

1. Pourquoi les "tables" sous MongoDB sont-elles appelées des collections ? (0,5 points)

Car une collection en programmation est une liste d'objets.

2. Sous MongoDB, quelle méthode permet de modifier la valeur d'une propriété d'un document ?

La méthode **update()** (ou une variante de cette méthode).

3. Qu'est-ce un service web ? (1 point)

Dans le cadre de l'architcture MEAN, un service web est un traitement serveur permettant l'échange de données (généralement formattées en JSON) avec le client, via des requêtes HTTP.

4. Comment les objets JavaScript insérés dans le corps d'une requête HTTP de méthode POST sont-ils manipulables dans un service web géré par Node.js ? (1 point)

Via l'objet **body** attribut de l'objet correspondant à la requête (*req.body*).

5. Dans l'architecture MEAN, où est géré le "Modèle" du paradigme de conception MVC ? (1,5 point)

Le modèle MVC correspond aux données et aux traitement métiers appliqués à celles-ci, et donc :

- dans les bases MongoDB de l'application ;
- dans les traitements effectués par le serveur Node ;
- dans les traitements sur les données (hors présentations) effectués par les composants Angular.

6. Dans une application Angular, à quels moments les instances des classes TypeScript sont-elles allouées ? (1,5 points)

- lors de la prise en compte des sélecteurs des composants ;
- lors des injections de dépendances (si premières instanciations) ;
- lors de l'invocation d'un composant via le routeur.

7. Ecriture d'une méthode d'un composant Angular (2 points)

Ecrivez la méthode *ngOnInit()* de la classe *ProduitsComponent* qui value son attribut *produits* en utilisant la méthode *getProduits()* du service *ProduitsService*.

```
ngOnInit() {  
    this.produitsService.getProduits().subscribe(produits => {  
        this.produits = produits;  
    })  
}
```

8. Ecriture d'un template (2 points)

Ecrivez le template associé à la classe *ProduitsComponent* qui affiche dans une liste HTML le nom et le marque des produits. La valeur de la propriété *stock* sera également affichée à condition que l'observable *admin* contienne une valeur différente de chaîne vide.

```
<ul>  
    <li *ngFor="let produit of produits"> {{produit.nom}} {{produit.marque}}  
        <span *ngIf="!(admin|async)"> {{produit.stock}} </span> <!-- autre balise possible -->  
    </li>  
</ul>
```

Quel est le problème de ce fragment de code (hormis l'éventuelle homonymie) ? (1.5 points)

```
app.get("/panier/:nom/:prenom", (req, res) => {
  db.collection("membres").find({"nom":req.params.nom, "prenom":req.params.prenom})
    .toArray((err, documents) => { let panier = [];
      db.collection("paniers").find({"email":documents[0].email}).toArray((err, documents2) => {
        panier = documents2[0].produits;
      });
      res.setHeader("Content-Type", "application/json");
      res.end(JSON.stringify(panier));
    });
});
```

Le problème est que, dû à l'asynchronisme de JavaScript, la liste *panier* a de fortes probabilités d'être renvoyée avant que le second appel à la base de données ait fourni un résultat.

Le fait que la variable panier soit initialisée en tant que liste puis qu'elle prenne directement une valeur n'est pas un problème en soi (cette initialisation étant un moyen d'indiquer dans le code son type).

Que résoud ce code et que se passe-t-il si l'instruction next() est oubliée ? (1.5 points)

```
app.use(function (req, res, next) {
  res.setHeader('Access-Control-Allow-Origin', '*');
  res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE');
  next();
});
```

Ce code résoud les problèmes de CORS (Cross-Origin Resource Sharing) qui surviennent quand un code chargé sur le client accède à des données provenant d'un serveur qui n'est pas celui qui a fourni ce code au client. Si l'instruction *next()* est oubliée, l'entête HTTP du résultat est bien modifiée, mais la main n'est pas redonnée à la méthode qui renvoie le résultat.

Décrivez (en quelques mots à chaque fois) les trois manières par lesquelles les objets sont indirectement instanciés dans une application Angular ? (3 points)

- via l'utilisation du sélecteur d'un composant
- via l'invocation d'un composant par une route
- via une injection de dépendance (ce qui revient au même que déclarer un service comme provider)

Communication composant père vers composant fils (2 points)

Soit cette balise intégrant un composant "fils" dans le template d'un composant "père" :

```
<compteur secondes="{{nombresSecondes}}"/>
```

Ecrivez le fragment de code qui dans la classe du composant fils permet de récupérer la valeur de l'attribut *secondes*.

```
@Input() secondes: number;
```

Voici le code de la classe AuthenticationService vue en cours ? (2 points)

```
private user:Subject<string> = new BehaviorSubject<string>(undefined);
constructor(private http: HttpClient) { }
getUser() { return this.user; }
connect(data: string) { this.user.next(data); }
disconnect() { this.user.next(null); }
verificationConnexion(identifiants): Observable<any> {
  return this.http.post('http://localhost:8888/membre/connexion', JSON.stringify(identifiants), options);
}
```

- *undefined* pourrait-il être remplacé par autre chose ?
oui mais seulement par "" (cette valeur correspondant à l'adresse mail de l'utilisateur connecté, et au lancement de l'application, personne n'est connectée par défaut)
- A quels endroits de l'application, l'observable créé dans cette classe va-t-il être utilisé ?
par tous les composants ayant besoin de savoir si l'internaute est identifié (et notamment dans leurs templates)
(et bien sûr la valeur de cet observable est modifiée lors de la connexion - et donc de la soumission du formulaire de connexion - ou de la déconnexion de l'internaute).

