Formation DWWM

Dossier de projet professionnel

DEREDEC Xavier

20/07/2021

# Table des matières

[Table des matières 1](#_Toc78468706)

[1- Présentation 2](#_Toc78468707)

[A- Liste des compétences couvertes par le projet 2](#_Toc78468708)

[B- Résumé du projet 3](#_Toc78468709)

[2- Cahier des charges 3](#_Toc78468710)

[A- Expression des besoins 3](#_Toc78468711)

[B- Charte graphique 4](#_Toc78468712)

[1- Logo 4](#_Toc78468713)

[2- Polices d’écriture 4](#_Toc78468714)

[3- Palette de couleur 4](#_Toc78468715)

[3- Spécifications techniques 5](#_Toc78468716)

[A- Base de donnée 5](#_Toc78468717)

[B- Technologies utilisées 5](#_Toc78468718)

[1- Maquettage 5](#_Toc78468719)

[2- Gestion de version 5](#_Toc78468720)

[3- Développement Front-end 6](#_Toc78468721)

[4- Développement Back-end 6](#_Toc78468722)

[C- User Stories 7](#_Toc78468723)

[D- Maquettage et Wireframe 7](#_Toc78468724)

[1- Mobile 7](#_Toc78468725)

[2- Desktop 8](#_Toc78468726)

[4- Réalisation du Projet 8](#_Toc78468727)

[A- Site vitrine 8](#_Toc78468728)

[1- Génération de PDF 8](#_Toc78468729)

[2- Envoi du formulaire de contact 11](#_Toc78468730)

[3- Responsive design 13](#_Toc78468731)

[B- Back Office 13](#_Toc78468732)

[1- Ajouter un forfait 13](#_Toc78468733)

[2- Connexion au back office 15](#_Toc78468734)

# Présentation

## Liste des compétences couvertes par le projet

Front-End :

Maquetter une application

Réaliser une interface utilisateur web statique adaptables

Développer une interface utilisateur dynamique

Back-End :

Créer une base de données

Développer les composants d’accès aux données

Développer la partie back-end d’une application web ou web mobile

Elaborer et mettre en œuvre des composant dans une application de gestion de contenues

## Résumé du projet

Mon projet a été réalisé pour l’entreprise « SASU Auto-Ecole de Pierrefeu », dans l’objectif d’acquérir le label qualité par l’état et de se positionner en concurrence des auto-école en ligne. L’une des conditions requise pour obtenir ce label est de posséder un site internet contenant les tarifs, la liste des enseignant, les méthodes utilisées ainsi que le déroulement de chaque examen. Les visiteurs doivent aussi pouvoir laisser un avis sur le site internet ou encore contacter l’entreprise afin d’obtenir des renseignements complémentaires sur les taux de réussite et les délais moyen d’obtention du permis. De plus, il m’a également été demandé de développer une interface permettant de gérer les éléments amener à changer régulièrement comme les tarifs, la liste des forfaits disponibles ou encore les informations relatives à ses moniteurs.

Mon projet est donc constitué d’une partie site vitrine et d’une partie back-office permettant la gestion du contenue dynamique.

Pour la partie site vitrine, j’ai décidé d’utiliser le framework Angular couplé au préprocesseur SASS concernant le front-end, et PHP couplé à MySQL pour la partie back-end.

Pour la partie back-office j’ai également utilisé le framework Angular couplé au préprocesseur SASS concernant le front-end, et PHP couplé à MySQL pour la partie back-end, tout en incluant l’API Electron.JS pour le déporté sous forme d’application de bureau.

# 2- Cahier des charges

## Expression des besoins

Pour que la « SASU Auto-Ecole de Pierrefeu » soit labellisée par l’État nous avons besoin d’une application web présentant l’établissement et contenant toutes les informations requises dans la charte de labellisation fournie en annexe. La charte graphique devra s’orienter autour du logo de son entreprise afin que l’identité graphique de l’entreprise se reflète sur son site web. Pour la disposition des éléments ainsi que les technologies employées, nous avons carte blanche. Il nous faudra également une interface permettant de pouvoir modifier les éléments dynamiques gérer par la base de données plus simplement.

