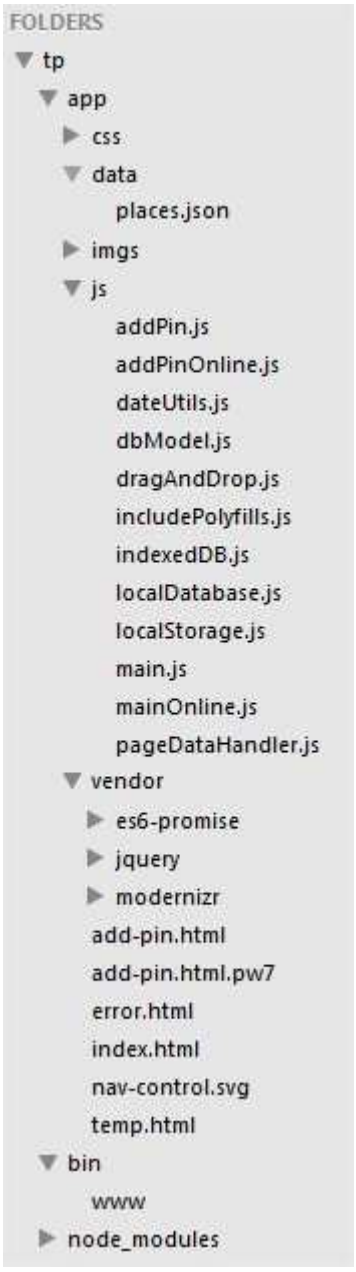
Informations sur les TPs

Sequence

PWs are based on a single application. Each new PWs add features to this application based on theoretical courses which are given just before. The advantage is that it allows you to work with the same resources during all the formation, and gives consistency to the whole project. The disadvantage is that PWs depend on each other. So, if you haven't completed your PW, don't hesitate to ask for the corrected version to the former.

Project Structure



Each of the files you will have to work with are in the app folder.

Javascript:

Our custom javascript files (the ones you will have to implement) are in the 'js' folder. Other ones (libs/tools) are in the 'vendor' folder.

Html:

Html files are at root. The interesting ones for you are index.html and add-pin.html.

Installation

In oder to work on this project, you need to:

1) Install nodejs (http://nodejs.org/)
2) Set the proxy for npm (executable provided with nodejs):
    a) Use the HTTP_PROXY and HTTPS_PROXY environment variables (for exemple HTTP_PROXY= http://prx-dev02:3128)
3) Run this command: npm install
4) Run this command: npm install –g bower
5) Run this command: bower install
6) Get a recent browser:
    a) Last firefox version
    b) Chrome canary (Standard Chrome is not enough when these lines are wrote)
7) Install the provided old portable version of Firefox
    a) WARN: Don't install any updates

Usage

To launch the server and make the application accessible:

```
node bin/ww
```

To access the application, go to url:

```
http://localhost:3000
```

PW1 – Ajax and DOM with jQuery

Let's use jQuery in order to dynamize our application by:

- loading the needed data to display our images and their metadata (author, title, description …).
- Inserting this data in the DOM

Part 1

1) Include *jquery* in your *index.html*. The file path is *vendor/jquery/dist/jquery.min.js*
2) Include also *dateUtils.js*, *mainOnline.js* and *main.js*
3) In *mainOnline.*js, update the *getPins* function by adding an ajax method which retrieves a json object from the JSON file available in *app/data/places.json*
   a) **Note**: the url to get the JSON object is not the same as the path of the JSON file defined on the server side, check *app.js* to find the good url)
   b) Jquery documentation : *http://api.jquery.com/jquery.getjson/*

Part 2

4) In *main.js*, find the call to *getPins*, then update the associated callback function to treat the retrieved json. This callback will have to:
   a) Set the *places* variable with the data retrieved from the ajax call
   b) Iterate over the data, and call the method *getPlaceFromTemplate* for each element of the array (*getPlaceFromTemplate* will be implemented after)
      (1) **Note**: Have a look at the *jQuery.each* function
   c) Append the result of the *getPlaceFromTemplate* to the DOM element whose id is *pinsWrapper*.
      i) **Note**: Have a look at the *jQuery.append* method
   d) Implement the function *getPlaceFromTemplate(place)* which returns the template of a place, fulfilled with the data in parameter (use the *dateUtils* object provided by *dateUtils.js* when setting dates in the template)
      i) **Note**: The *getPlaceFromTemplate* method returns a String which corresponds to the HTML of a place
      ii) **Note**: Here is a tool which converts HTML to a javascript String: **http://www.javascriptkit.com/script/script2/jstohtml.shtml**
      iii) **Note**: Look for the *<article>* element in your index.html application in order to get the the static html that you now have to dynamize

iv) **Note**: The static content of the '#pins-wrapper' has to be removed…

e) Create a jQuery plugin and use it to do the work:

i) The plugin function will allow to use the custom parameter 'error', which is the callback to use when an error occurs during the ajax call…

