|  |  |  |
| --- | --- | --- |
|  |  |  |
| *Nom de naissance* |  | CAUCHE |
| *Nom d’usage* |  | CAUCHE |
| *Prénom* |  | Anthony |
| *Adresse* |  | 66 grande rue 01500 Ambronay |
|  |  |

|  |  |
| --- | --- |
| **Titre professionnel visé** | |
|  | |
| GRADUATE DEVELOPPEUR ANGULAR | |
|  | |
| **Modalité d’accès :** | |
|  | |
|  | Parcours de formation |
|  | Validation des Acquis de l’Expérience (VAE) |
|  | |

|  |  |
| --- | --- |
| **Présentation du dossier** | |
|  | |
|  | |
| Le dossier professionnel (DP) constitue un élément du système de validation du titre professionnel. **Ce titre est délivré par le Ministère chargé de l’emploi.**  Le DP appartient au candidat. Il le conserve, l’actualise durant son parcours et le présente **obligatoirement à chaque session d’examen**.  Pour rédiger le DP, le candidat peut être aidé par un formateur ou par un accompagnateur VAE.  Il est consulté par le jury au moment de la session d’examen.  **Pour prendre sa décision, le jury dispose :**   1. des résultats de la mise en situation professionnelle complétés, éventuellement, du questionnaire professionnel ou de l’entretien professionnel ou de l’entretien technique ou du questionnement à partir de productions. 2. du **Dossier Professionnel** (DP) dans lequel le candidat a consigné les preuves de sa pratique professionnelle. 3. des résultats des évaluations passées en cours de formation lorsque le candidat évalué est issu d’un parcours de formation 4. de l’entretien final (dans le cadre de la session titre).   *[Arrêté du 22 décembre 2015, relatif aux conditions de délivrance des titres professionnels*  *du ministère chargé de l’Emploi]*  **Ce dossier comporte :**   * pour chaque activité-type du titre visé, un à trois exemples de pratique professionnelle ; * un tableau à renseigner si le candidat souhaite porter à la connaissance du jury la détention d’un titre, d’un diplôme, d’un certificat de qualification professionnelle (CQP) ou des attestations de formation ; * une déclaration sur l’honneur à compléter et à signer ; * des documents illustrant la pratique professionnelle du candidat (facultatif) * des annexes, si nécessaire. | |
| *Pour compléter ce dossier, le candidat dispose d’un site web en accès libre sur le site.* | |
|  | [**http://travail-emploi.gouv.fr/titres-professionnels**](http://travail-emploi.gouv.fr/titres-professionnels) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sommaire** | | | |
| **Exemples de pratique professionnelle** | | |
| **Intitulé de l’activité-type n° 1** | | **p.** | **5** |
|  |  Intitulé de l’exemple n° 1  Site Vitrine photographe Charles Cantin p. | p. |  |
|  |  Intitulé de l’exemple n° 2  p. | p. |  |
|  |  Intitulé de l’exemple n° 3  p | p. |  |
|  |  |  |  |
| **Intitulé de l’activité-type n° 2** | | **p.** | **11** |
|  |  Intitulé de l’exemple n° 1  Site e-commerce p. | p. |  |
|  |  Intitulé de l’exemple n° 2  p. | p. |  |
|  |  Intitulé de l’exemple n° 3  p | p. |  |
|  |  |  |  |
| **Intitulé de l’activité-type n° 3** | | **p.** | **28** |
|  |  Intitulé de l’exemple n° 1  Blog site p. | p. |  |
|  |  Intitulé de l’exemple n° 2  p. | p. |  |
|  |  Intitulé de l’exemple n° 3  p | p. |  |
|  |  |  |  |
| **Intitulé de l’activité-type n° 4** | | **p.** |  |
|  |  Intitulé de l’exemple n° 1  p. | p. |  |
|  |  Intitulé de l’exemple n° 2  p. | p. |  |
|  |  Intitulé de l’exemple n° 3  p | p. |  |
|  |  |  |  |
| **Titres, diplômes, CQP, attestations de formation** *(facultatif)* | | **p.** |  |
| **Déclaration sur l’honneur** | | **p.** |  |
| **Documents illustrant la pratique professionnelle** *(facultatif)* | | **p.** |  |
| **Annexes** *(Si le RC le prévoit)* | | **p.** |  |

**Exemples de pratique**

**professionnelle**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Activité-type** | **1** | **Cliquez ici pour entrer l’intitulé de l’activité** | | | |
| ***Exemple n°1***  | | *Cliquez ici pour entrer l’intitulé de l’exemple* | | | |
|  | | | | | |
|  | | | | | |
| **1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :** | | | | | |
|  | | | | | |
| - Pour ce site il m’avait été demandé de faire un portfolio sous la forme d’un site web, de faire 4 pages, de faire une page complète en page d’accueil, d’y incorporer un système de tri / filtre de photos dynamique.  -J’ai d’abord commencé par définir la charte graphique en me basant sur les besoins clients.  Ensuite, j’ai fait les mockups à la main et wireframes sur figma en version desktop et mobile.        -Puis j’ai intégré les librairies nécessaires pour ce travail à savoir :   * AOS * Bootstrap   Ainsi que des photos libres de droit avec le logo qui m’a été confié pour ce devoir.  -J’ai créé les composants et fais chaque partie du frontend dans un premier temps en intégrant des fonctionnalités simples, basiques et natives de Bootstrap comme la navbar, les carrousels, etc...  -Le plus difficile fût de créer le système de tri de photos. J’ai mis côté front un input avec les boutons qui serviront à trier les photos pour le rendu final.  -Ensuite j’ai créé une div cards qui allait contenir tout ce que j’allais intégrer en back.  -Côté backend j’ai créé un objet cards ou j’ai défini chaque élément par une description, une catégorie, un titre ainsi qu’une image dans un fichier script dédié.    -Ensuite, en parcourant le tableau, l’objet cards, j’ai créé des composants, des div, auxquelles j’ai ajouté des classes que j’avais aussi définis dans mon fichier CSS au préalable en m’aidant des méthodes natives de javascript à savoir classList.add ou createElement() par exemple. Ainsi, mes composants étaient créés et retournés avec du style.    -Pour les systèmes de recherche et de tri j’ai joué avec le front en utilisant les méthodes add ou remove en back et la classe hide qui cachait un élément en CSS ou bien active qui mettait en avant la catégorie de filtre sélectionnée.  -J’ai créé une fonction filterProduct que j’ai pu paramétrer à « all » onload de la page pour que tous les composants s’affichent à l’ouverture de la page.  - Ensuite en ce qui concerne le formulaire de contact à la dernière page j’ai créé des fonctions possédant du REGEX pour que chaque champ soit bien rempli correctement par l’utilisateur. Je me suis basé sur « never trust user input » pour vraiment formater le rendu.      -Il ne me restait plus qu’à dynamiser le tout. En utilisant gsap, j’ai pu faire un ordre d’apparition de composant sur la page d’accueil.    -Puis grâce à la librairie d’évènements onscroll AOS je paramètre des effets d’apparition de div.    -Ensuite j’ai mis en ligne mon travail grâce à netlify qui héberge et mets en ligne gratuitement les sites en se connectant à nos projets sur git. | | | | | |
|  | | | | | |
|  | | | | |
|  | | | | | |
| **2. Précisez les moyens utilisés :** | | | | | |
|  | | | | | |
| Je me suis servi des cours de STUDI, de VS code, du live server, de tutos etc.  Je me suis servi du site UNSPLASH qui permet de télécharger des images libres de droit.  J’ai utilisé les librairies Bootstrap et AOS que j’avais téléchargé au préalable.  Je me suis aidé d’un tuto YouTube qui expliquait comment se servir de gsap pour dynamiser une page.  Je me suis servi à plusieurs reprise de la documentation developer.mozilla qui explique explicitement le rôle de chaque méthode.  J’ai utilisé formspee.io pour rediriger le formulaire vers mon adresse mail.  Je me suis servi bien évidemment des besoins clients et des supports mis à disposition. | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
| **3. Avec qui avez-vous travaillé ?** | | | | | |
|  | | | | | |
| J’ai travaillé seul | | | | | |
|  | | | | | |
|  | | | | |
|  | | | | | |
| **4. Contexte** | | | | | |
|  | | | |  | |
| Nom de l’entreprise, organisme ou association | | | | *STUDI* | |
| Chantier, atelier, service | | | Evaluation d’entrainement | | |
| Période d’exerciceDu : *Cliquez ici*au :*Cliquez ici* | | | | | |
|  | | | | |
|  | | | | |
| **5. Informations complémentaires** *(facultatif)* | | | | |
|  | | | | | |
| J’ai fait le responsif principalement à la fin. Ce site n’est pas mobile first.  Le site est disponible sur  <https://charlesphotographies.netlify.app/>  Le git est le suivant :  https://github.com/Toty-01/Charles-Cantin | | | | | |
|  | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Activité-type** | **2** | **Cliquez ici pour entrer l’intitulé de l’activité** | | | |
| ***Exemple n° 1***  | | *Cliquez ici pour entrer l’intitulé de l’exemple* | | | |
|  | | | | | |
|  | | | | | |
| **1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :** | | | | | |
|  | | | | | |
| -Pour ce site j’ai voulu réaliser un site e-commerce en angular. Le but était de me servir de réaliser un site web dynamique, jouant avec les éléments et leurs données.  -J’ai commencé par faire le mockup à la main, pour me donner une idée du style générale du site.  -J’ai décidé pour ce projet de me contenter d’un style minimaliste en mettant pas beaucoup de couleurs.  -J’ai ensuite décidé de lancer l’installation de l’application grâce à la commande ng new store. Je décide d’y implémenter app-routing module.  -Ensuite, pour simplifier le côté front et donner du style plus facilement à chaque composant que je vais créer, j’installe les librairies et composants qui me seront nécessaires pour ce projet à savoir :    -J’allais donc me servir de angular Material.  -J’installe aussi tailwind pour pouvoir gérer les composants directement depuis le html et je l’initialise    -Pour commencer, je vais créer mon premier composant header-component qui sera toujours en haut de la page à l’aide de la commande « ng g c header ».  -Je vais créer chaque composant ou service avec l’invite de commande.  -Cette navbar qui sera toujours présente je décide de la placer tout en haut de l’arborescence avec le router-outlet pour pouvoir naviguer dans l’app. Ici, j’appelle composant grâce à son selector app-header.    -Après avoir créé cette nabar, je suis allé m’occuper de l’aspect que j’allais lui donner. Ce composant comme la plupart de ceux qui seront dans ce projet possèderont une partie html en front et un fichier css pour y injecter un style plus précis si besoin.  -Je vais définir chaque composant dynamique par les noms que je donnerais aux fonctions associées que je définirais plus tard. Il ne restera plus qu’à mettre les doubles accolades.    -Cette navbar va contenir un lien « logo » qui ramènera l’utilisateur vers la page home, un panier qui switchera si l’utilisateur clic dessus et qui redirigera vers le panier ou videra le panier.  J’ai mis aussi une bulle qui indiquera la quantité de produits dans le panier.    -Puis je me suis occupé de l’arborescence en créant des fichiers components, models, pages et bien sûr services.  -Apres cela j’ai créé mes sous dossiers pages dans lesquelles j’ai mis les futures pages et j’ai ainsi fait tous mes composants à la volée.    -Ensuite, j’ai configuré mes routes dans l’app-routing-module en précisant que toute les urls ou une url vide redirigera la page vers le component HomeComponent    -Ensuite je suis allé m’occuper de la page home ou j’ai créé la mise en page à l’aide des mat drawer qu angular material met à notre disposition.  -Je créer la sidebar qui possèdera un menu déroulant qui affichera les différentes catégories que j’importerais d’un futur composant.  -Ensuite je suis allé créer le component products-header que je range dans les components de la page home. Ce composant possedera :   * Un menu déroulant sur sa gauche permettant de trier les prix par ordre croissant ou décroissant. * Un autre menu déroulant qui permettra d’afficher un nombre d’éléments choisit. Ici 6 ou 12 ou 20. * Des icônes qui afficheront le nombre d’élément par colonne. Ici 1 ou 2 ou 3 par défaut.     -Ensuite je créer un composant filters. C’est ce composant que je vais injecter à la page sidebar pour trier les éléments par catégories.    -Ensuite j’ai créé des catégories manuellement et je les ai injecté en les bouclant avec \*ngfor pour me rendre compte que ça marchait bien après avoir exporté le composant dans la sidebar .  -Après ça je suis allé créer une méthode onShowCategory qui prendra en paramètre une catégorie de type string et qui retournera un new EventEmitter de type string. Cette méthode va donc prendre la catégorie sélectionnée et la retourner.  Ensuite j’exporte les données jusqu’au composant parent -> home avec un output.    -Puis je créer une méthode qui renverra la catégorie sélectionnée.    onShowCategory(newCategory: string): void {      this.category = newCategory;    }  -Ensuite j’ai paramétré les rows-height de la mat grid list dans une constante    -J’ai créé après cela le composant product-box. Je créer une mat-card d’angular material que j’alimente d’une image, d’un titre, d’une description, d’un prix et d’une icône qui ajoutera les produits dans le panier.  -Je décide de créer un fullWidthMode à false comme je voudrais que l’affichage de départ soit composé de 3 articles    @Input() fullWidthMode = false;  -J importe le booleen que je créer à l’élément parent home que je définis à ‘col === 1’.  -Je me servirais de ce composant pour définir le front de mes cards en fonction de lui.  -Exemple :      [ngClass]="{'w-[200px], truncate': !fullWidthMode, 'w-[auto]': fullWidthMode}"  Ou encore      [ngClass]="{'px-8 flex flex-col justify-between': fullWidthMode}"  -Ensuite j’ai créé un modèle de produit qu‘il y aura dans le panier.  export interface Cart {    items: Array<CartItem>  }  export interface CartItem {    product: string;    name: string;    price: number;    quantity: number;    id: number;  }  -Ensuite j’ai créé la page cart et dans le typescript j’ai fait des faux produits qui me serviront de modèle.  -Côté front, j’ai commencé par un affichage conditionnel dans le cas ou le panier serait vide qui redirigerait vers la page home.  <mat-card \*ngIf="!cart.items.length">    <p class="max-w-7xl p-3 pt-5 text-center">      Votre panier est vide      <button class="ml-2" mat-raised-button routerLink='/home'>Continuer achats</button>    </p>  </mat-card>  -Ensuite j’ai créé un tableau toujours à l’aide de angular material qui allait retourner :   * L’image * Le nom de l’article * Le prix * La quantité * Le total   -Enfin j’allais mettre une colonne avec des boutons et des icônes      <table mat-table [dataSource]="dataSource" class="mat-elevation-z8 w-full">  <ng-container matColumnDef="product">        <th mat-header-cell \*matHeaderCellDef> Produit</th>        <td mat-cell \*matCellDef="let element">         <img src="{{ element.product }}" alt="product" class="w-[100px] my-5 ml-3" />        </td>        <td mat-footer-cell \*matFooterCellDef>          <button mat-raised-button routerLink="/home" class="ml-3">Continuer Achats  </button>        </td>      </ng-container>  J’ai repeté le même type de row plusieurs fois. En y injectant des fonctions désignant la future action demandée. Exemple avec la fonction getTotal qui prend en paramètre un array du cartItem créé dans le modèle.    getTotal(items: Array<CartItem>): number {     return this.cart.getTotal(items);    }  -Ensuite j’ai laissé ces fonctions pour le moment et je suis allé créer le modèle des produits appelé product models que je pourrai ainsi appelé dans mes fonctions. J’ai fais un faux produit encore dans le product-box-component.ts et je suis remonté en haut de l’arborescence pour créer l’évènement addTocart    <app-product-box            (addToCart)="onAddToCart($event)"            class="w-full"            [fullWidthMode]="cols === 1"            >          </app-product-box>  -A ce moment j’ai créé le cartService dans un fichier services.  Je l’ai ajouté dans les providers de l’app module.  J’ai commencé par me servir de l’observable BehaviorSubject qui prend en paramètre une valeur initiale, ici un array des items vide, et qui émet la « current value »  Je rajoute dans le constructor la snackBar qui s’activera à chaque élément ajouté.    cart = new BehaviorSubject<Cart>({ items: []});      constructor(private \_snackBar: MatSnackBar) { }  Ensuite j’ai définis la fonction addToCart()    addToCart(item: CartItem): void {      const items = [...this.cart.value.items];      const itemInCart = items.find((\_item) => \_item.id === item.id);        if (itemInCart) {        itemInCart.quantity += 1;      } else {        items.push(item);      }      this.cart.next({ items });      this.\_snackBar.open("L'article a bien été ajouté !", 'ok', { duration: 3000 });    }  -Ici je fais 2 constantes. Une qui set la valeur des « items » du caddie et l’autre qui va chercher s’il y a le produit dans le caddie  S’il y a un produit dans le caddie  Je rajoute 1 au panier  Puis je push le produit sélectionné dans l’items array et j’active la snackbar  -Ensuite je l’importe dans la page home dans le constructor et côté front.  