy!

## Zadanie: Powiązanie karuzeli z mapą



Ponownie stwórz nowy branch w repozytorium swojego projektu, aby przestać go do sprawdzenia po ukończeniu zadania.

## Etap 1 - zmiana slajdu po kliknięciu w marker

Znajdź w swoim kodzie JS pętlę, za pomocą której dodajesz markery na mapie. W tej pętli, po dodaniu markera, dodaj akcję na zdarzenie kliknięcia tego markera.

Wewnątrz tej akcji ma wykonywać się bardzo podobny kod do tego, który wykonuje się w reakcji na kliknięcie guzika "Reset". Jedyna różnica, to że zamiast do slajdu o indeksie `0`, karuzela ma się przewijać do slajdu o indeksie `i` (zakładając, że Twoja pętla używa zmiennej `i` jako iteratora).

## Etap 2 - wycentrowanie mapy po zmianie aktywnego slajdu

Znajdź fragment swojego kodu JS, w którym animujesz pasek postępu karuzeli w reakcji na event `scroll`. Tym razem napiszesz bardzo podobny kod, z tym że reagujący na event `change`. Sprawdź przykład działania tego eventu w [dokumentacji](#) i wykorzystaj go, aby zmieniać centrowanie mapy.

Zwróć uwagę, że kolejność slajdów jest identyczna, jak odpowiadających im obiektów w tablicy zdefiniowanej w kodzie HTML. Funkcja reagująca na event `change` otrzymuje informację o indeksie slajdu, który został aktywowany.

Pozostaje więc tylko znaleźć odpowiedni element we wspomnianej tablicy obiektów i wycentrować mapę na współrzędnych zapisanych w tym obiekcie.

Samodzielnie zdecyduj, czy przy zmianie wycentrowania chcesz również zmieniać powiększenie mapy, oraz czy chcesz skorzystać z naszej funkcji animującej zmianę powiększenia i wycentrowania.

## Podsumowanie

Jeśli wszystko poszło sprawnie, to:

1. Niezależnie od sposobu zmiany slajdu, mapa wyśrodkuje się na markerze powiązanym z tym slajdem.
2. Po kliknięciu markera zmieni się aktywny slajd w karuzeli, a co za tym idzie mapa wyśrodkuje się na markerze powiązanym z tym slajdem (czyli na klikniętym markerze).

## Dla bardzo ambitnych - powstrzymanie wycentrowania mapy po kliknięciu w marker



Ta część zadania jest **nieobowiązkowa i trudna**. Jeśli jednak masz ochotę zmierzyć się z tym problemem, wyznacz sobie ile czasu możesz na to poświęcić i do dzieła!

Jak napisaliśmy w podsumowaniu, kliknięcie markera powoduje zmianę aktywnego slajdu, co z kolei powoduje wycentrowanie mapy.

Chcielibyśmy jednak, aby po kliknięciu markera mapa w ogóle nie zmieniała wyświetlanej pozycji ani powiększenia.

Spróbuj wymyślić jakiś sposób - choćby nawet był bardzo nieelegancki - na to aby mapa centrowała się po każdej zmianie aktywnego slajdu, poza sytuacją kiedy aktywny slajd został zmieniony w skutek kliknięcia markera.

Pamiętaj, aby zaplanować ile czasu chcesz spędzić nad tym problemem! To jedna z tych zagwozdek, które wydają się niemożliwe do rozwiązania, mimo że kiedy już wpadniesz na rozwiązanie, będzie Ci się ono wydawało oczywiste. ;)

Uważaj też, aby nie zepsuć innym wyzwania - nie podawaj swojego rozwiązania nikomu poza mentorem!

Powodzenia!

Podgląd zadania

<https://github.com/0na/>

Wyślij link ✓

## 10.5. Podstawy jQuery



Wspomnieliśmy sobie już na początku tego modułu o jQuery. Przypomnijmy, do czego służy ta biblioteka: jQuery upraszcza i usprawnia najbardziej potrzebne web developerowi akcje, takie jak manipulowanie elementami DOM i klasami CSS.