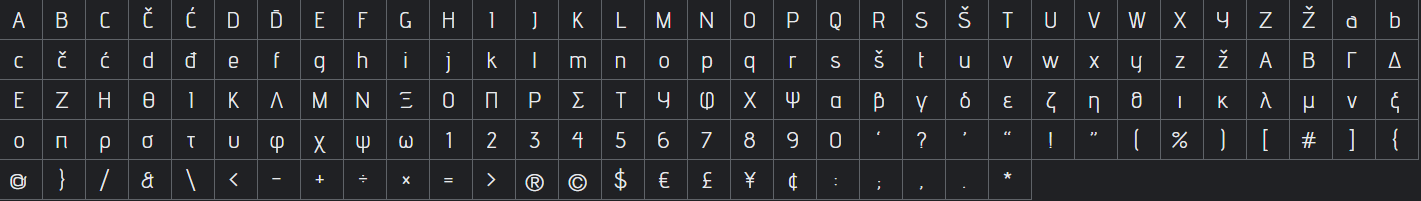
## Charte graphique

### Logo

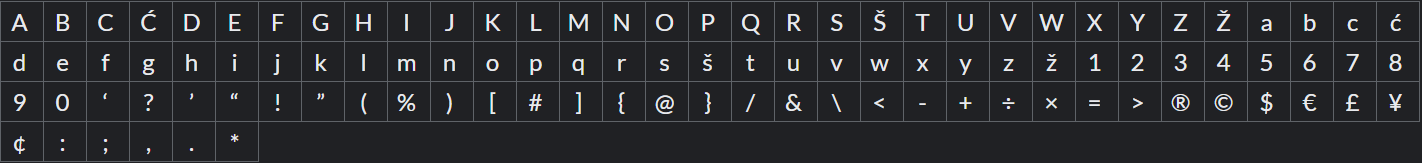


### Polices d’écriture

Pour les titres la police choisi est **Advent Pro**



Pour les corps de texte la police choisi est **Lato**



### Palette de couleur

**#FFFFFF #D699C2 #990066 #000000**

# Spécifications techniques

## Base de donnée

Pour la partie base de donnée j’ai commencé par effectuer une liste des éléments pouvant être amener à changer régulièrement comme les tarifs, les forfaits ainsi que leurs contenues ou encore la liste des enseignant et de leurs informations.

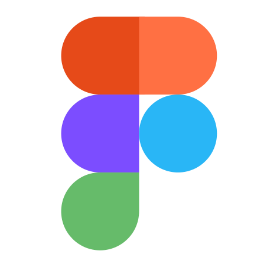
Suite à cela j’ai réfléchis à qu’elle type de donnée ils pourraient être associer afin de rédiger le dictionnaire de donnée (voir annexe 1).

De ce dictionnaire en à découlée le modèle conceptuel de donnée avec toutes les table, les colonnes et les contraintes de clef primaire, d’index et de clef étrangère (voir annexe 1).

## Technologies utilisées

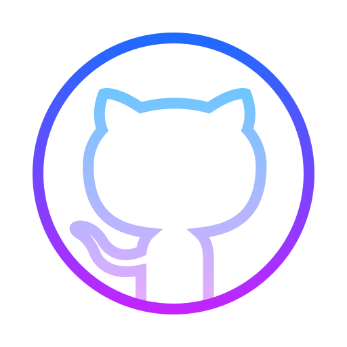
### Maquettage

J’ai utilisé le logiciel Figma pour toute la partie maquettage de ce projet



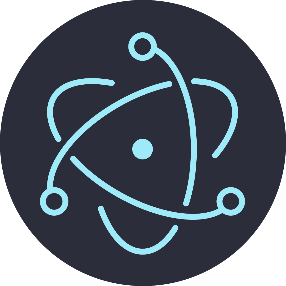
### Gestion de version

Afin d’effectué un suivi des versions de mon j’ai décidé d’utiliser git au travers de l’outil GitHub Desktop



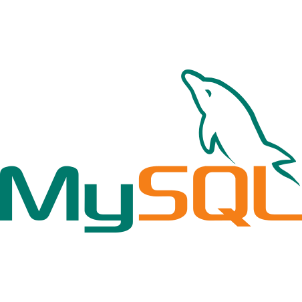
### Développement Front-end

Pour la partie front-end de ce projet j’ai décidé d’utilisé Angular couplé avec le préprocesseur SASS ainsi la librairie HammerJS. L’API ElectronJS a aussi été utilisé pour la partie backoffice afin de le déporter sous forme d’application de bureau et d’éviter d’éventuels essaie d’intrusion sur le système de gestion du contenue.