1. Qu'est-ce qu'est une application monopage ? Quels sont ses avantages et inconvénients ? (1.5 points)

Une SPA est une web app qui est présentée à l'utilisateur comme une seule page html

*Avantages : - navigation plus rapide car on ne charge que ce qui change
- Le serveur est déchargé de la tache des vues.*

*Inconvénients : - l'historique est nul
- chargement initial long
- problème lié au SEO (Search Engine Optimisation)*

2. Sous MongoDB dans l'objet filtre, comment sélectionner les documents qui ne possèdent pas la propriété *myprop* ? (1 point)

`db.nomCollection.find({ "myprop": { $exists: false } });`

3. Quelles sont les trois principales forces d'une application Angular ? (1 point)

- Angular fait du DI, data binding, gestion de state, outils de gestion de route intégré
- création de composants qui permet la modularité et la réutilisation
- Angular CLI qui permet de créer des projets, ajouter des fichiers, débuguer, tester et déployer l'app.

4. Comment déclare-t-on en TypeScript une collection ? (0.5 point)

`public nomCollection: type Collection<T> = [{ id: string; val1: ...; valn: ...; }]`

Id collection = _____

5. Que signifie exactement une erreur CORS (1 point)

survient quand un code chargé sur le client accède à des données provenant d'un autre serveur que celui qui a fourni le code au client

6. Dans le paradigme MVC, à quoi correspond le contrôleur ? (0.5 point)

il négocie les actions soumises par l'utilisateur, puis demande au modèle d'opérer les actions nécessaires pour réaliser cet action (à travers les events)

7. Dans le contexte du routeur Angular, quelle différence il y-a-t-il entre une route principale et une route secondaire ? (1 point)

on ne peut utiliser qu'une route principale mais on peut utiliser plusieurs routes secondaires en les nommant différemment

8. Comment implémente-t-on proprement la gestion des rôles dans une application Angular (1 point)

En gérant l'authentification des utilisateurs et en utilisant les authentifications guards.

9. Un composant peut-il être invoqué simultanément dans plusieurs outlets ? (0.5 point)

Si à condition que ça soit dans un <router-outlet> nommé correspondant à une route auxiliaire

10. Pourquoi utilise-t-on l'injection de services sous Angular ? (1 point)

Pour factoriser certaines fonctionnalités ou d'accéder à un état permettant aux composants de communiquer entre-eux.

11. Quelle est l'erreur dans ce fragment de code ? (1 point)

```
export class MembresService {
  constructor(private http: HttpClient) { }
  getMembres(): Observable<any> { return http.get("http://localhost:8888/membres"); }
}
```

il manque le this dans return this....

1. Interrogation sous Mongo (1,5 points)

Dans la ligne suivante qui interroge une collection *Produits* sous Mongo

```
db.Produits.find(...)
```

remplacer les ... par le code sélectionnant les documents qui possèdent une propriété *prix* dont la valeur est comprise entre 10 et 100.

```
db.Product.find({prix: {$gte: 10, $lte: 100}});
```

2. Commentez ce fragment de code utilisé par un serveur Node.js (2 points)

```
app.get("/produits/:nom", (req, res) => {
  db.collection("Produits").find({"nom":req.params.nom}).toArray((err, documents)=> {
    let collectionVendeurs = [];
    for (let doc of documents) {
      db.collection("vendeurs").find({"email":doc.vendeur}).toArray((err, documents)=> {
        collectionVendeurs.push(documents[0]);
      });
    }
    res.setHeader("Content-type", "application/json");
    res.end(JSON.stringify(collectionVendeurs));
  });
});
```

Permet de récupérer la liste des vendeurs d'un produit et de les renvoyer sous format JSON au client

Voir annale 16/01/2019 pour plus de détail

3. Quelle différence conceptuelle il y-a-t-il entre composants et modules dans Angular ? (1 point)

Modules consist of one or more components. They do not control any html. Your modules declare which components can be used by components belonging to other modules, which classes will be injected by the dependency injector and which component gets bootstrapped. Modules allow you to manage your components to bring modularity to your app.

