IHM – Application HTML/CSS/TS : préparation et exécution

Développement

étape 1 : préparer le projet

Architecture d'une application Web

Sur votre « devhom », emplacement où se trouve vos applications web sur le serveur « devweb » de l'IUT de Metz

1. **créer un dossier « ihm »**
2. **créer dans le dossier « ihm », un dossier par application ou projet :** *pour notre exemple, créez le dossier « tp1 »*
3. **copier le contenu du fichier compressé « squelette » à télécharger sur Arche, dans le dossier**

   le fichier « tsconfig.json » et les dossiers « vue », « vue/css », « src/controleur » et « src/modele » apparaissent

   3.1. contenu du dossier « vue »
   - fichiers HTML
   - dossier CSS qui contiendra les fichiers « css » et les éventuelles sources images (fichiers jpg, png)

   3.2. contenu du dossier « src/controleur »
   - fichiers TypeScript (.ts) avec la définition des classes associées aux fichiers HTML du dossier « vue »,

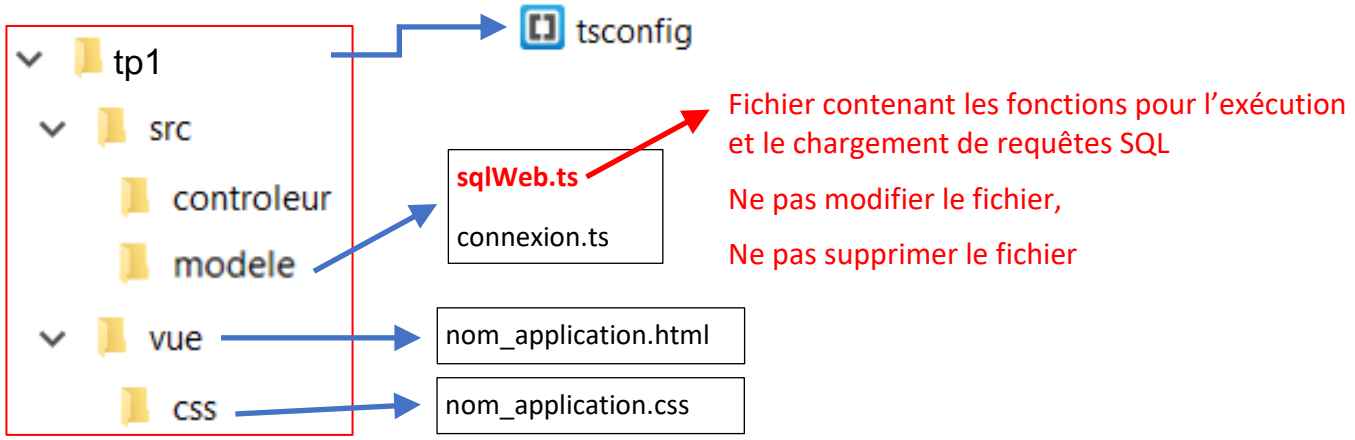
   3.3. contenu du dossier « src/modele »
   - fichiers TypeScript (.ts) et leur conversion javascript (.js), avec la définition des classes pour gérer(lecture, écriture) les données des tables de la base de données

Deux autres dossiers **« controleur »** et **« modele »** contenant la conversion des fichiers « ts » en « js » seront créés lors de la transpilation des fichiers « ts » en « js »

Le fichier « **tsconfig.json** » **copié à la racine du dossier** contient notamment les directives d'interprétation du code TypeScript et de transpilation (conversion du langage TypeScript vers le langage javascript)

NE PAS MODIFIER le contenu du fichier

Exemple : sur votre « devhom », dans le dossier « tp1 » :



IHM – Application HTML/CSS/TS : préparation et exécution

Contenu du fichier « sqlWeb.ts »