### Développement Back-end

Pour la partie back-end j’ai utilisé le langage PHP couplé à MySQL pour la gestion de la base de donnée



## User Stories

Pour la partie user story de ce projet j’ai commencé par lister toutes les contraintes de fonctionnalité lister dans la charte de labélisation des auto-école.

J’ai ensuite ajouter les besoins supplémentaires de l’entreprise et trié chaque fonctionnalité selon leur importance, leurs difficultés et la business value.

Pour finir je les aie consignés dans un tableau trié en 2 volet, Site-Vitrine et Back-Office, afin de pouvoir consulter à chaque étape si toutes les fonctionnalités nécessaires ont bien été ajouter (voir annexe 2).

## Maquettage et Wireframe

Pour le site vitrine je n’ai effectué que le wireframe car la partie UI Design a été confiée à une graphiste indépendante nommée **Charlotte Divo.** Pour la partie backoffice je n’ai pas eu à tenir compte des format mobile et tablette étant donnée qu’il va se présenter sous la forme d’une application de bureau, qui ne pourra être exécuter que sur ordinateur (voir annexe 3).

### Mobile

Accueil

Pour la page d’accueil j’ai décidé de faire apparaitre les éléments les plus significatif et vendeur de l’entreprise et de favoriser les carrousels pour l’affichage du contenue dynamique afin d’obtenir une meilleure ergonomie (voir annexe 3).

Permis

Pour la page permis j’ai décidé d’essayer limiter la zone d’affichage à la taille d’une vue d’écran afin que les utilisateurs aient le contenue principale dans leur champ de vue initial (voir annexe 3).

Tarifs

Pour la page des tarifs je suis partie du même postula que pour la page permis tout en me laissant une petite marge pour le téléchargement des tarifs détaillés (voir annexe 3).

Contact

Pour la page de contact j’ai aussi voulu limiter la taille en fonction d’une zone de vue étant donnée le peu de contenue qu’elle contient (voir annexe 3).

### Desktop

Pour le wireframe desktop la seule différence notable pour le site vitrine est la disposition des éléments sur la pages d’accueil (voir annexe 3).

Pour le back office j’ai décidé de partir sur un système d’interface simple et épuré baser sur des onglets au noms compréhensible facilement afin que l’utilisateur puissent gérer son contenue de manière fluide et intuitive.

# 4- Réalisation du Projet

## Site vitrine

### Génération de PDF

Dans le component **prices** de mon site vitrine j’ai inclus un lien permettant de consulter la grille tarifaire détaillée de l’entreprise en version PDF.

Pour ce faire j’ai utilisé les fonctions suivantes :

#### getPdf()

  getPdf(){

    this.getService.getAllDetail().subscribe((datas: any[]) => {

      if(datas){

        for(let data of datas) {

          this.detailList.push(

[data.Nom\_tarif, data.Nom\_type, String(data.Prix\_tarif)+"€ TTC"])

        }

        this.pdfService.createPdf(this.detailList);

      }

    });

Quand l’utilisateur clique sur le lien, la fonction **getPdf()** s’exécute de la manière suivante :

- Elle souscrit à l’observable retourner par la fonction **getAllDetail()** appartenant au service **get** qui retourne la liste des tarifs détaillés

- Elle stocke chaque ligne du tableau de tarif dans le tableau **detailList** sous la forme :

[**nom\_Tarifs, nom\_Types, Prix\_tarif + € TTC**]

- Elle appelle la fonction **createPdf()** du service **pdfmake** et lui passe en paramètre le tableau **detailList**

#### getAllDetail()

getAllDetail(): Observable<ITarif[]> {

    return this.http.get<ITarif[]>(`http://autoeci.cluster030.hosting.ovh.net/script/getDetail.php`);

  }

La fonction **getAllDetail()** s’exécute de la manière suivante :

- Elle retourne sous la forme d’un observable le résultat de la requête contenue dans le fichier **getDetail.php**

#### createPdf()

  createPdf(data: any[]) {

    var docDefintition: TDocumentDefinitions = {

      header: {

        columns:[

          {

            text:'Nos Tarifs Detaillées',

            style: 'title',

            alignment: 'center'

          }

        ]

      },

      content: [

        {

          margin: [65,20, 0, 0 ],

          table: {

            headerRows: 1,

            body: data,

          }

        }

      ],

      styles: {

        title: {

          fontSize: 22,

          bold: true,

          color: '#900066'

        },

        header: {

          fontSize: 15,

          bold: true,

          color: '#000000'