PW2 – Ajax Submit Form –jQuery Selectors

Let's see how to submit a simple form with the help of jQuery

1) Add the *jquery.js*, *addPinOnline.js* and *addPin.js* in *add-pin.html*
2) In *addPin.js*, fulfill the method *extractInfo* to:
   a) Return a pin object which contains all needed fields to create a new pin (name, category, …)
   b) This pin object has to be populated with data available in the form element
3) In the submission form process, you can see a call to the *addPinInfo* function. Have a look at this function in order to:
   a) Make a redirection to index.html in case of success
   b) Log an error in the console in case of failure (*console.log(…)*)

PW3 – Polyfills

Let's use modernizr and polyfills to make the application compatible with an older browser version with the help of jQuery

1) Launch the application with the provided browser. As you can see, the rendering is not correct. This is because of the 'flexbox' functionality which is not supported on this browser (the *'flex-flow*' and 'display:flex' attribute on the '*#pinsWrapper*' element). So let's use *modernizr* to know if our browser supports flexbox or not and include add a new css in case it's not
   a) In *index.html*, import *includePolyfills.js* and *modernizr.custom.18437.js* , just after *jquery.js*
   b) In *includePolyfills.js*, use *modernizr* to load *'css/noFlexbox.css*' style if the *flexbox* functionality is not available. If the rendering is correct then you're right.

c) Have a look at *noFlexbox.css* to understand how the problem has been solved

2) In the future PWs, let's have to work with promises, which is a *javascript* feature. There are several implementations, as the jQuery one (http://api.jquery.com/category/deferred-object/) or the *angularjs* one (https://docs.angularjs.org/api/ng/service/$q). But we are not working on *angularjs*, and for the PW needs, we don't want the *jQuery* one anymore. So let's use the standard one, which comes with recent versions of javascript. Sure, our old browser doesn't know promises so let's have to use a polyfill.
   a) Still in *includePolyfills.js*, fulfill the *includePromisesPolyfill* method:
      - Check that promises are available in the browser (*window.Promise* is defined)
      - If promises are not supported, use jQuery to create the following script tag and insert it dynamically just after the body: <script type="text/javascript" src="vendor/es6-promise/promise.js" />

PW4 – ContentEditable HTML5 functiunality

The aim of this PW is to show how easy it is to make something editable and to persist this data.

1) Make the comments editable by using the appropriate html5 attribute (you have to modify the template started from the previous PW)
2) Have a look at the *clickOnSaveButton* function and be sure that it is called when a user click on the 'save' button (nothing to do here, just understand the mechanism)
3) Fulfill the *updateContent* function in *mainOnline.js* and complete it to:
   a) Post the place to the '/api/places' url
   b) On the post 'success' callback, let's add a visual highlight effect:
      i) Remove the class 'saving' from the element in parameter
      ii) Then add the class 'saved' on the element in parameter
      iii) Wait 200 milliseconds and remove the class 'saved'

PW5 –Ajax submit form with upload – FormData API

Let's add an input file field in the form which will be used to upload a file and use the *FormData* API

1) In *add-pin.html*:
   a) Uncomment the bloc containing an input file which will be used to upload an image
   b) Add the html5 'progress' tag near to this input file, let's use it to display the upload progress… [TODO: Ca serait bien de ralentir l'upload, pas encore réussi]
2) In *addPin.js*:
   a) Comment your *extractInfo* function
   b) modify the submission process (form.submit() content) in order to use the *FormData* api instead of *extractInfo,* and use the *addPinImage* function instead of *addPinInfo*
   c) Dynamize the progress bar by using the third parameter of the *addPinImage* function
3) Still in addPin.js, have a look at the *inputFileValidation* function and fulfill it according to comments and following rules:
   a) Check that the filename is not more than 50 chars
   b) Check that the file is a jpeg image
   c) Check that the file size is less than 1mo
   d) Display a message if the constraints are not satisfied, and reset previous values in this case.

   Note: Remember that some javascript behaviors could be modified by the end user, so in a production environment, this validation process should also be done on the server side

PW6 –Cache API

Let's use the cache api in order to cache images in our application.

1) In *index.html*, activate the cache functionality
   a) **Note**: The manifest is generated dynamically, so you will not find it on the FS. Its name is *application.manifest*.
2) Define the environment variable FULL_MANIFEST and set it any value, then restart node.

3) Look at the manifest content, and make some experimentations to be sure everything works fine
    a) Check your cache entries with the development console of your browser
    b) Be sure that the cached files are not downloaded anymore (check the network exchanges in your development console…)
    c) Be sure that when the manifest file changes, the files are downloaded and updated in the cache
        i) Note: To modify the manifest, use the environment variable MANIFEST_VERSION and restart node (for example, MANIFEST_VERSION=v2)

Note: Remember that from this step, your *index.html* file is cached. When you will have to modify it in next PWs, you will have to change the manifest version as well to uncache it or **just disable the cache functionality by removing the manifest attribute in the html tag**.