Nous découvrirons le contenu de ce fichier lors du développement d'une application avec utilisation d'une base de données.

```
type TtabAsso = {
    [key:string] : string;
}
type TdataSet  = TtabAsso[];


class SQLWeb {
    spExec      : string;
    cheminHTML  : string;
    http        : string;
    bd          : {host:string, port:string, bdname:string, user:string, pwd:string
                   , charset:string, driver:string };

    init(cheminHTML : string, http : string) : void{
        this.spExec      = http +'spExec.php';
        this.cheminHTML  = cheminHTML;
        this.http        = http;
    }

    getXhr(): XMLHttpRequest
    {
        let xhr = null;
        if (window.XMLHttpRequest)  // firefox et autres
        {    xhr = new XMLHttpRequest;  }
        return xhr;
    }

    SQLexec(sp : string, params : string[]):boolean {
        this.SQLloadData(sp, params, 'manipulation');
        return true;
    }
```

```
 SQLloadData(sp : string, params : string[], req ='interrogation'):TdataSet  {
    // fetch ne fonctionne pas en mode synchrone ==> mode synchrone obligatoire
        const xhr = this.getXhr();
        let resultat : TdataSet = [];
        if (xhr)
        {   // on définit ce qu'on va faire quand on aura la réponse
            xhr.onreadystatechange = function():void{
                if (xhr.readyState === 4 && xhr.status === 200)
                {
                    let src = JSON.parse(xhr.responseText);
                    resultat = src['resultat'];
                }
            }
            xhr.open ("POST", this.spExec, false);  // mode synchrone obligatoire
            xhr.setRequestHeader ('Content-Type', 'application/x-www-form-urlencoded');
            for (let i in params) {
                params[i] =encodeURIComponent(params[i]);
            }
            xhr.send('sp=' +encodeURIComponent(sp) +'&bd=' +JSON.stringify(this.bd)
                  +'&params=' +JSON.stringify(params) +'&req=' +req);
        }

        return resultat;
    }

    bdOpen(host :string, port : string, bdname : string, user : string
            , pwd : string, charset ='utf8', driver ='mysql'):void {
        this.bd = {host:host, port:port, bdname:bdname, user:user, pwd:pwd
                   , charset:charset, driver:driver };
        this.SQLloadData("",[]);
    }

}

let sqlWeb = new SQLWeb()
export { sqlWeb, TtabAsso, TdataSet }
```

IHM – Application HTML/CSS/TS : préparation et exécution

Utilisation de Visual Studio Code

1. **ouvrir l'application « visual studio code » sous « Linux » ou « Windows »**

2. **choisir le dossier de travail** créé précédemment (voir « architecture d'une application Web »)

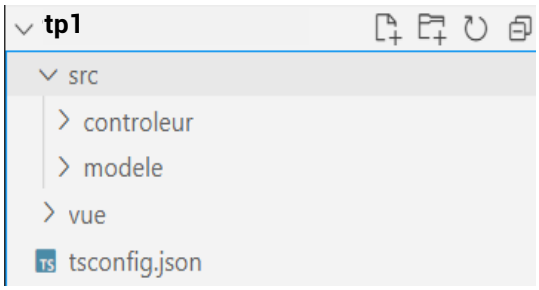
   Fichier → Ouvrir le dossier…

   les dossiers « src », « vue »
   et les sous dossiers de « src » : « contrôleurs et « modele »
   s'affichent

   ainsi que le fichier « tsconfig.json »

   cliquer sur un des dossiers pour voir son contenu

   exemple : choix du dossier « tp_appli »

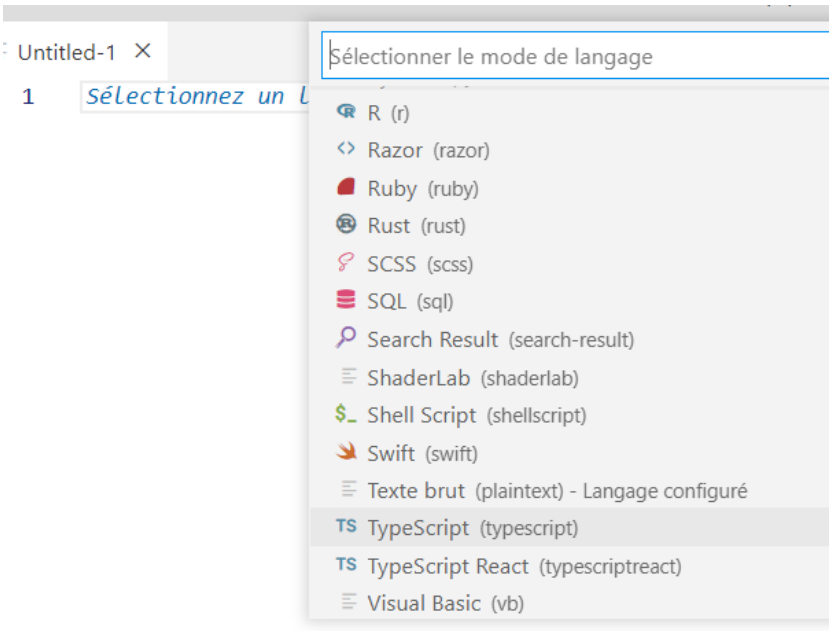


3. **éditer un fichier**
   3.1. cliquer sur un des fichiers pour l'afficher dans l'éditeur