4. Interrogation autour de l'injection de dépendances ? (1 point)

Pourquoi un composant Angular va-t-il utiliser le service *HttpClient* ainsi :

```
...
export class monService {
  constructor(private http: HttpClient) { }
...
}
```

alors que la programmation objets "classique" nous pousserait à le mettre en oeuvre comme ceci :

```
...
export class monService {
  private http = new HttpClient();
...
}
```

cf annale 16/01/2019 4)

5. Interrogation autour des sélecteurs ? (1 point)

Le sélecteur *app-timer* possède un attribut *format* dont la valeur peut être soit la chaîne de valeur *symbolique* ou *numérique*. Parmi les possibilités suivantes, quelles sont celles syntaxiquement correctes (et pourquoi) ?

```
<app-timer format="numerique" />
<app-timer format="numerique" />
<app-timer [format]="numerique" />
<app-timer [format]="'numerique'" />
```

<app-timer format="numerique" />: permet de fixer l'attribut *format*

<app-timer [format]="'numerique'" />: on on injecte directement la chaîne et en utilisant le *property binding*

6. Qu'implique la présence de la balise *<router-outlet>* dans un template ? (1 point)

il existe un system de routing et ça implique qu'on injecte

7. Comment est mis en œuvre un guard *CanActivate* ? (1 point)

```
const appRoutes: Route[] = [
  {
    path: 'date-time-form',
    component: DateTimeFormComponent
  },
  {
    path: '',
    redirectTo: 'posts',
    pathMatch: 'full'
  }
];

@NgModule({
  declarations: [
    UserSearchComponent,
    DateTimeFormComponent
  ],
  imports: [
    BrowserModule,
    ReactiveFormsModule,
    RouterModule.forRoot(appRoutes)
  ],
  providers: [
    PostHehe,
    UserService
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }

const httpOptions = {
  headers: new HttpHeaders({
    "Access-Control-Allow-Methods": "GET,POST",
    "Access-Control-Allow-Headers": "Content-type",
    "Content-Type": "application/json",
    "Access-Control-Allow-Origin": "*"
  })
};

@Injectable({
  providedIn: 'root'
})
export class PanierService {
  private urlBase: string = 'http://localhost:8888/';
  private user: string = "undefined";

  constructor(private authService: AuthentificationService,
              private http: HttpClient) {
  }

  this.authService.getUser().subscribe((user :string) => {
    this.user = user;
  });

  getPanier(email : string): Observable<any> {
    let emailObj = {"email": email};
    let url = this.urlBase+'panier/';
    console.log("Dans le service PanierService avec "+ url + " et " + email);
    return this.http.post(url, JSON.stringify(emailObj), httpOptions);
  }
}
```

```
<div class="container">
  <div class="col-xs-12">
    <h2>Posts</h2>
    <div class="list-group">
      <app-post-list-item *ngFor="let post of posts" [post]="post">
        </app-post-list-item>
      </div>
    </div>
  </div>
</div>
```

1. Interrogation sous MongoDB (1 point)

Dans la ligne suivante qui interroge une collection nommée *Phones* sous Mongo

```
db.Phones.find(...)
```

remplacez les ... par le code sélectionnant les documents qui possèdent une propriété nommée *network* dont la valeur est une des trois suivantes : *3G*, *4G* ou *5G*.

2. Commentez ce fragment de code Node.js : son but, ce qu'il fait... (3 points)

```
app.get("/livre/:nom", (req, res) => {
  db.collection("Livres").find({ "nom": req.params.nom }).toArray((err, documents) => {
    let collectionAuteurs = [];
    for (let doc of documents) {
      for (let auteur of doc.auteurs) {
        db.collection("Auteurs").find({ "email": auteur }).toArray((err, documents) => {
          collectionAuteurs.push(documents[0]);
        });
      }
    }
    res.setHeader("Content-type", "application/json");
    res.end(JSON.stringify(collectionAuteurs));
  });
});
```

3. Qu'est-ce qu'est une application web monopage ? (1 point)

4. Interrogation autour de l'injection de dépendances (1 point)

Pourquoi un composant Angular va-t-il utiliser le service *HttpClient* ainsi :

```
...
export class monService {
  constructor(private http: HttpClient) { }
  ...
}
```

alors que la programmation objets en TypeScript nous pousserait à le mettre en oeuvre comme ceci :

```
...
export class monService {
  private http = new HttpClient();
  ...
}
```