Ensuite je créer la fonction onAddOnCart quue j’injecte côté front aussi    onAddToCart(product: Product): void {     this.cartService.addToCart({      product: product.image,      name: product.title,      price: product.price,      quantity: 1,      id: product.id,     });    }  Je vais vers le header maintenant afin de créer l’animation sur le badge car à ce moment la valeur de la quantité du produit qui augment n’est pas encore visible.  Pour la bonne pratique je mets un underscore avant l’élément cart pour souligner qu’il est privé. Je set l items quantity à 0.  Dans un input je vais mettre un getter et un setter qui va parcourir le tableau avec l’aide de la méthode .map puis avec la méthode .reduce qui va traiter chaque valeur pour la réduire à une seule.    private \_cart: Cart = { items: [] };    itemsQuantity = 0;    @Input()    get cart(): Cart {     return this.\_cart    };      set cart(cart: Cart) {      this.\_cart = cart;      this.itemsQuantity = cart.items       .map((item) => item.quantity)       .reduce((prev, current) => prev + current , 0)     };       constructor(private cartService: CartService) {}  Puis je l’injecte côté front avec du conditionnel      <mat-icon        [matBadge]="itemsQuantity"        [matBadgeHidden]="!itemsQuantity"        matBadgeColor="warn"        >shopping\_cart      </mat-icon>  Puis je peaufine le front en lui mettant la même fonction getTotal créée précédemment par exemple dans le service donc je la récupère depuis ce service  -Ensuite j’ai créé une nouvelle fonction onClearCart qui va vider le panier. Je la créer dans le service aussi et je m’en sers pour simplifier la méthode.     onClearCart() {      this.cartService.clearCart();     }  Ensuite je suis remonté en haut de l’application pour créer un observable qui transmettra les valeurs.  Je me sers du cartService puis me sers de la souscription dans un ngOnInit. Enfin je la rentre dans mon composant app-header.  @Component({    selector: 'app-root',    template: `    <app-header [cart]="cart"></app-header>    <router-outlet></router-outlet>    `,    styles: []  })  export class AppComponent implements OnInit{    cart: Cart = { items: []};    constructor(private cartService: CartService) {}    ngOnInit() {     this.cartService.cart.subscribe((\_cart) => {      this.cart = \_cart;     });   }  }  -A ce moment du projet je peux ajouter et supprimer les éléments dans mon panier et le vider. Il faut maintenant aller sur la page cartComponent et souscrire à ce qu’on vient de créer pour que cette page récupère les informations qui transites.    ngOnInit():void {      this.dataSource = this.cart.items;      this.cartService.cart.subscribe((\_cart: Cart) => {        this.cart = \_cart;        this.dataSource = this.cart.items;      })  Maintenant les données sont transmises et on peut voir les éléments choisis dans cette page aussi.  Ensuite je créer les méthodes dans le cartService    addToCart(item: CartItem): void {      const items = [...this.cart.value.items];      const itemInCart = items.find((\_item) => \_item.id === item.id);        if (itemInCart) {        itemInCart.quantity += 1;      } else {        items.push(item);      }      this.cart.next({ items });      this.\_snackBar.open("L'article a bien été ajouté !", 'ok', { duration: 3000 });    }    removeQuantity(item: CartItem): void {      let itemForRemoval: CartItem | undefined;      let filteredItems = this.cart.value.items.map((\_item) => {        if (\_item.id === item.id) {            \_item.quantity--;          if(\_item.quantity === 0) {              itemForRemoval = \_item;           }        }        return \_item;      })      if (itemForRemoval) {        filteredItems = this.removeFromCart(itemForRemoval, false);      }      this.cart.next({ items: filteredItems });      this.\_snackBar.open("L'article à été correctement supprimé !", 'Ok', {duration: 3000});    }    getTotal(items: Array<CartItem>): number {      return items.map((item) => item.price \* item.quantity)      .reduce((prev, current) => prev + current, 0);     }     clearCart(): void {      this.cart.next ({ items: []});      this.\_snackBar.open('Le panier à été vidé avec succès !', 'Ok', { duration: 3000 });     }     removeFromCart(item: CartItem, update = true): Array<CartItem> {       const filteredItems = this.cart.value.items.filter(        (\_item) => \_item.id !== item.id)        if(update) {          this.cart.next({items: filteredItems});          this.\_snackBar.open("Les articles ont été correctement supprimés !", 'Ok', {duration: 3000});        }        return filteredItems;     };  que j’appelle dans des fonctions dans le cartComponent comme ceci :    getTotal(items: Array<CartItem>): number {     return this.cartService.getTotal(items);    }    onRemoveFromCart(item: CartItem): void {      this.cartService.removeFromCart(item);    }    onClearCart(): void {      this.cartService.clearCart();    }    onRemoveQuantity(item: CartItem): void {      this.cartService.removeQuantity(item);    }    onAddQuantity(item: CartItem): void {      this.cartService.addToCart(item);    }  Ici fût la partie la plus longue et ardue car il a fallu trouver les bonnes méthodes au bon endroit en prenant en compte les bons paramètres.  /\* -------- \*/  Ensuite pour définir les produits je suis allé les chercher sur fakestoreapi.com qui met à disposition des produits au format json. Pour cela je commence par créer un service store.service et à l’importer dans le provider de l’app module.  Le storeservice possèdera 2 fonctions principales dont on se servira pour « get » les catégories et les produits.    Ensuite je suis allé sur le composant home ou j’ai d’abord défini la fonction getProducts()  Puis soumis une propriété productsSuscription à une souscription puis j’ai parametré l’app onInit et onDestroy    productsSubscription: Subscription | undefined;    getProducts(): void {      this.productsSubscription = this.storeService.getAllProducts(this.count, this.sort, this.category)      .subscribe((\_products) => {        this.products = \_products;      })    }    ngOnInit(): void {      this.getProducts();    }    ngOnDestroy(): void {      if (this.productsSubscription) {        this.productsSubscription.unsubscribe();      }    }  Ensuite j’ai créé des outputs et des fonctions dans le product-header pour trier les éléments.    @Output() itemsCountChange = new EventEmitter<number>();    @Output() sortChange = new EventEmitter<string>();     sort = 'desc';     itemsShowCount = 12;       onSortUpdated(newSort: string): void {      this.sort = newSort;      this.sortChange.emit(newSort);     }     onItemsUpdated(count: number): void {      this.itemsShowCount = count;      this.itemsCountChange.emit(count);     }    Et finalement j’ai importé toutes mes fonctions sur le front et le tout était joué    <mat-drawer-content class="p-4">      <app-products-header      (sortChange)="onSortChange($event)"      (itemsCountChange)="onItemsCountChange($event)"      (columnsCountChange)="onColumnsCountChange($event)"></app-products-header>      <mat-grid-list        gutterSize="16"        [cols]="cols"        [rowHeight]="rowHeight"        >        <mat-grid-tile class="grille" \*ngFor="let product of products">          <app-product-box            [product]="product"            (addToCart)="onAddToCart($event)"            class="w-full"            [fullWidthMode]="cols === 1"            >          </app-product-box>        </mat-grid-tile>      </mat-grid-list>    </mat-drawer-content>  Ensuite j’ai rajouté une page de résumé de commande ainsi qu’une page de payement fictive qui pourrait être développée avec stripe par exemple qui mets à disposition des systèmes de payement.  Pour la mise en ligne j’ai build mon app et je l’ai déployée sur byethost qui est un hébergeur gratuit. | | | | | |
|  | | | | | |
|  | | | | |
|  | | | | | |
| **2. Précisez les moyens utilisés :** | | | | | |
|  | | | | | |
| Je me suis servi d’angular material ainsi que de la doc angular tout au long de ce projet.  Je me suis aidé de tutos et de différents projets angular.  Je me suis servi des cours mis à disposition par l’école STUDI.  J’ai utilisé tailwind pour le style et fake store api pour définir des produits fictifs. | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
| **3. Avec qui avez-vous travaillé ?** | | | | | |
|  | | | | | |
| J’ai travaillé seul | | | | | |
|  | | | | | |
|  | | | | |
|  | | | | | |
| **4. Contexte** | | | | | |
|  | | | |  | |
| Nom de l’entreprise, organisme ou association | | | | *STUDI* | |
| Chantier, atelier, service | | | Exercice | | |
| Période d’exerciceDu : *05/12/2023*au :*30/12/2023* | | | | | |
|  | | | | |
|  | | | | |
| **5. Informations complémentaires** *(facultatif)* | | | | |
|  | | | | | |
| Le site est disponible sur :  http://totystore.byethost18.com/home  Le git du site est : | | | | | |
|  | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Activité-type** | **3** | **Cliquez ici pour entrer l’intitulé de l’activité** | | | |
| ***Exemple n° 1***  | | *Cliquez ici pour entrer l’intitulé de l’exemple* | | | |
|  | | | | | |
|  | | | | | |
| **1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :** | | | | | |
|  | | | | | |
| Ce projet est un blog ou les utilisateurs pourront voir les posts, puis s’ils s’authentifies, ils pourront créer des posts, les modifier et les supprimer. Mais aussi mettre à jour leur profil en mettant une photo.  -Je vais me servir de REACT pour ce projet. En utilisant mongodb comme DataBase.  -J’ai commencé par rédiger les modèles que j’allais implémenter sur figma(création des tables). Puis j’ai créé les mockups à la main en me faisant une idée générale de ce que j’allais faire :   * Une navbar au top * Une sidebar * Un espace réservé au posts * Un espace réservé à l’utilisateur * Une page fictive contact etc.   Pour commencer ce projet j’ai initié un projet react. Je vais m’occuper du front end Dans un premier temps. J’ai d’abord créé la navbar que j’ai mis tout en haut de l’arborescence puisque toujours présente. Ensuite j’ai créé les pages qui allaient constituer ce projet à savoir :  -Home  -Contact  -Login  -Register  -Settings(Mise à jour du profil)  -Single  -Write  -Ensuite j’ai fait un fichier component dans lequel j’ai créé :  -Header  -Post  -Posts  -Sidebar  -singlePost  Pour les créer je les mets dans l’app directement puis lorsqu’ils sont tous créés je les imbrique dans les pages où ils doivent être.  Je m’occupe bien évidement du responsive en jouant avec les propriétés flex de css.  Une fois que le front était fait avec les systèmes de route Je me suis attelé au backend ou j’ai dans un premier temps créé un dossier api dans lequel j’installe express mongoose dotenv pour la sécurité à ma connexion de base de données, multer, nodemon etc ...  