TP Angular

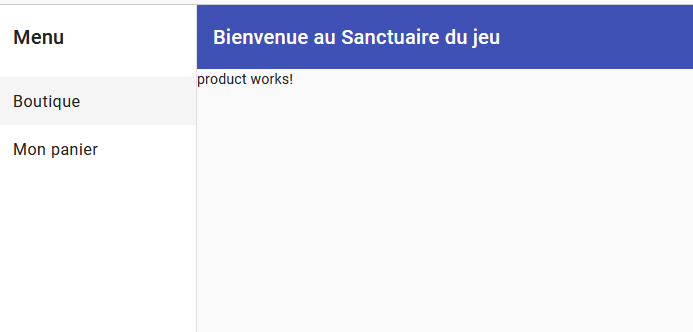
Création d’une application de e-commerce

# TP 1 : Créer une application et un premier composant

* Créer une application Angular avec le cli d’Angular.
* Remplacer la page de démonstration par un texte personnalisé « Bienvenue sur le site e-commerce ».
* Créer un composant « product » qui servira à afficher un produit (le laisser tel quel pour l’instant).
* Afficher le composant « product » dans votre page.
* Lancer l’application.

# TP 2 : Ajouter la lib angular material

* Ajouter la bibliothèque Angular Material.
* Créer un composant « menu » avec le schematic « navigation » d’Angular Material (voir le getting started d’Angular Material).
* Remplacer le contenu de la page principale par ce composant.
* Modifier le titre dans la toolbar par « Bienvenue au Sanctuaire du jeu ».
* Modifier le contenu du menu (avec les libellés « Boutique » et « Mon Panier »).
* Utiliser la directive « ng-content » au sein de votre composant « menu » pour projeter son contenu (composant « product »), en remplacement du commentaire <!-- Add Content here -->. *Voir capture.*