Wspomnieliśmy też, że obecnie większość z tych ułatwień można łatwo zastąpić czystym kodem JS. Biorąc pod uwagę, że każda biblioteka opóźnia nieco wyświetlenie strony, zwiększa jej rozmiar i często wydłuża działanie skryptów które z niej korzystają — lepiej jest nie używać jQuery bez wyraźnej potrzeby.

Mimo tych wad, wiele pluginów, a co za tym idzie wiele stron, opiera się o jQuery. Z tego względu zajmiemy się przez chwilę jQuery, aby czytanie skryptów wykorzystujących tę bibliotekę nie stanowiło dla Ciebie większego problemu.

---

## Przykład kodu jQuery

---

W czystym JavaScriptcie, aby odwołać się do elementu DOM o danym id, należy użyć następującej konstrukcji:

```
var item = document.getElementById('item');
```

jQuery znacznie upraszcza ten proces — za jego pomocą to samo wyrażenie zapiszemy tak:

```
var item = $('#item');
```

Jak widzisz, różnica w długości zapisu jest spora.

W kolejnych częściach tego modułu powiemy sobie więcej o podstawowych funkcjach jQuery, a zaczniemy od odpowiedzi na pytanie, czym właściwie jest jego charakterystyczny znak dolara.

---

## Magiczny znak dolara

---

Mówiąc w skrócie, znak dolara w jQuery wyszukuje i zwraca nam selektor (lub selektory) o żądanych parametrach. Oznacza to, że zapis `$('.item');` odczytamy jako *wszystkie elementy posiadające klasę "item"*.

Zamiast znaku dolara, możesz spotkać się też z zapisem `jQuery('.item');` — oznacza on dokładnie to samo, dolar jest po prostu krótszą i wygodniejszą konwencją.



### Dla ciekawskich - co zwraca `$('.item')` ?

Technicznie rzecz biorąc, `$` jest nazwą funkcji, zwracającej pewien obiekt. Ten obiekt jest w swojej budowie podobny do tablicy - tak samo jak wynik wyrażenia

`document.querySelectorAll('.item')` .

Różnica jest jednak znacząca - jQuery do tego wynikowego obiektu dodaje bardzo dużo własnych metod, które pozwalają na wykonywanie wielu operacji na znalezionych w ten sposób elementach DOM.

---

## Podstawowe metody

---

Koniec teorii, czas na praktykę! Zobaczmy w końcu jQuery w akcji. W tym module, tak jak i w poprzednich, będziemy używać CodePena do demonstracji przykładów kodu. Jeżeli natomiast chcesz użyć jQuery na swojej stronie internetowej, zajrzyj do [tego tutoriala](#).

## Dodawanie, usuwanie i przełączanie klas

Za pomocą jQuery możemy łatwo manipulować klasami elementów — dodawać je, usuwać i przełączać.





HTML SCSS JS Result

```
1 'use strict';
2 (function(){
3
4  /* Poćwiczmy
   jQuery! Poniższe
   przykłady pokażą
   Ci, w jaki
   sposób:
5  - przypinać
   akcję do eventu
   (klik),
6  - zmieniać
   klasy selektorom
   DOM,
7  - zmieniać
   treść elementów
   na stronie.
8
9  Na początku
   sprawdzimy, by
   pierwszy
   nagłówek stał
   się czarny po
   kliknięciu
   buttona poniżej.
10 */
11
12 $('h1.remove-
   green').on('click', function() {
```

# 1. Hello World!

Click the button below to make the header black.

Make header black

# 2. Hello World!

Click the button below to make the header green.