Puis dans l’index.js je vais les initialiser, les configurer et paramétrer. En externalisant la connexion à la base de donnée dans un fichier .env que je précise dans le gitignore pour que ces données sensibles ne puissent pas apparaitre.  Ensuite je vais créer les models user et posts et les routes users, posts et auth.  En exemple ce model postshema :  const mongoose = require(« mongoose ») ;  const PostSchema = new mongoose.Schema(    {      title : {        type : String,        required : true,        unique : true,      },      desc : {        type : String,        required : true,      },      photo : {        type : String,        required : false,      },      username : {        type : String,        required : true,      },    },    { timestamps : true }  ) ;  module.exports = mongoose.model(« Post », PostSchema) ;  Ensuite j’ai créé les routes dans lesquelles j’ai créé les méthodes associées nécessaires. Pour les posts par exemple je vais créer un crud complet grâce aux méthodes js post, put delete et get.  Const router = require(« express »).Router() ;  const User = require(« ../models/User ») ;  const Post = require(« ../models/Post ») ;  //CREATE POST  router.post(« / », async (req, res) => {    const newPost = new Post(req.body) ;    try {      const savedPost = await newPost.save() ;      res.status(200).json(savedPost) ;    } catch (err) {      res.status(500).json(err) ;    }  }) ;  //UPDATE POST  router.put(« / :id », async (req, res) => {    try {      const post = await Post.findById(req.params.id) ;      if (post.username === req.body.username) {        try {          const updatedPost = await Post.findByIdAndUpdate(            req.params.id,            {              $set : req.body,            },            { new : true }          ) ;          res.status(200).json(updatedPost) ;        } catch (err) {          res.status(500).json(err) ;        }      } else {        res.status(401).json(« You can update only your post ! ») ;      }    } catch (err) {      res.status(500).json(err) ;    }  }) ;  //DELETE POST  router.delete(« / :id », async (req, res) => {    try {      const post = await Post.findById(req.params.id) ;      if (post.username === req.body.username) {        try {          await post.delete() ;          res.status(200).json(« Post has been deleted... ») ;        } catch (err) {          res.status(500).json(err) ;        }      } else {        res.status(401).json(« You can delete only your post ! ») ;      }    } catch (err) {      res.status(500).json(err) ;    }  }) ;  //GET POST  router.get(« / :id », async (req, res) => {    try {      const post = await Post.findById(req.params.id) ;      res.status(200).json(post) ;    } catch (err) {      res.status(500).json(err) ;    }  }) ;  module.exports = router ;  -Pour le router auth je vais me servir de bcrypt qui est un générateur de hash et je m’en sers pour envoyer le mode passe à la base de données de cette manière.  //UPDATE  router.put(« / :id », async (req, res) => {    if (req.body.userId === req.params.id) {      if (req.body.password) {        const salt = await bcrypt.genSalt(10) ;        req.body.password = await bcrypt.hash(req.body.password, salt) ;      }      try {        const updatedUser = await User.findByIdAndUpdate(          req.params.id,          {            $set : req.body,          },          { new : true }        ) ;        res.status(200).json(updatedUser) ;      } catch (err) {        res.status(500).json(err) ;      }    } else {      res.status(401).json(« You can update only your account ! ») ;    }  }) ;  Pour vérifier le bon fonctionnement de mes router je vais me servir de Postman.  Pour la photo je me sers de multer pour uploader une image et la stocker dans un fichier qui sera dédié.  -S’en fût terminé pour l’api. J’ai commencé par ajouter le proxi et à installer axios comme client http dans mon projet.  -Ensuite dans la page home je vais utiliser un useEffect qui va get les posts et que je vais passer au composant en props  export default function Home() {    const [posts, setPosts] = useState([]) ;    useEffect(() => {      const fetchPosts = async () => {        const res = await axios.get(« /posts » + search) ;        setPosts(res.data) ;      } ;      fetchPosts() ;    }, []) ;    return (      <>        <Header />        <div className= »home »>          <Sidebar />          <Posts posts={posts} />        </div>      </>    ) ;  }  -Ensuite je vais dans mon composant posts pour maper les « post » que je renvoie à l’enfant encore en le mettant en props.  -Fichier POSTS :    <div className= »posts »>        {posts.map((p, i) => (          <Post key={i} post={p}/>        ))}      </div>  -Fichier POST :  import « ./post.css » ;  import { Link } from « react-router-dom » ;  export default function Post({ post }) {    const PF = « http ://localhost:5000/images/ » ;    return (      <div className= »post »>        <Link to={`/post/${post.\_id}`} className= »link postImg »>          {post.photo && <img className= »postImg » src={PF + post.photo} alt= » » />}        </Link>        <div className= »postInfo »>          <Link to={`/post/${post.\_id}`} className= »link »>            <span className= »postTitle »>{post.title}</span>          </Link>          <hr />          <span className= »postDate »>            {new Date(post.createdAt).toDateString()}          </span>        </div>        <p className= »postDesc »>{post.desc}</p>      </div>    ) ;  }  -Maintenant je vais aller vers le composant single post qui retournera seulement le post sélectionner et je les sets à l’aide de useEffect dans des fonctions asynchrones.    