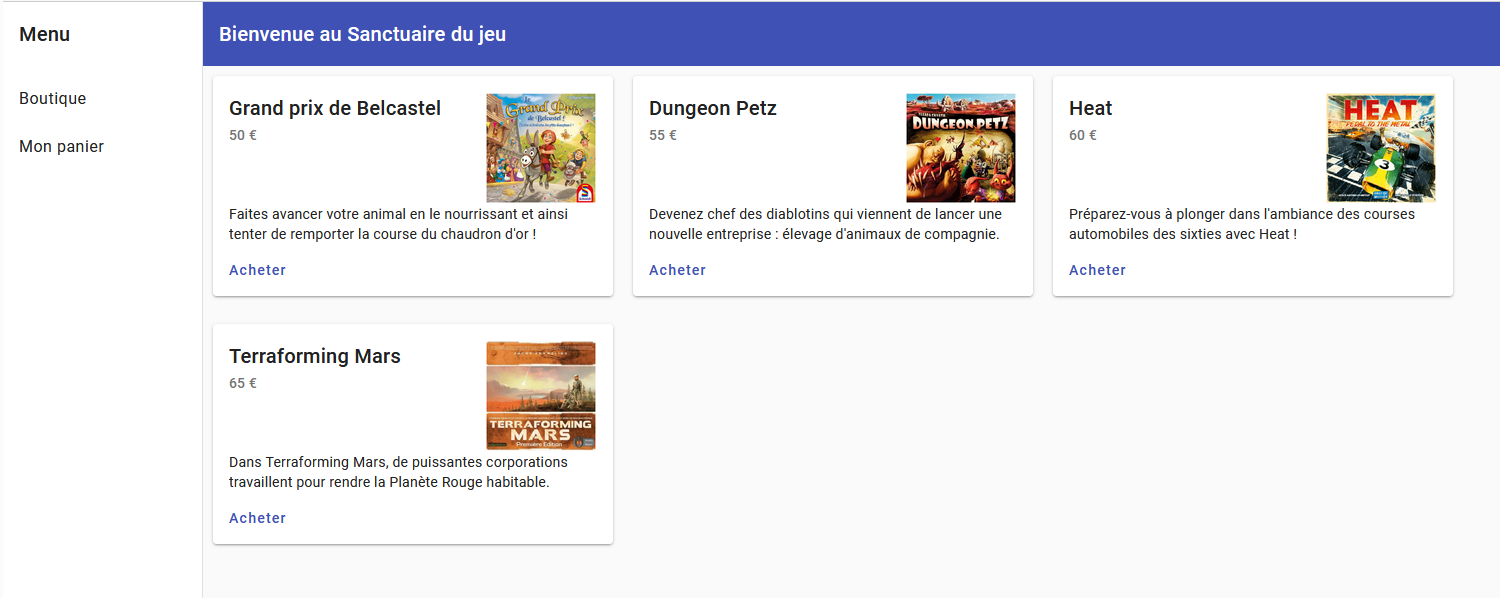


* Ajouter un dossier « img » dans les assets du projet, et y ajouter les images nécessaires à l’application (voir « *Assets TP 1* »).
* Copier/coller les Importer le MatCardModule dans votre app.module.ts.
* Avec les données « *Assets TP 1* », modifier votre composant *product* pour afficher une carte (mat-card) qui affiche un titre, un sous-titre avec le prix, une image à droite, une description et un bouton (colorisé) « acheter ». *Voir capture.*



# TP3 : Rendre dynamique votre composant et afficher une liste de produit

* Créer une interface « Product » avec les champs suivants :
* id : Entier
* title : Texte
* description : Texte
* photo : Texte
* price : Entier
* stock : Entier
* Dans AppComponent, créer des entrées pour notre composant « product » pour définir le nom, la description, le prix et la photo. *Voir « Assets TP 3 ».*
* Afficher plusieurs produits grâce à une boucle « ng-for ».
* Ajouter les règles CSS nécessaires pour que les cards s’affichent comme dans la capture ci-dessous.



* Afficher au dessus de la liste une carte qui affichera un titre « Mon panier » en titre et en-dessous, un paragraphe avec le total du panier.
* Lors du clic sur le bouton « acheter », ajouter à une variable locale le prix du produit.
* Si le total est de 0, afficher un texte (« Vous n'avez pas encore d'articles dans votre panier »), sinon le total du panier.

*L’application est maintenant capable d'afficher une liste de produits, d'ajouter un produit dans un panier et de calculer le prix total du panier. Cependant, nous pouvons encore ajouter un nombre infini de produits alors que nous n'en avons qu'un nombre limité en stock.*

* Lorsque vous cliquez sur Acheter, décrémenter le stock.
* Si un produit à un stock proche de 1, ajouter une classe CSS pour modifier le style (mettre un background rouge par exemple)
* Si le stock est de 0, ne plus afficher le produit.

# Simuler un serveur (pré-requis au TP4)

* Développer un backend rest pour récupérer la liste des produits
* Liberté sur le choix de l’outil (java/spring, node/express, …)
* Voici la spécification :
  + GET */rest/products*
    - Retourne la liste des produits
    - Format d’un produit :
      * title : texte
      * description : texte
      * photo : url
      * price : nombre
      * stock : nombre
  + GET */rest/products/:id*
    - Retourne un produit
  + POST */rest/basket*
    - Ajouter un produit dans le panier
    - Décrémenter le stock du produit
    - En entrée, un produit
  + GET */rest/basket*
    - Retourne la liste des produits ajoutés dans le panier
  + POST */rest/basket/confirm*
    - Valider le formulaire pour envoyer une commande
    - Simuler l’envoie d’une facture (imprimée dans les logs les infos, et les produits)
    - Réinitialiser le panier en ligne
    - Format d’entrée :
      * name : texte
      * address : texte
      * creditCard : texte

# TP4 Ajouter un service et un pipe

* Ajouter un service *ProductService.*
* Déplacer la liste des produits dans ce service.
* Ajouter les méthodes suivantes à ce service :
* **getProducts**

*Récupérer tous les produits.*

* **getProduct**

*Récupérer un produit.*

* **decreaseProduct**

*Décrémenter le stock d’un produit.*

* **hasLimitedStock**

*Savoir si un produit est limité (stock égal à 1).*

* **hasNoStock**

*Savoir si un produit est limité (stock égal à 0).*

* Ajouter un service *BasketService*.
* Ajouter les méthodes suivantes à ce service :
* **addToBasket**

*Ajouter un produit au panier.*

* **getProductFromBasket**

*Récupérer le total d’un produit du panier.*

* **getBasket**

*Récupérer tous les produits du panier.*

* Utiliser maintenant ces services dans le AppComponent.
* Utiliser le *CurrencyPipe* pour afficher et formater le prix de vente en VF. *Faire appel à « registerLocaleData(localeFr) » (@angular/common/locales/fr).*

# TP 5 Utiliser le serveur http

* Ajouter le module *HttpClientModule* dans le *AppModule*.