5. Quel est le rôle de la balise <router-outlet> ? (1 point)

6. Qu'est-ce un guard, où est-il mis en oeuvre ? (1 point)

7. XSL/XPath (2 points)

Ecrivez le ou les templates d'une feuille de style XSL qui attachée à n'importe quel document balisé affiche le nombre de toutes les occurrences des éléments du DOM correspondant à des balises qui ont le même nom que les balises filles de la balise top-level. Par exemple attachée au document XML suivant :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<A>
  <B/>
  <A> <A/> <B/> </A>
  <B> <B/> <C/> </B>
</A>
```

elle devrait afficher :

B : 4
A : 3

Dans une expression XPath, un comptage est réalisé par la fonction *count()*.

Le noeud *contexte* (celui qui est évalué par l'expression Xpath) est désigné par un point.

Le noeud *courant* (celui qui a invoqué le template) est désigné par la fonction *current()*.

Et le nom de quelque chose par une fonction intuitive...

- 1) db.phone.find ({"network": {\$in: ["3G", "4G", "5G"]}})
- 2) Ce code à pour but de recuperer la liste des auteurs d'un livre dont le nom est passé en paramètre de la requête ce comme suit:
 - recuperer sous forme de tableau l'ensemble des documents ayant la propriété nom qui possède comme valeur le nom passé en paramètre dans la requête.
 - Pour chaque auteur d'un document, on récupère ses informations sous forme d'un tableau en utilisant la fonction "find" et un filtre sur l'email puis on stocke le document récupéré dans la collection "CollectionAuteur".
 - À la fin, on retourne la collection "CollectionAuteur" sous format JSON.
- 4) Pour que le service http ne soit pas instancier par la classe MonService mais par le framework d'injection de dépendance, ce dernier pourra instancier un nouvel objet ou utiliser une instance existante. DI est un design pattern qui consiste à séparer l'instanciation d'une dépendance et son utilisation. Permet d'inverser les dépendances; d'éviter le couplage fort; de factoriser l'instanciation; pour faciliter le remplacement.
- 5) La balise <router-outlet> permet d'indiquer l'emplacement de l'insertion d'un component, ainsi en fonction de la route visitée, le composant associé sera injecté en-dessous de la balise <router-outlet>
- 6) Un guard est un code utilisé pour la vérification et qui est exécuté avant que l'utilisateur puisse accéder à une route.

```

export const appRouteList = [
  {
    path: 'cart',
    loadChildren: './views/cart/cart-routing.module#CartRoutingModule',
    canActivate: [
      IsSignedInGuard
    ]
  }
]

constructor(private _router: Router,
            private _session: Session) {
}

canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
  const isSignedIn = this._session.isSignedIn();

  if (isSignedIn !== true) {
    this._router.navigate([...]);
  }

  return isSignedIn;
}

```

(dans un fichier is-signed-in.guard.ts) (modifié)

Exam 2016

- 1) fonction de callback : fonction passé en argument d'une autre fonction et qui sera appelé à la fin de l'exécution de cette dernière.
- 2) vue : dans les templates html

contrôleur : dans les méthodes qui permettent de capturer les événements (fichier .ts)

Module : dans les services

```

<form [formGroup]="userForm" (ngSubmit)="trace($event); onSaveUser() "
id="FORM__userForm__a2c7e0ad-7380-40c5-a9dc-873d6859763b">
  <div class="form-group">
    <label for="firstName">First Name*</label>
    <input type="text" class="form-control" id="firstName"
formControlName="firstName" (focusout)="trace($event);"
data-eventful-widget="true">
  </div>
  <button class="btn btn-success" type="submit" [disabled]="userForm.invalid"
data-eventful-widget="true" id="BUTTON__Create_User__fe93e3b0-5024-4304-
a10a-9064e99d3610">Create User</button>
</form>

```

```

getIngredients(): Observable<Ingredient[]> {
  return this.http.get<Ingredient[]>(this.urlBase+'ingredients');
}

```

```

@Component({
  selector: 'app-panier',
  templateUrl: './panier.component.html',
  styleUrls: ['./panier.component.css']
})
export class PanierComponent implements OnInit {
  public contenu: any

  public message : string = "";

  public user: BehaviorSubject<string>;

  constructor(
    private route: ActivatedRoute,
    private authService: AuthentificationService,
    private panierService: PanierService) {
    console.log("Dans le constructeur du composant panier")
    this.user = this.authService.getUser();
  }

  ngOnInit() {
    let email = this.user.value;
    console.log("Dans ngOnInit de PanierComponent");
    if (email!=="undefined"){
      console.log("fetching panier for "+ email);
      this.panierService.getPanier(email).subscribe((panier) => {
        if (panier['resultat'] ==1){
          this.contenu = panier['panier'].produits;
          console.log(this.contenu)
        }
      });
    }
  }

  supprimer(indice: number){
    console.log("should delete product number "+ indice);
    this.contenu.splice(indice,1);
  }

  savePanier(){
    console.log("Should save panier");
    this.panierService.updatePanier({"email":this.user.value, "contenu":this.contenu}).subscribe(reponse=> {
      if (reponse['code'] == 0){
        console.log("Erreur in updating");
      }
      else console.log("Panier updated");
    })
  }

  submitPanier(){
    console.log("Submitting panier");
    this.panierService.supprimerPanier().subscribe((reponse : any)=> {
      console.log(reponse)
      if (reponse['resultat']){
        this.message = "Commande validée!";
      }
    });
  }
}

```

```

export const coursesRouterConfig = [
  {
    path: 'courses',
    children: [
      {
        path: '',
        component: Courses
      },
      {
        path: ':id',
        component: CourseDetail,
        canActivate: [CanCourseDetailActivate],
        canDeactivate: [CanCourseDetailDeactivate]
      }
    ]
  },
  {
    path: 'playlist',
    component: Playlist,
    outlet: 'playlist'
  }
];

<div class="main-container">
  <div class="list">
    <router-outlet></router-outlet>
  </div>
  <div class="list">
    <router-outlet name="playlist"></router-outlet>
  </div>
</div>

```