   ou

   3.2. pour créer un nouveau fichier

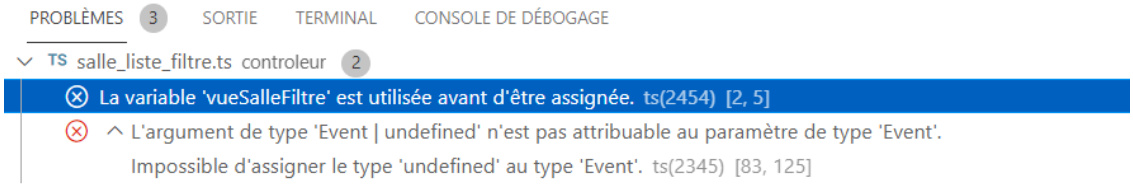
   - Fichier → Nouveau fichier
   - **sélectionner le langage** de programmation souhaité (TypeScript, HTML, CSS, etc)



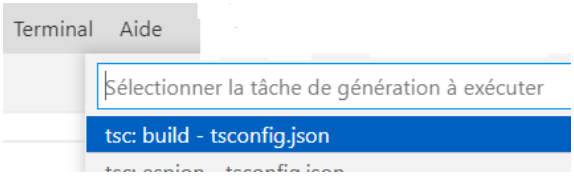
   - taper votre code
   - **enregistrer** votre travail (enregistrement par défaut dans le dossier ouvert dans l'explorateur « Visual Studio Code »)

4. **si votre fichier est un fichier « ts »,** corriger les erreurs et **transpiler** (conversion en javascript pour intégrer le code dans une page HTML)

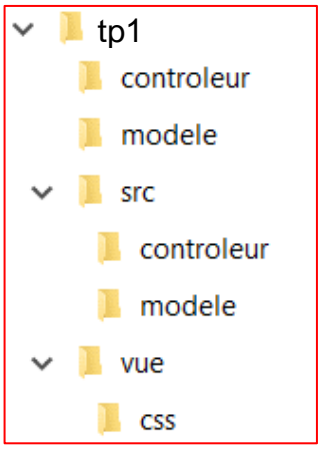
- corriger les erreurs qui apparaissent en bas de l'éditeur



- **Terminal → Exécuter la tache de build …**
- choisir « **tsc build – tsconfig.json** »



- les fichiers « **js** » sont dans les dossiers « controleur » et « modele » du dossier de l'application (le dossier « modele » n'est pas créé pour une application sans base de données)



étape 2 : identifier les zones et les composants de la maquette

Il faut découper la maquette en zones, puis découper chaque zone en sous-zone et ainsi de suite.

Dans chaque, il faut identifier les composants à implémenter (<label>, <input type =’button’> <input type =’text’>, ‘<select>’, …).

Les fichiers « **html** » à créer seront dans le dossier « **/vue** » de l’application.

Les fichiers « **css** » à créer seront dans le dossier « **/vue/css** » de l’application.

IMPORTANT

Convention de nommage pour les balises HTML identifiées : id="…"

Balises	Préfixe à utiliser
input type="text", textarea	id="edt_"
input type="checkbox"	id="chk_"
input type="radio"	id="radio_"
input type="button"	id="btn_"
select	id="select_"
div, span	id="div_"

Si votre application Web contient plusieurs pages HTML (cas d’une application de gestion d’une base de données) :

il faut ajouter en début de préfixe un identifiant supplémentaire, comme le nom de la table principale utilisée par la page HTML
(*par exemple* : « edt_salle_lib » dans la page HTML « salle_edit.html » qui gère l’édition d’une salle)

IHM – Application HTML/CSS/TS : préparation et exécution

étape 3 : code TypeScript

Fichiers « typescript » à implémenter

Il faut déterminer les événements à implémenter (*click* sur un bouton, *keypress* sur une zone de saisie, …) à partir du modèle des tâches.

A chaque événement, une fonction TS sera associée.

Par exemple : à l'événement « click » sur le bouton « valider » sera associé la fonction « valideSaisie » à implémenter dans le ficher « ts »

Pour une maquette écran, il y aura toujours deux fichiers « ts » contenant :

- le premier la définition de la classe et son instanciation (objet de la classe),
- le second l'initialisation de l'objet et la définition des événements

Les fichiers « **ts** » à créer seront dans le dossier « **/src/controleur** » de l'application.