*Note : Par convention, les observables sont préfixés ou suffixés par « $ ».*

* Dans le ProductService, remplacer la liste des produits par un appel au serveur http.
* Dans le BasketService, appeler le serveur http lors de l’ajout d’un produit dans le panier.
* Utiliser un BehaviorSubject pour garder la liste des produits présents dans le panier.
* Convertir la méthode « getBasket() » pour initialiser le panier via un appel au serveur.
* Ajouter les méthodes suivantes au BasketService :
* **loadBasket**

*Récupérer le panier depuis le serveur.*

* **getTotalBasket**

*Calculer le prix total du panier.*

Utiliser un opérateur RXJS pour calculer le prix total.

# TP 6 Utiliser le router

* Ajouter un module **router** au AppModule :

ng generate module app-routing --flat --module=app

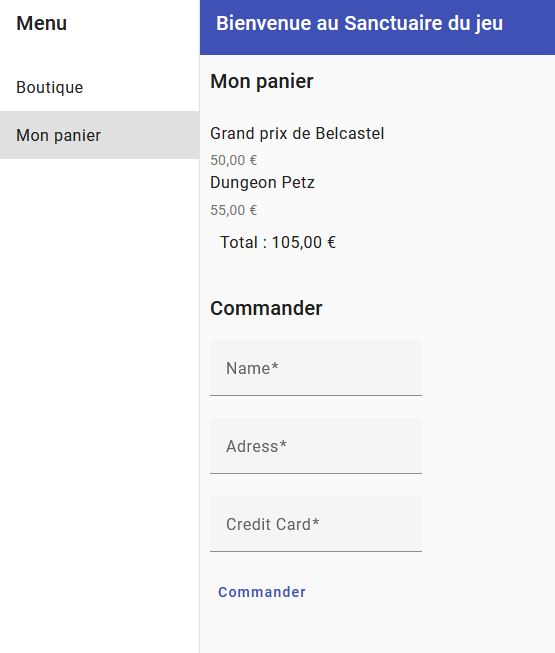
* Créer un composant *ProductPage*. Celui-ci sera en charge de l'affichage de la page d'accueil de notre application, sauf du menu qui est un élément commun à toutes les pages. Déplacez-y tout le code nécessaire.
* Remplacer le code à l’intérieur du composant menu dans le fichier « app.component.html » par un <router-outlet>.
* Créer la route par défaut qui afficher le composant *ProductPage*.
* Créer un composant *BasketPage* qui affichera le panier.
* Ajouter la route « basket » qui affiche le composant *BasketPage*.
* Modifier les liens du menu (« Mon panier » et « Boutique ») pour pouvoir naviguer d’une page à l’autre.
* Dans *BasketPage*, afficher la liste des Produits achetés.



* Ajouter un guard pour interdire la navigation vers la page du panier si le panier est vide.
* Bonus : Ajouter une page de détail d’un produit avec un clic sur le titre du produit pour y avoir accès.

# TP 7 Ajouter un formulaire

* Dans *BasketPage*, ajouter un formulaire avec trois champs (Nom, Adresse, Carte de crédit).
* Créer une interface « ClientForm » avec les champs suivants :
* name : Texte
* address : Texte
* creditCard : Texte



* La validation du formulaire doit appeler le serveur avec les informations pour valider la commande (au retour, on revient sur la page de la boutique, le panier est maintenant vide).
* Les trois champs sont obligatoires, et le numéro de carte de crédit doit être composé uniquement de 10 chiffres.
* Ajouter un message d’erreur sous chaque champ (le message de s’affiche que si il y a une erreur et que l’utilisateur a interagi avec le champ).
* Afficher une boîte de dialogue de confirmation.

# TP 8 Test E2E

* Ajouter cypress sur le projet (Suivre le getting started).
* Créer un premier test e2e vide. Ajoutez-y les éléments suivants :
* Vérifier qu'on peut cliquer sur un bouton "Acheter" sur la page d'accueil
* Vérifier qu'au clic sur un bouton "Acheter", le total est bien mis à jour sur la page d'accueil
* Vérifier que sur la page "Panier" je retrouve bien mon produit sur lequel j'ai cliqué sur "Acheter"
* Valider le formulaire et vérifier qu’on arrive bien sur la page boutique et que le panier est vide.