```

<form [FormGroup]="userForm" (ngSubmit)="trace($event); onSaveUser()" id="FORM_userForm_a2c7e0ad-7380-40c5-a9dc-873d6859763b">
  <div class="form-group">
    <label for="firstName">First Name*</label>
    <input type="text" class="form-control" formControlName="firstName" (focusout)="trace($event);" data-eventful-widget="true">
  </div>
  <button class="btn btn-success" type="submit" [disabled]="" data-eventful-widget="true" id="BUTTON_Create_User_fe93e3b0-5024-4304-a10a-9064e99d3610">Create User</button>
</form>

```

De quelle / Réponse

```
<?php
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

$request = Request::createFromGlobals();
$name = $request->get('name');
$response = new Response();

$response->setContent(
    '<html><body>Hello'
    . $name
    .'</body></html>'
);
$response->setStatusCode(Response::HTTP_OK);
$response->headers->set('Content-Type', 'text/html');

// Retourne une réponse HTTP valide
$response->send();

<?php
use Symfony\Component\HttpFoundation\Request;

// Récupération des valeurs accessibles dans les super variables
$request = Request::createFromGlobals();

// Récupérer l'url
$request->getPathInfo();

// récupérer des attributs en GET ou POST
$request->query->get('name');
$request->request->get('name', 'nom par défaut');

$request->getMethod(); // e.g. GET, POST, PUT, DELETE ou HEAD
```

Contrôler

```
<?php
// src/Controller/HelloController.php
class HomeController extends AbstractController
{
    /**
     * Page d'accueil
     *
     * @Route('/home', name='accueil')
     */
    public function home()
    {
        return new Response(" Bienvenue sur la page d'accueil ! ");
    }

    /**
     * Page d'accès à un article
     *
     * @Route('/article/{articleId}', name='show-article')
     */
    public function show($articleId)
    {
        // Nous retrouvons la valeur de la variable $articleId à partir de l'URI
        // Par exemple /article/1234 => $articleId = '1234'

        return new Response(" Voici le contenu de l'article avec l'ID $articleId
    );
    }
}
```

Le framework Symfony est construit autour du paradigme fondamental du web : un utilisateur fait une requête et le serveur (ici, Symfony) doit retourner une réponse.

Le composant HttpFoundation fournit une abstraction PHP objet pour la requête et la réponse.

Le composant HttpKernel a la responsabilité de récupérer la requête de l'utilisateur et de renvoyer une réponse.

Dans la très grande majorité des applications, la réponse du serveur sera différente selon l'URL à laquelle il sera accédé et fera appel à une fonction différente pour retourner un résultat.

Un contrôleur Symfony peut être une simple fonction d'une classe PHP et il est possible de configurer le routing à l'aide d'annotations PHP même si d'autres formats de déclaration sont possibles.

Le framework Symfony non seulement a un composant pour gérer le routing, mais fournit aussi un contrôleur frontal en charge de recevoir toutes les requêtes de l'utilisateur et de trouver la bonne action (fonction) du contrôleur à exécuter.

Souscripteurs d'événements

```
<?php
// src/EventSubscriber/SummaryMailSubscriber
namespace App\EventSubscriber;

use Symfony\Component\EventDispatcher\EventSubscriberInterface;
use Symfony\Component\HttpKernel\Event\PostResponseEvent;
use Symfony\Component\HttpKernel\KernelEvents;

class SummaryMailSubscriber implements EventSubscriberInterface
{
    public static function getSubscribedEvents()
    {
        return [
            // 'kernel.terminate'
            KernelEvents::TERMINATE => [
                ['sendProductPaidMail', 0],
            ],
        ];
    }

    public function sendProductPaidMail(PostResponseEvent $event)
    {
        // envoi de l'email
    }
}
```

Nous avons d'abord vu comment déléguer la construction et la récupération de nos objets au container de services.

Grâce à l'autowiring, l'essentiel du temps, nous n'avons rien de spécial à faire pour que nos objets soient automatiquement retrouvés par le container et accessibles dans nos services et nos contrôleurs.

L'autoconfiguration permet d'ajouter des tags à nos services s'ils implémentent une interface spécifique et les services tagués sont traités différemment par le framework.

Ensuite, nous avons vu comment utiliser la programmation événementielle à l'aide du composant EventDispatcher.

Le principal intérêt est que l'on peut changer le comportement d'une application sans en changer le code, et ajouter de nombreux comportements sur une même action, sans pour autant que ces comportements soient liés entre eux.

Nous avons vu que Symfony dispose de nombreux événements natifs qui sont envoyés aux écouteurs durant le cycle de vie de l'application.

Enfin, il est également possible de créer et de "dispatcher" ses propres événements métiers.

Il faut créer l'événement qui doit "implémenter" la classe Event de Symfony, à laquelle on peut passer des informations au besoin. Ensuite, il suffit de faire appel à l'EventDispatcher pour envoyer l'information à tous les écouteurs concernés.

- Controller front permet de retourner une réponse à partir de la requête d'un user ; de demander et configurer le kernel
- Rôle du routeur Symfony : il relie une URL à n'importe quel fonction PHP
- Event dispatcher : permet à different partie indépendante de communiquer entre-elles ; Alterer le comportement de l'application à tout moment du cycle ; c'est un émetteur d'évent qui peut propager nos propres events
- Container de service Symfony : il conserve en un seul endroit les règles de construction d'un obj

Twig