        }

      }

    };

    pdfMake.createPdf(docDefintition).open();

  }

La fonction **createPdf()** s’exécute de la manière suivante :

-Elle met en forme le contenue du pdf via la variable **docDefinition** et y inclus le contenue du paramètre **data**

-Elle fait appelle à la fonction **createPdf()**  de la librairie **PdfMake** en lui donnant comme paramètre la variable **docDefinition**

-Elle exécute de manière asynchrone la fonction **open()**  de la librairie **PdfMake** afin d’ouvrir le pdf générer juste au-dessus

### 2- Envoi du formulaire de contact

Pour transmettre les informations du formulaire de contact par mail j’ai mis en place plusieurs contrôle coté front-end et back end qui sont :

#### Interface IContact

interface IContact {

  nom: string,

  prenom: string,

  sujet: string,

  mail: string,

  message: string

}

Afin de faire la passerelle entre mon **reactive formgroup** et mon **PHP,** j’ai décidé d’implémenter une interface pour éviter les problèmes liés au typage.

#### initform()

initForm(){

    this.contactForm = this.formBuilder.group({

      nom: ['', [Validators.required, Validators.pattern(/[0-9a-zA-Z]/), Validators.maxLength(100)]],

      prenom: ['', [Validators.required, Validators.pattern(/[0-9a-zA-Z]/), Validators.maxLength(100)]],

      sujet: ['', [Validators.required, Validators.pattern(/[0-9a-zA-Z]/), Validators.maxLength(100)]],

      mail: ['', [Validators.required, Validators.email, Validators.maxLength(100)]],

      message: ['', [Validators.required, Validators.pattern(/[0-9a-zA-Z]/), Validators.maxLength(255)]]

    });

  }

Afin de contrôler au mieux les saisies utilisateur coté front-end, j’ai utilisé la technologie **reactive formgroup d’angular** pour effectuer une vérification de premier niveau sur mon formulaire et lever des exceptions ainsi qu’afficher des retours d’erreur à l’utilisateur de manière dynamique.

#### onSubmit()

  onSubmit(){

    if(this.contactForm.valid){

      let infoContact: IContact = {

        nom: this.contactForm.get(['nom'])?.value,

        prenom: this.contactForm.get(['prenom'])?.value,

        sujet: this.contactForm.get(['sujet'])?.value,

        mail: this.contactForm.get(['mail'])?.value,

        message: this.contactForm.get(['message'])?.value

      }

      this.httpClient.post('http://autoeci.cluster030.hosting.ovh.net/script/contact.php', JSON.stringify(infoContact)).subscribe(

        (response: any) => {

          response['success'] ? alert("message envoyé !") : alert("oups une erreur est survenue, veuillez reesayer !");

        },

        (error) => console.log(error)

      );

    }

    else{

      alert("veuillez remplir tout les champs correctement !")

    }

  }

Afin de contrôler la validité des saisies utilisateur au moment de l’envoi, j’ai implémenté une deuxième vérification qui permet d’envoyer le formulaire uniquement si tous les **validators** sont respectés.

#### regex\_data()

function regex\_data($param)

{

    $param = trim($param);

    $param = stripcslashes($param);

    $param = htmlspecialchars($param);

    return $param;

}

Afin de contrôler les saisies utilisateurs coté back-end, j’utilise la fonction **regex\_data()** qui permet d’échapper les espaces, supprimer les antislash et de convertir les caractères spéciaux en entités HTML pour éviter les injections de code via mon formulaire de contact.

### Responsive design

Pour la partie responsive du site vitrine j’ai intégré différents breakpoint/media queries au sein de mon **SASS** afin d’adapter la disposition, la taille et la forme de mes différents éléments au format de l’appareil de l’utilisateur.

## Back Office

### Ajouter un forfait

Dans mon component **add-pack,** j’ai créé un input de type **submit** qui à l’activation exécute les fonctions suivantes :

#### publish()

publish(){

    const newPack: IPack = {

      nom: this.addPackForm.get('nom')?.value,

      prix: this.addPackForm.get('price')?.value,

      description: this.addPackForm.get('description')?.value,

      type: this.typeId

    }

    this.addService.addPack(newPack);

  }

La fonction **publish()**  permet d’ajouter un nouveau forfait en effectuant plusieurs actions qui sont :

- Création d’un objet **newPack** de type **IPack** contenant toutes les informations concernant le forfait à ajouter