IMPORTANT

conventions en typescript

- *le Camel case* pour les **variables**, **attributs, fonctions**, **instances** de classes, **méthodes** :
  les identificateurs sont liés, sans espace ;
  chaque identificateur commence par une majuscule, sauf le premier (*exemple* : firstName)
- *le Pascal case* pour les **types**, les **classes** : les identificateurs sont liés, sans espace ;
  chaque identificateur commence par une majuscule (*exemple* : FirstName)
- un attribut d'une classe est toujours
  déclaré « private » et préfixé par le symbole « _ »
- l'accès à un attribut « private » par l'objet (instance de classe) se fait en lecture par un getter (méthode « get ») et en écriture par un setter (méthode set)

convention de nommage pour les attributs objets « ts » associés aux balises HTML

Balises	Préfixe à utiliser pour l'attribut	*exemple*
input type="text", textarea	edt	*edtNom*
input type="checkbox"	chk	*chkLinux*
input type="radio"	radio	*radioMadame*
input type="button"	btn	*btnValider*
select	liste	*listeDiplome*
div, span	div	*divFormBoutons*

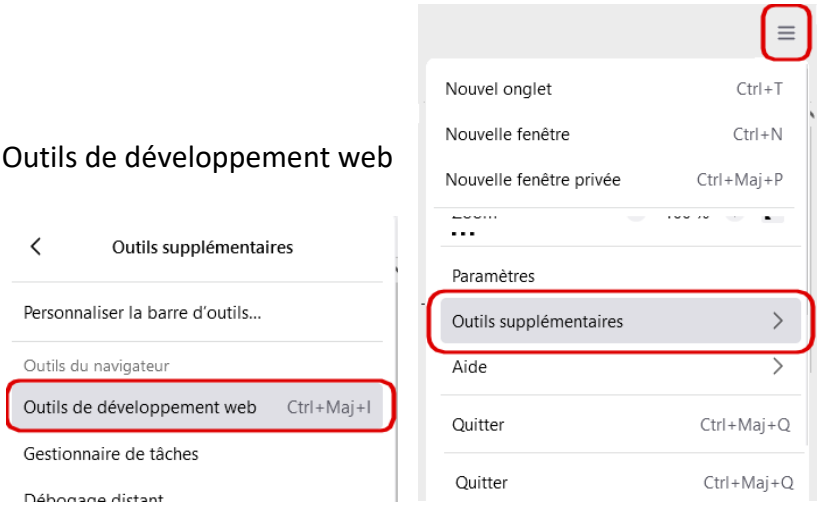
IHM – Application HTML/CSS/TS : préparation et exécution

étape 4 : lancement/test de l'application

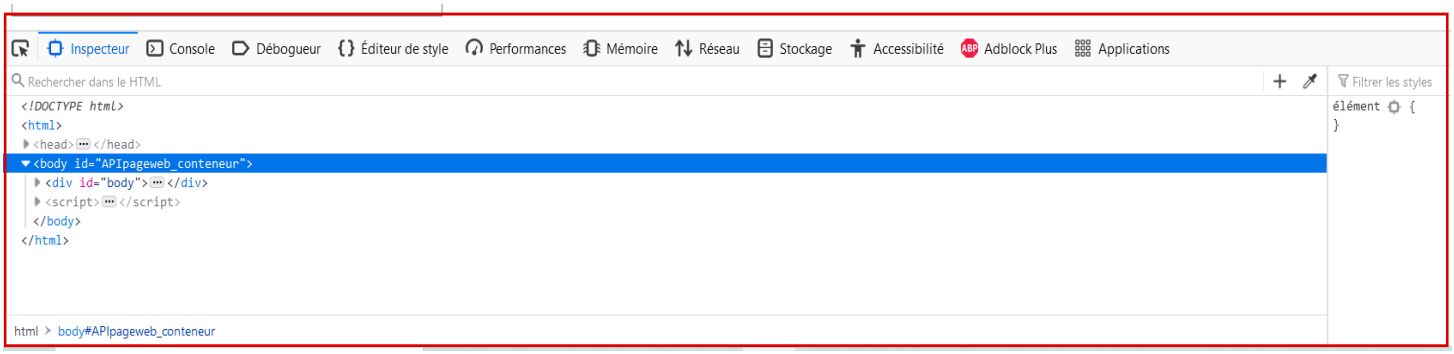
1. Configuration du navigateur

1.1. Firefox

- cliquer sur le menu « burger »
- puis Outils supplémentaires→ Outils de développement web