```
{# Ceci est un exemple de gabarit Twig #}
{% set collection = [1, 2, 3] %}

<ul>
{% for item in collection %}
    <li>{{ item }}</li>
{% endfor %}
</ul>

{#
    Va afficher seulement ce code HTML:

    <ul>
        <li>1</li><li>2</li><li>3</li>
    </ul>
#}

{% for key, element in elements %}
    {% if loop.index % 2 %}
        Element pair
    {% else %}
        Element impair
    {% endif %}
{% else %}
    Il n'y a aucun élément à afficher.
{% endfor %}
```

Twig est un moteur de gabarit intégré au framework Symfony, avec une syntaxe claire et sécurisée par défaut.

Les fonctions sont utiles pour l'intégration de vos pages, par exemple dump dont nous avons parlé dans un chapitre précédent.

Les filtres sont utiles lorsque vous souhaitez changer l'affichage de vos données, et ils peuvent être chaînés.

Les macros permettent d'automatiser toutes les tâches d'intégration répétitives qui ne nécessitent aucun calcul : il faudra importer la macro dans chaque gabarit où vous en aurez besoin.

Il est possible de créer ses propres fonctions et filtres grâce à l'autoconfiguration du container de services.

Il suffira de créer une classe qui étend AbstractExtension et les extensions seront disponibles dans vos gabarits.

Enfin, Twig supporte l'héritage : ceci permet de mettre les blocs de vos pages communes (haut de page, bas de page) dans les gabarits parents et de seulement surcharger des parties spécifiques du gabarit.

L'intégration devient plus facile, car il y a moins de gabarits à maintenir et ils sont plus configurables.

Création formulaire

```
class ArticleType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('title')
            ->add('content', TextareaType::class)
            ->add('author', TextType::class)
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            'data_class' => Article::class,
        ]);
    }
}
```

Utilisation formulaires et rendu dans twig

```
class FormController extends AbstractController
{
    /**
     * @Route("/form/new")
     */
    public function new(Request $request){
        $article = new Article();
        $article->setTitle('Hello World');
        $article->setContent('Un très court article.');
        $article->setAuthor('Zozor');

        $form = $this->createForm(ArticleType::class, $article);

        $form->handleRequest($request);

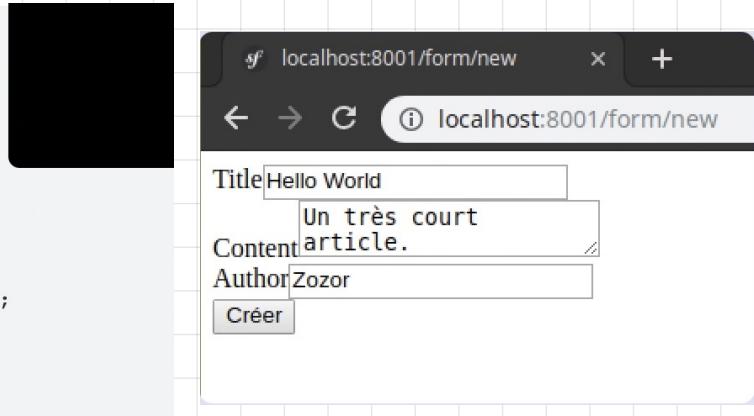
        if ($form->isSubmitted() && $form->isValid()) {
            dump($article);
        }

        return $this->render('default/new.html.twig', array(
            'form' => $form->createView(),
        ));
    }
}

(modifié)

{{-- templates/default/new.html.twig --}}
<html>
    <head></head>
    <body>
        {{ form_start(form) }}
        {{ form_row(form.title) }}
        {{ form_row(form.content) }}
        {{ form_row(form.author) }}

        <button type="submit" class="btn btn-primary">Créer</button>
    {{ form_end(form) }}
</body>
</html>
```



Validator

```
<?php

use Symfony\Component\Validator\Validation;
use Symfony\Component\Validator\Constraints\Length;
use Symfony\Component\Validator\Constraints\NotBlank;
use App\Entity\Article;

$article = new Article();
$validator = Validation::createValidator();
$violations = $validator->validate($article, [
    new Length(['min' => 10]),
    new NotBlank(),
]);

if (0 !== count($violations)) {
    // Affiche les erreurs
    foreach ($violations as $violation) {
        echo $violation->getMessage(). '<br>';
    }
}
```

Le résultat de cette fonction est une liste de violations qui permettent d'accéder aux messages d'erreurs correspondant aux contraintes qui n'ont pas été validées. Si la valeur est valide, alors la liste de violations est vide.

Les formulaires Symfony permettent :

en créant des Types de modéliser des champs ou aggrégations de champs

de réaliser une double liaison (binding) entre la valeur de l'entité et celle dans le champ (value HTML) puis lors de la soumission, la valeur saisie ou modifiée est liée à l'entité qui est modifiée

de créer des formulaires sophistiqués emboîtés avec des sous-formulaires multiples (Collection) en utilisant du JavaScript

d'effectuer un rendu graphique personnalisable (CSS) grâce à Twig {{form_row}} . . .

d'utiliser des contraintes de validation, le service Validator, de gérer les erreurs de validation . . .

Doctrine