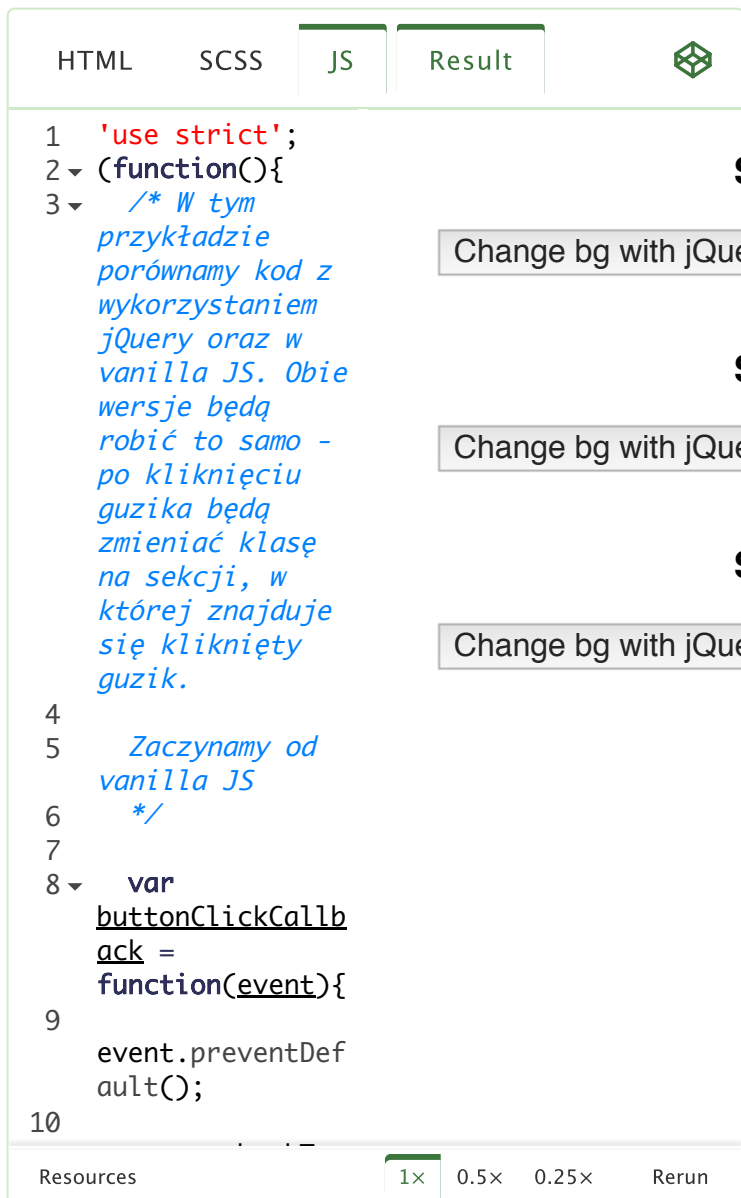
Make header green

# 3. Hello World!

Resources 1x 0.5x 0.25x Rerun

## Trawersowanie drzewa DOM

Kolejną przydatną metodą jQuery jest poruszanie się po drzewie DOM.



```
1 'use strict';
2 (function(){
3   /* W tym
   przykładzie
   porównamy kod z
   wykorzystaniem
   jQuery oraz w
   vanilla JS. Obie
   wersje będą
   robić to samo -
   po kliknięciu
   guzika będą
   zmieniać klasę
   na sekcji, w
   której znajduje
   się kliknięty
   guzik.
4
5   Zaczynamy od
   vanilla JS
6   */
7
8   var
   buttonClickCallb
   ack =
   function(event){
9     event.preventDefault();
10  }
```

## Więcej? W dokumentacji!

W tym rozdziale omówiliśmy tylko kilka bardzo podstawowych komend jQuery. Pełna lista znajduje się w [dokumentacji](#) — warto z niej korzystać, gdyż zawiera użyteczne przykłady kodu i wyjaśnienia.

Mnóstwo informacji o jQuery wraz z gotowymi kawałkami kodu znajdziesz również na portalach typu [StackOverflow](#). Wokół jQuery zgromadzona jest duża i aktywna społeczność (jQuery to w końcu jedna z najbardziej popularnych bibliotek webowych!), więc najprawdopodobniej odpowiedzi na Twoje pytania już na Ciebie czekają ;)

## Podsumowanie - czy używać jQuery?



Uproszczenia pokazane w powyższych przykładach sprawiają, że jQuery jest często używany przez developerów, ponieważ ułatwia i przyspiesza pisanie skryptów JS. Czy jest to wystarczająco dobry powód - światowe trendy pokazują, że z czasem ten argument jest coraz mniej przekonujący.

W naszym kursie nauczysz się tylko podstaw jQuery, ponieważ wykorzystywanie jego wygodnych "dróg na skróty" sprawia, że wolniej nabiera się dobrych nawyków, mniej rozumie się kod JS, a co za tym idzie, jest się słabiej przygotowanym na rozpoczęcie nauki frameworków, takich jak np. ReactJS.