PW7 – localStorage API

Let's use the locaStorage api in order to store user preferences, which are in this case the application layout parameter

4) In *index.html*, import *localStorage.js* and uncomment the *'#navigation-status'* element
5) Look for the *localStorage.js* file content, and add implementation based on comments
    a) **Note**: the css class, the value in the localStorage and the text associated to the element have all the same value ('vertical' or 'horizontal')
    b) **Note**: *updateLayout* just set the layout in parameter in the localStorage, while *getCurrentLayout* retrieves it (use the key of your choice), and *getOpposedLayout* get the opposed one as its name says

PW8 – IndexedDB API & promises

Let's do the following:

- Store places data in a local DB when the application starts

- Use this local DB in main.js instead of doing ajax calls
- Use promises, which will help to get a more readable code, because *indexedDB* api is mostly based on callbacks

Remarks:

- Promises normalized implementation is available with newer versions of js only (1.6), which is not really a problem because indexedDB is not so old ;-)

1) In *index.html*, import *indexedDB.js* before *main.js* and after *mainOnline.js*
2) In *main.js*, assign *indexedDBModule* to the module variable instead of *mainOnline*
3) In *main.js*, look for the methods called on the *module* variable, and be sure that they are correctly implemented in *indexedDB.js*
4) In *indexedDB.js*, look for the code around onupgradeneeded, and be sure that the createAndInsertData method is correctly implemented

PW9 – Drag & Drop API

Let's do the following:

- Use the drop api to display an image in the concerned zone when it is dragged from the file system
- Display this image as well when it is selected with the concerned button
- Make the necessary to be sure that the file is uploaded in both cases

1) To prepare the environment:
   a) replace the *add-pin.html* content by the *add-pin.html.pw7* content
      i) **Note**: Main differences are the inclusion of the *dragAndDrop.js* file and the fact that the input file element is not anymore in the form element. The reason is that now, we are not sure that the file comes from the input field (when the image is dropped, the input field is not used).

2) In *dragAndDrop.js*
   a) Implement the *addDropFunctiunality* function by following the comments
      i) **Note**: The aim of this function is to treat the 'drop' event to call the method which will display the image
      ii) **Note**: To disable the default behavior of an event, use its method *preventDefault()*, and use the *stopPropagation()* one as well to be sure that the event will not be propagated
   b) Implement the *captureInputFiles* function by following the comments
      i) **Note**: The aim of this function is to treat the 'change' event on the input file element to call the method which will display the image
3) Complete the *addPin.js* file to take into account the file to upload. As said in 1)a)i) , the input file is not in the form, so you will have to add the data manually during the form submission by:
   a) Retrieving the file you need to upload (look for the *retrieveFile* function in *dragAndDrop.js* to know where it is stored)
   b) Append this file with the *FormData* api

PW10 – Geolocation API

Let's use the geolocation API to display a map (with help of google maps api).

1) In *add-pin.html*:
   a) Import *http://maps.googleapis.com/maps/api/js* and *geolocation.js*
2) In *geolocation.js*, function *setMessage*
   a) Create a new div element
   b) append it to the end of the element <main role="main" />
   c) Assign it the class locOpts.divInfoClass
   d) Set the text in parameter to this div
3) In *geolocation.js*, function *positionError*
   a) Define a message depending on the error code
   b) Display this message using the *setMessage* function
   c) cancel the geolocation API process if possible (maybe that the API was working at the begining and that an error occured later, in which case we want to stop the geolocation feature)

4) In *geolocation.js*, at the end of the file, read comments to initialize the geolocation API and use it with previous functions
5) Test you dev, be sure that an error message is displayed when an error occurred

PW11 – offline API

In the first part, let's just alert the user when its network status changes.
And to go further (optional), let's make the application accessible offline

1) Update the online/offline element on the top right of the window accordingly to the network status (set the text and the class to 'online'/'offline')
2) Be sure also that the status is correctly initialized when the page is loaded
3) To go further (**optional**), implement everything to make work your application offline.
   a) You have to make your *add-pin.html* page accessible offline
   b) Then, be sure that your ajax calls (during the form submission) are deferred and will wait the online event to be really processed. You can do this by using a simple array containing the data to be processed later.
      i) **Note**: Be careful with the callbacks (if you make 2 submissions offline, then you have also 2 ajax request waiting for the 'online' event. If the success callback of the first request redirects to 'index.html', the second request will never be processed…)
4) You still want more (**optional**)?
   a) Don't use a simple array to keep your data, but use indexedDB instead, so you can close the application even if it is offline, restart it, and your requests will be processed when the 'online' status is back.