```
<?php

namespace AppBundle\Entity;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity()
 * @ORM\Table(name="blog_article")
 */
class Article
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue(strategy="AUTO")
     * @ORM\Column(type="integer")
     */
    public $id;

    /**
     * @ORM\Column(type="string")
     */
    public $title;

    /**
     * @ORM\Column(type="text")
     */
    public $content;

    /**
     * @ORM\Column(type="datetime", name="date")
     */
    public $date;
}
```

insertion dans doctrine

```
/**
 * @Route("/form/new")
*/
public function new(Request $request)
{
    $article = new Article();
    $article->setTitle('Hello World');
    $article->setContent('Un très court article.');
    $article->setAuthor('Léa');

    $form = $this->createForm(ArticleType::class, $article);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $em = $this->getDoctrine()->getManager();

        $em->persist($article);
        $em->flush();
    }

    return $this->render('default/new.html.twig', array(
        'form' => $form->createView(),
    ));
}
```

Dans la plupart des projets Symfony, c'est l'ORM Doctrine qui est utilisé pour la gestion de vos objets en base de données. L'ORM est composée de deux objets principaux :

l'Entity Manager qui fournit des fonctions que l'on manipule pour créer, éditer, rechercher et supprimer nos objets métiers ; le Data Mapper qui s'occupe d'effectuer les requêtes auprès du système de gestion de base de données sélectionné (généralement MySQL).

Pour cela, à l'aide d'annotations, on indique à Doctrine ORM quels sont les objets que nous souhaitons manipuler en base et quelles relations existent entre eux. Elles sont de trois types principalement : les relations 1 à 1 ; les relations 1 à n ; les relations n à n.

Ensuite, le repository accessible pour toutes les entités par défaut fournit de nombreuses fonctions pour parcourir la liste d'objets stockés en base : il retourne soit l'objet métier, soit un objet de type ArrayCollection qui contient une liste de nos objets métiers.

Enfin, il existe un plugin open source pour nous aider à gérer les mises à jour de base de données nécessaires à l'évolution de vos projets Symfony. L'extension DoctrineMigrationBundle permet, en quelques commandes, de générer des fichiers de migration et de les exécuter sans risques en production.

Relation m - m

```
/**
 * Un produit peut être mis dans plusieurs paniers
 * @ORM\ManyToMany(targetEntity="App\Entity\Cart", inverseBy="products")
 * @JoinTable(name="products_carts")
 */
private $carts;

/**
 * Les produits sont liés à un panier
 * @ORM\ManyToMany(targetEntity="App\Entity\Products", mappedBy="carts")
 */
private $products;
```

Relation 1-1 / 1-n

```
/**
 * Un client a potentiellement plusieurs adresses
 * @ORM\OneToOne(targetEntity="App\Entity\Address", mappedBy="customer")
 */
private $addresses;

/**
 * Il y a un seul panier possible par commande
 * @ORM\OneToOne(targetEntity="App\Entity\Cart")
 * @ORM\JoinColumn(name="cart_id", referencedColumnName="id")
 */
private $cart;
```

Suppression

```
/**
 * @Route("/form/delete/{id<\d+>}", methods={"POST"})
*/
public function delete(Request $request, Article $article)
{
    $em = $this->getDoctrine()->getManager();

    $em->remove($article);
    $em->flush();

    // redirige la page
    return $this->redirectToRoute('admin_article_index');
```

Configuration sécurité

config/security.yaml

```
security:
    encoders:
        App\Entity\User: bcrypt

    providers:
        database_users:
            entity: { class: App\Entity\User, property: username }

    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false

        main:
            pattern: ^
            anonymous: true
            logout:
                path: security_logout
            guard:
                authenticators:
                    - App\Security\FormLoginAuthenticator
```

Il y a deux parties à distinguer dans le processus de sécurité

l'authentification, permet d'identifier l'utilisateur :

l'autorisation, vérifie son identité et autorise, ou non, l'accès au contenu demandé.

Grâce au fichier de configuration security.yaml, vous pouvez paramétrier chaque élément de la sécurité, notamment :

le firewall pour l'authentification

le contrôle d'accès pour l'autorisation, en appliquant des règles sur des URL ou les méthodes des contrôleurs, selon vos besoins.

Vous pouvez définir des rôles pour les visiteurs pour établir des droits d'accès. Nous pouvons contrôler les autorisations d'un utilisateur authentifié partout dans nos applications : dans nos contrôleurs ; dans nos vues ; dans nos services...

et de façon très fine à l'aide du système de "Voteurs".

base.html.twig

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>{% block title %}Factorielle et combinaisons{% endblock %}</title>
    {%# Run `composer require symfony/webpack-encore-bundle`%
      and uncomment the following Encore helpers to start using Symfony UX %}
    {% block stylesheets %}
      {#{ {{ encore_entry_link_tags('app') }} #}}
    {% endblock %}

    {% block javascripts %}
      {#{ {{ encore_entry_script_tags('app') }} #}}
    {% endblock %}
  </head>
  <body>
    <div><h2>Calcul de n!</h2>
      <form method="get">
        <input type="number" name="k" min=0 step=1 placeholder="n"
          {% if k is defined %}
            value={{k}}
          {% endif %}
        />
        <input type="submit" value="OK" name="fact">
      </form>
    </div>
    <div><h2>Calcul des combinaisons de n éléments dans p places</h2>
      <form method="get">
        <input type="number" name="n" min=0 step=1 placeholder="n"
          {% if n is defined %}
            value={{n}}
          {% endif %}
        />
        <input type="number" name="p" min=0 step=1 placeholder="p"
          {% if p is defined %}
            value={{p}}
          {% endif %}
        />
        <input type="submit" value="OK" name="combi">
      </form>
    </div>
    {% block resultat %}{% endblock %}
  </body>
</html>
```