Pamiętaj też, że w kodzie opartym na jQuery możesz swobodnie używać vanilla JS! Za to developer, który zawsze opiera swoje skrypty o jQuery, nie odnajdzie się tak łatwo w vanilla JS. Dobrze o tym wiedzą Twoi przyszli pracodawcy, więc nie krępuj się o tym mówić otwarcie!

Dlatego właśnie radzimy ograniczać użycie jQuery do niezbędnego minimum. Korzystaj z jQuery kiedy potrzebujesz pluginu opartego o tę bibliotekę, który nie ma **zamiennika w vanilla JS**. Podobna sytuacja z którą możesz się spotkać, to zadanie modyfikacji skryptu wykorzystującego jQuery. Zachęcamy jednak, aby nawet w tych wypadkach kod pisany przez Ciebie od podstaw był w vanilla JS. Dzięki temu, jeśli kiedykolwiek zechcesz zrezygnować z jQuery, będzie to znacznie prostsze.

Mimo tego, że nie będziemy zgłębiać tajników jQuery, już znajomość powyższych przykładów i wykonanie poniższego zadania pozwoli Ci zapoznać się z jQuery na tyle, że bez większych problemów poradzisz sobie z czytaniem kodu opartego o jQuery.

---

## Zadanie: Akordeon oparty na jQuery

---

Na bibliotece jQuery opiera się również biblioteka jQueryUI, która posiada kilka przydatnych widgetów i metod. W tym zadaniu przyjrzymy się jednemu z widgetów - **akordeonowi**.

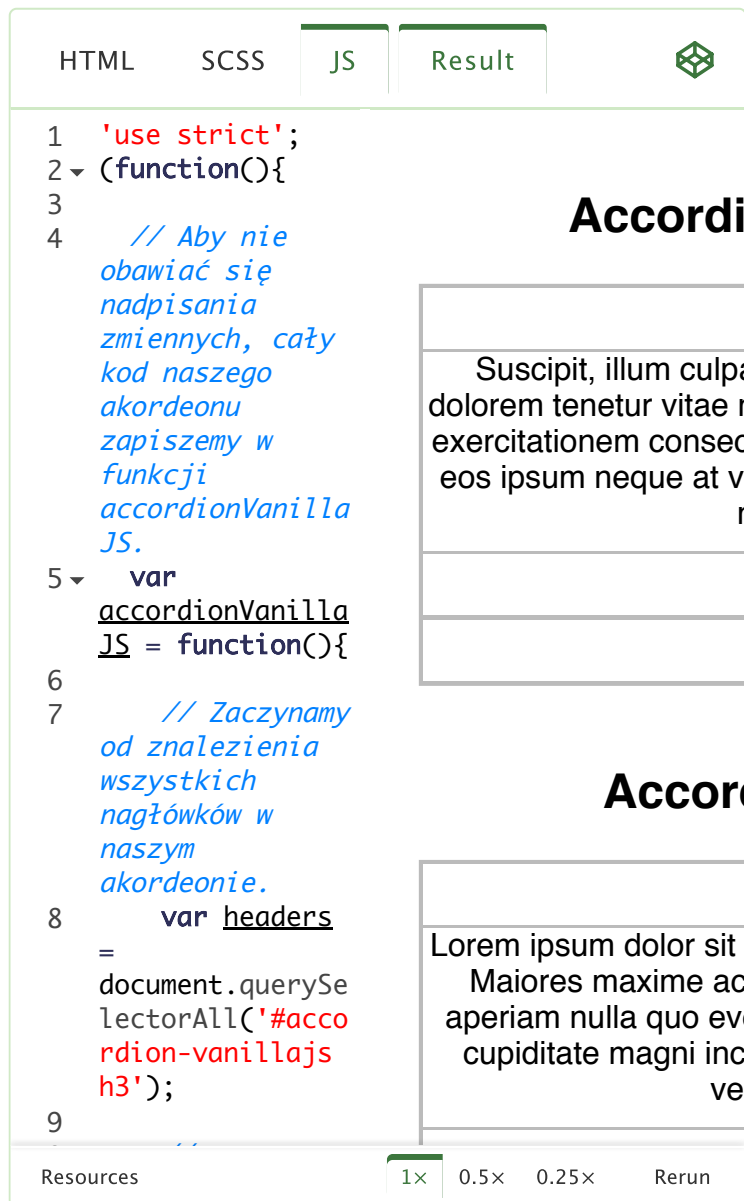
Jak widzisz w przykładzie na stronie jQueryUI, akordeon to zestaw nagłówków i przynależących do nich treści. Po kliknięciu w nagłówek otwiera się przypisana do niego treść, a wszystkie inne treści są ukrywane. W podstawowym przykładzie wszystkie treści mają tę samą, stałą wysokość.