Const [post, setPost] = useState({}) ;  const [title, setTitle] = useState(«  ») ;    const [desc, setDesc] = useState(«  ») ;    useEffect(() => {      const getPost = async () => {        const res = await axios.get(« /posts/ » + path) ;        setPost(res.data) ;        setTitle(res.data.title) ;        setDesc(res.data.desc) ;      } ;      getPost() ;    }, []) ;  puis les transmets aux front :    <div className= »singlePostInfo »>            <span className= »singlePostAuthor »>              Auteur :              <Link to={`/ ?user=${post.username}`} className= »link »>                <b className= »user »> {post.username}</b>              </Link>            </span>            <span className= »singlePostDate »>              {new Date(post.createdAt).toDateString()}            </span>          </div>            <div className= »inputs »>              <textarea                className= »singlePostDescInput »                rows={5}                value={desc}                onChange={€ => setDesc(e.target.value)}              />            </div>  -Vous pouvez aussi constater que j’ai créé la date grâce à new Date simplement.  -Après ça je suis allé m’occuper de l’authentification. Dans le fichier register.jsx, j’ai commencé par initialiser des useState.    Const [username, setUsername] = useState(«  ») ;    const [email, setEmail] = useState(«  ») ;    const [password, setPassword] = useState(«  ») ;    const [error, setError] = useState(false) ;  Puis j’ai créé la fonction asynchrone qui allait envoyer la réponse. Je rajoute un message d’erreur à l’utilisateur en cas de problème.     Const handleSubmit = async € => {      e.preventDefault() ;      setError(false) ;      if(username.length > 3 && email.length > 3 && password.length > 3 ) {      try {        const res = await axios.post(« /auth/register », {          username,          email,          password,        }) ;        res.data && window.location.replace(« /login ») ;      } catch (err) {        setError(true) ;      }      } setError(true) ;    } ;  Puis je rajoute ce code cette fonction dans les inputs exemple :          <input            type= »password »            className= »registerInput »            placeholder= »Entrer mot de passe... »            onChange={€ => setPassword(e.target.value)}          />          <button className= »registerButton » type= »submit »>            S’inscrire          </button>        </form>        <button className= »registerLoginButton »>          <Link className= »link » to= »/login »>            Se connecter          </Link>        </button>        {error && <span style={{color : »red », marginTop : »10px »}}>Une érreur s’est produite, veillez à saisir plus de 3 caractères</span>}  -Pour la suite, je vais avoir besoin de créer un contexte que je pourrais utiliser en ternaire pour l’affichage de mes composants en fonction de si l’utilisateur est connecté. Je créer donc un fichier contexte dans lequel je mets un fichier actions.js, un fichier context.js et un reducer.js.  -Pour commencer je vais créer un state initial que je vais rentrer en paramètre dans le hook createContext de react. Ensuite je vais utiliser useEffect pour stocker en localstorage  const INITIAL\_STATE = {    user : JSON.parse(localStorage.getItem(« user »)) || null,    isFetching : false,    error : false,  } ;  export const Context = createContext(INITIAL\_STATE) ;    useEffect(() => {      localStorage.setItem(« user », JSON.stringify(state.user)) ;    }, [state.user]) ;  Ensuite je vais créer les actions (fonctions) dont je pourrais me servir puis je set les paramètres du reducer pour définir pour chacune des actions si le User existe ou non.  -Voilà ce que donne par exemple le LoginSucces dans le fichier actions.js :  export const LoginStart = (userCredentials) => ({    type : « LOGIN\_START »,  }) ;  export const LoginSuccess = (user) => ({    type : « LOGIN\_SUCCESS »,    payload : user,  }) ;  export const LoginFailure = () => ({    type : « LOGIN\_FAILURE »,  }) ;  -Puis dans le fichier reducer.js :  const Reducer = (state, action) => {    switch (action.type) {      case « LOGIN\_START » :        return {          user : null,          isFetching : true,          error : false,        } ;      case « LOGIN\_SUCCESS » :        return {          user : action.payload,          isFetching : false,          error : false,        } ;  -Ensuite je retourne dans le contextprovider pour créer ce composant.  Export const ContextProvider = ({ children }) => {    const [state, dispatch] = useReducer(Reducer, INITIAL\_STATE) ;    return (      <Context.Provider        value={{          user : state.user,          isFetching : state.isFetching,          error : state.error,          dispatch,        }}      >        {children}      </Context.Provider>    ) ;  } ;  -Ensuite, pour que le contextprovider soit utilisable dans toute l’app je remonte l’injecter tout en haut de l’arborescence dans l’index.js  -Je peux ensuite retourner à la page login pour créer mes fonctions qui vont permettre à l’utilisateur de se connecter et de changer le payload dans un deuxième temps come ceci    const userRef = useRef() ;    const passwordRef = useRef() ;    const { dispatch, isFetching } = useContext(Context) ;    const handleSubmit = async € => {      e.preventDefault() ;      dispatch({ type : « LOGIN\_START » }) ;      try {        const res = await axios.post(« /auth/login », {          username : userRef.current.value,          password : passwordRef.current.value,        }) ;        dispatch({ type : « LOGIN\_SUCCESS », payload : res.data }) ;      } catch (err) {        dispatch({ type : « LOGIN\_FAILURE » }) ;      }    } ;  A noter que je me sers de useRef pour me baser sur les paramètres éxistants.  