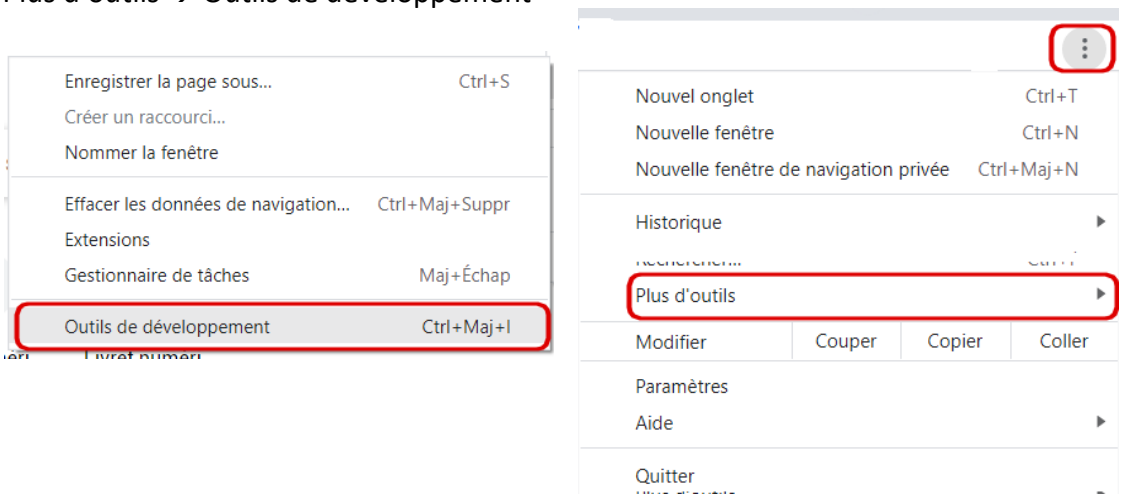
- Un panneau avec plusieurs onglets apparaît



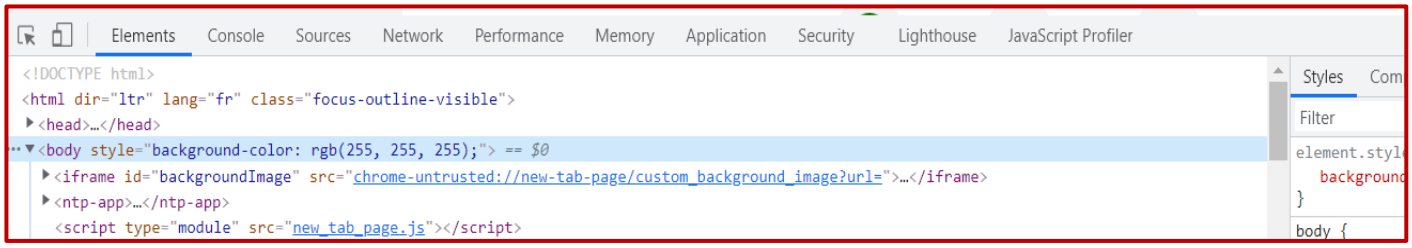
- Cliquer sur l'onglet « Réseau » et vérifier que la case est cochée sinon la cocher.

1.2. Chrome

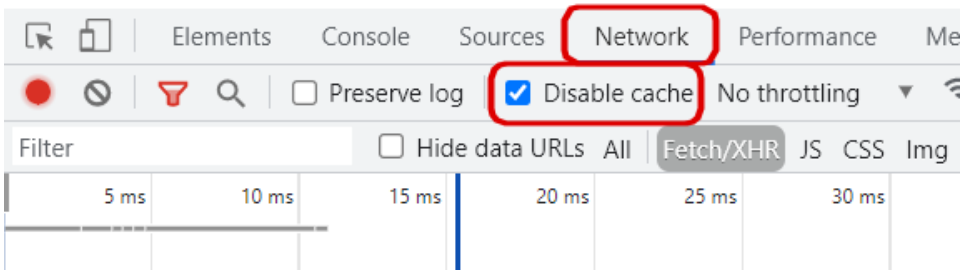
- cliquer sur le menu « trois points verticaux »
- puis Plus d'outils → Outils de développement



- Un panneau avec plusieurs onglets apparaît



- Cliquer sur l'onglet « Network » et vérifier que la case est cochée sinon la cocher.



2. Lancer l'application

Pour tester, lancer votre application web HTML-TypeScript/javascript sur « devweb ».

Par exemple, l'étudiant « toto3u » veut tester son application web « tp1 » qui est située dans le dossier « ihm/tp1 », il devrai taper la commande suivante dans la barre de navigation du navigateur :

https://devweb.iutmetz.univ-lorraine.fr/~toto3u/ihm/tp1/vue/tp1.html