-Appel de la fonction **addPack()**  du service **add** qui prend comme paramètre l’objet créé ci-dessus

#### addPack()

  addPack(newPack: IPack) {

    let packData = JSON.stringify(newPack);

    this.http.post('http://autoeci.cluster030.hosting.ovh.net/script/addPack.php', packData).subscribe(

      (response: any) => {

        response['success'] ? this.router.navigate(['/dashboard']) : console.log('erreur');

      },

      (error) => console.log(error)

    );

La fonction **addPack()**  permet de transmettre les informations fournies en paramètre à un fichier **PHP** afin de créer un nouveau forfait dans la base de donnée en suivant ces étapes :

-Création d’une variable **packData** contenant l’objet passé en paramètre converti au format **JSON**

-Appel de la fonction **post()** du module **HTTPClient**  d’**Angular**, qui permet de transmettre le contenue de la variable crée ci-dessus au fichier **addPack.php**

-Traitement de la réponse retourné par la fonction citée ci-dessus afin de lever des exceptions en cas d’erreur ou de rediriger sur le component **dashboard** en cas de succès

### Connexion au back office

Afin de pouvoir sécuriser l’accès aux fonctionnalités du back office mon application fait appelle au fonction et service suivant :

#### Service AuthGuard

export class AuthGuardService {

  constructor(private authService: AuthService, private router: Router) { }

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean {

    if (this.authService.userGuard != null) {

      return true;

    } else {

      return this.router.navigate(['/auth']);

    }

  }

}

Ce service permet de bloquer l’accès aux fonctionnalité à tout utilisateur non-authentifié en effectuant les actions suivantes :

-Si la valeur de la variable **userGuard** du service **auth** est différente de null elle autorise l’accès

-Sinon elle redirige l’utilisateur sur le component **auth**

#### Service Auth

export class AuthService {

  userGuard: any = sessionStorage.getItem('id');

  userGuardSubject = new Subject<any>();

  constructor(private http: HttpClient, private router: Router) { }

  emituserGuard(){

    this.userGuardSubject.next(this.userGuard);

  }

  signIn(user: IUser):boolean{

    let check: boolean = false;

    let userData = JSON.stringify(user);

    this.http.post('http://autoeci.cluster030.hosting.ovh.net/script/auth.php', userData).subscribe(

      (response: any) => {

        if(response['success']) {

          sessionStorage.setItem('id', response['id']);

          this.userGuard = sessionStorage.getItem('id');

          this.emituserGuard();

          this.router.navigate(['/dashboard']);

          check = true;

        }

        else{

          check = false;

        }

      },

      (error) => console.log(error)

    )

    return check;

  }

  singOut(){

    sessionStorage.removeItem('id');

    this.userGuard = null;

    this.emituserGuard();

    this.router.navigate(['/auth'])

  }

}

Dans mon service **auth,** il y a deux fonctions capitales :

##### signIn()

-Elle stocke dans une variable **userData** le contenu du paramètre qui lui est passé converti au format **JSON**

-Elle utilise la fonction **post()** du module **HTTPClient** d’**Angular** afin de transmettre la variable précédemment créer au fichier **auth.php**

-Elle traite la réponse envoyée par le fichier **auth.php** :

-En cas de succès, elle crée un token **JWT** qui est stocké en session storage, met à jour la variable **userGuard** et son **subject** puis redirige l’utilisateur vers le component **dashboard**

-En cas d’échec sur les identifiant elle retourne la valeur false

-En cas d’erreur php elle retourne l’erreur php

##### SignOut()

-Elle supprime le token **JWT** du session storage

-Elle remet la variable **userGuard** à null et met à jour son **subject**

-Elle redirige l’utilisateur sur le component **auth**

#### Component auth

Au sein de mon component **auth** les contrôles ont été effectuer grâce aux fonctionnalités suivante :

-Un **reactive formGroup** qui me permet de contrôler les saisie utilisateur de manière immédiate afin de faire un retour graphique directement à l’utilisateur

-Une interface **IUser** qui permet d’éviter les problèmes de typage

#### auth.php

Du coté **PHP,** plusieurs éléments ont été mis en place afin de sécuriser et contrôler les données d’authentification :

-La fonction **regex\_data()** expliquée plus haut pour éviter les injections de code via le formulaire de connexion

-La fonction **password\_verify** de **PHP** afin de vérifier la correspondance entre le mot de passe saisie et le mot de passe hash dans la base de donnée