Możesz zrealizować to zadanie jako projekt w GitHubie lub jako nowy pen na Twoim CodePenie. Pamiętaj, że zmiany które wprowadzisz w zagnieżdżonym poniżej edytorze nie będą zapisane!



Twoim zadaniem jest napisanie skryptu obsługującego akordeon. Oprzyj swój kod o jQuery, ale bez wykorzystania biblioteki jQueryUI.

Dla ułatwienia w poniższym przykładzie napisaliśmy skrypt vanilla JS, który działa bardzo podobnie do przykładu z powyższego linka. Przygotowaliśmy też miejsce na Twój kod, który będzie realizacją tego zadania.



Do realizacja tego zadania będziesz potrzebować następujących metod jQuery:

- `.click(function(){ /* ... */ })` ,
- `.addClass('class')` ,
- `.removeClass('class')` ,
- `.siblings('element#id')` .

Znajdź je w [dokumentacji](#) i przeczytaj przykłady ich zastosowania. Poza tymi metodami będzie Ci potrzebne wyłącznie wyszukiwanie elementów za pomocą selektora – używaliśmy go w każdym z powyższych przykładów zastosowania jQuery.



Świetne rozwiązanie tego zadania zajmuje 3 (słownie: trzy) linie kodu dodane w funkcji **accordionjQuery**, ale nie przejmuj się jeśli Twoje rozwiązanie zajmie nawet 4 (słownie: cztery) linie kodu. ;)

Nie przyzwyczajaj się jednak za bardzo do wygody jaką daje jQuery - za chwilę wracamy do vanilla JS, aby wyszkolić Cię na wszechstronnego web developera!

Powodzenia!

Podgląd zadania

<https://0na.github.io/jQuery>

Wyślij link ✓

## 10.6. Dodatkowy projekt dla ambitnych



Kolejny dodatkowy projekt dla ambitnych :)

Zasady są nadal takie same:

- Projekt jest NIEOBOWIĄZKOWY
- Wykonując go, zdajesz się na umiejętności które zdobyłeś/aś w czasie kursu
- Możesz korzystać jedynie z pomocy społeczności na czacie.

Jak wiadomo, chodzi o nabranie praktyki przy cięciu szablonów. Czyli zadaniu którym Front-endowcy zajmują się najczęściej.

Jeżeli nie chcesz wykonywać tego projektu to po prostu kliknij w przycisk i o nim zapomnij ;)

Jednak jeżeli chcesz poćwiczyć to Twoim zadaniem jest:

1. Utworzenie nowego projektu na GitHubie.
2. Zakodowanie layoutu na podstawie grafiki.
3. Uzupełnienie treści, mogą być sztuczne (wypełniacze).



4. Zachowanie dobrych praktyk przedstawionych w dotychczasowych treściach.
5. Wykorzystanie Bootstrapa.
6. Przystosowanie strony do odpowiedniego wyświetlania na urządzeniach mobilnych.

Poniżej znajdziesz link do grafiki do pobrania w formacie **.png**:

[Link do grafiki](#)

Poniżej znajdziesz link do grafiki do pobrania w formacie **.psd**:

[Link do pliku .psd](#)

W submodule 3.4 znajdziesz tutoriale, które pomogą Ci odwzorować szablon korzystając z plików .psd.

Powodzenia! :)

Jeśli masz wątpliwości do powyższego materiału, to - zanim zatwierdzisz - zapytaj na czacie :)

✓ Zapoznałem się!

[Regulamin](#)

[Polityka prywatności](#)

© 2019 Kodilla