J’y intègre dans le front de cette manière    -Ensuite je me suis occupé de la fonction de logout dans la topbar avant d’aller sur la partie crud du projet dans la page d’édition des posts dans la page write ou je vais créer la fonction handlesubmit.    Const [title, setTitle] = useState(«  ») ;    const [desc, setDesc] = useState(«  ») ;    const [file, setFile] = useState(null) ;  const handleSubmit = async € => {      e.preventDefault() ;      const newPost = {        username : user.username,        title,        desc,      } ;      if (file) {        const data = new FormData() ;        const filename = Date.now() + file.name ;        data.append(« name », filename) ;        data.append(« file », file) ;        newPost.photo = filename ;        try {          await axios.post(« /upload », data) ;        } catch (err) {}      }      try {        const res = await axios.post(« /posts », newPost) ;        window.location.replace(« /post/ » + res.data.\_id) ;      } catch (err) {}    } ;  Ensuite je me suis occupé des boutons de modification du post et de suppression dans le composant singlepost.jsx.    const handleDelete = async () => {      try {        await axios.delete(`/posts/${post.\_id}`, {          data : { username : user.username },        }) ;        window.location.replace(« / ») ;      } catch (err) {}    } ;    const handleUpdate = async () => {      try {        await axios.put(`/posts/${post.\_id}`, {          username : user.username,          title,          desc,        }) ;        setUpdateMode(false)      } catch (err) {}    } ;  -Ensuite je suis allé sur la page settings qui servira à l’utilisateur pour ajouter une photo de profil et mettre à jour l’email et le password de son compte.    Const handleSubmit = async € => {      e.preventDefault() ;      dispatch({ type : « UPDATE\_START » }) ;      const updatedUser = {        userId : user.\_id,        email,        password,      } ;      if (file) {        const data = new FormData() ;        const filename = Date.now() + file.name ;        data.append(« name », filename) ;        data.append(« file », file) ;        updatedUser.profilePic = filename ;        try {          await axios.post(« /upload », data) ;        } catch (err) {        }      }      if (email.length > 3 && password.length > 3) {      const res = await axios.put(« /users/ » + user.\_id, updatedUser) ;      setSuccess(true) ;      dispatch({ type : « UPDATE\_SUCCESS », payload : res.data }) ;      alert(‘Profil mit à jour’)      } else {      dispatch({ type : « UPDATE\_FAILURE » }) ;      setError(true) ;      }    }  -Pour les derniers réglages je m’occupe du responsive à l’aide de flex-direction column-reverze pour afficher les posts les plus récents en premier. Le site est responsif et maintenant on peut s’enregistrer puis se connecter.  -Lorsque nous sommes connectés nous pouvons créer modifier et supprimer nos posts. Puis nous pouvons mettre à jour notre profil en mettant toujours plus de 3 lettres dans les inputs ainsi qu’une photo de profil.  -Mon site est à disposition de qui voudra l initialiser il suffit juste de mettre sa MONGO\_URL dans un fichier .env ainsi que de faire un npm start et le tour est joué. | | | | | |
|  | | | | | |
|  | | | | |
|  | | | | | |
| **2. Précisez les moyens utilisés :** | | | | | |
|  | | | | | |
| -Je me suis servi de vs code, de mongodb, de postman, de tutos youtube, de la documentation javascript. Je me suis servi des cours de studi aussi ainsi que différents projets réalisés en amont.  -Je me sers de figma pour créer les models qui mêmes s’ils sont du nombre de 2 existent et sont dans le projet. | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
| **3. Avec qui avez-vous travaillé ?** | | | | | |
|  | | | | | |
| J’ai travaillé seul | | | | | |
|  | | | | | |
|  | | | | |
|  | | | | | |
| **4. Contexte** | | | | | |
|  | | | |  | |
| Nom de l’entreprise, organisme ou association | | | | *STUDI* | |
| Chantier, atelier, service | | | *Cliquez ici pour taper du texte.* | | |
| Période d’exerciceDu : *10/08/2022*au :*10/09/2022* | | | | | |
|  | | | | |
|  | | | | |
| **5. Informations complémentaires** *(facultatif)* | | | | |
|  | | | | | |
| Le git de ce site qui possède un readme avec les instructions pour faire marcher ce site en local est le suivant : <https://github.com/Toty-01/blog-app> | | | | | |
|  | | | | | |

|  |  |  |
| --- | --- | --- |
| **Titres, diplômes, CQP, attestations de formation** | | |
|  | | |
| *(facultatif)* | | |
| **Intitulé** | **Autorité ou organisme** | **Date** |
| Cliquez ici. | Cliquez ici pour taper du texte. | Cliquez ici pour sélectionner une date. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |
| --- |
| **Déclaration sur l’honneur** |
|  |
|  |

Anthony Cauche

Je soussigné(e) [prénom et nom] ,

déclare sur l’honneur que les renseignements fournis dans ce dossier sont exacts et que je suis l’auteur(e) des réalisations jointes.

Ambronay

20/02/2023

Fait à le

pour faire valoir ce que de droit.

Signature :

|  |
| --- |
| **Documents illustrant la pratique professionnelle** |
|  |
| *(facultatif)* |
| **Intitulé** |
| Cliquez ici pour taper du texte. |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
| **Annexes** |
|  |
| *(Si le RC le prévoit)* |