factucombi.html.twig

```
{% extends 'base.html.twig' %}

{% block resultat %}
<h2>Résultat </h2>
{% if p is defined %}
Combinaisons({{n}}, {{p}}) = {{r}}

{% elseif k is defined %}
Factorielle( {{k}} ) = {{ r }}

{% endif %}
{% endblock %}
```

FactController.php

```

<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;
use App\Service\HistoOp;
use App\Service\Mastermind;
use Symfony\Component\HttpFoundation\Session\SessionInterface;

class FactController extends AbstractController{
    /**
     * @Route("/fact/{n<\d+>?1}", name="fact")
     */
    public function fact($n): Response{
        return new Response(facto($n));
    }
    /**
     * @Route("/combi/{n<\d+>?1}/{p<\d+>?1}", name="combi")
     */
    public function combi($n,$p): Response{
        return new Response(facto($n)/(facto($p)*facto($n-$p)));
    }
    /**
     * @Route("/")
     */
    public function index(Request $request){
        // $request = Request::createFromGlobals();
        if (null !== $request->query->get('p') && null !== $request->query->get('n')) {
            $n=intval($request->query->get('n'));
            $p=intval($request->query->get('p'));
            $r=facto($n)/(facto($p)*facto($n-$p));
            return $this->render('factoCombi.html.twig',[

                'p' => $p,
                'n' => $n,
                'r' => $r
            ]);
        } else if (null !== $request->query->get('k')) {
            $k=intval($request->query->get('k'));
            $r=facto($k);
            return $this->render('factoCombi.html.twig',[

                'k' => $k,
                'r' => $r
            ]);
        } else {
            return $this->render('base.html.twig',[]);
        }
    }
}

```

①

```

    /**
     * @Route("/{x<\d+>?1}/{y<\d+>?1}")
     */
    function add($x,$y, HistoOp $histoop) : Response {
        $r=strval(intval($x)+intval($y));
        $histoop->addOp($x.'+'.y.'='.$r); // sauve
        return new Response($r);
    }
    /**
     * @Route("/histo/{action?}")
     */
    function histo(histoop $histoop, $action) : Response {
        if($action==null){
            $r="";
            foreach($histoop->getOps() as $op){
                // $r.= $op[0].$op[1].$op[2].'='.$op[3].'  
';
                $r.= $op."  
";
            }
        } elseif ($action=='clear'){
            $r=strval($histoop->clear()).' opérations supprimées !';
        }
        return new Response($r);
    }
    /**
     * @Route("/session/{k?}/{v?}")
     */
    function session($k,$v, SessionInterface $session) : Response {
        $r="";
        if($k==null){
            foreach($session->all() as $c => $w){
                if(is_array($w)){ // la valeur peut être un tableau
                    $s=""; // passé par réf à la callback pour "var_dump"
                    array_walk_recursive($w,function($v, $k) use(&$s){
                        $s .= $k . ":" . $v . ",";
                    });
                } else {
                    $s=strval($w);
                }
                $r .= strval($c)."=>".$s."  
";
            }
        }else{
            $session->set($k,$v);
        }
    }
}

```

②

```

// $session->set("nom", "michel");
// $r="";
// foreach($session->get('histoop')[0] as $ligne){
//   $r.= $ligne[1];
// }
return new Response($r);
//return new Response(implode(',', $session->get('histoop')[0]). $session-
>get("nom"));
}
/***
 * @Route("/master")
*/
public function master(SessionInterface $session){
if(!$session->has('mastermind')){
  $session->set('mastermind',new Mastermind()); // créé
}
$jeu=$session->get('mastermind',null); // on a un jeu !

$request = Request::createFromGlobals();
$prop = $request->query->get('prop');
if (null !== $prop){ // une proposition
  $r=$jeu->test($prop); // résultat
  if ($r==false){ // proposition invalide
    $message="$prop : proposition invalide !";
  } else if ($r['bon']!=$jeu->getTaille()){ // valide !
    $message="$prop : proposition valide : {$r['bon']} bien placé(s),
{$r['mal']} mal placé(s)";
  } else {
    $message="Félicitations, vous avez gagné !";
  }
  $session->set('mastermind', $jeu); // sauve le jeu
} else { // pas de proposition !
  $message = "Veuillez saisir une proposition S.V.P. !";
  $nouv = $request->query->get('nouveauJeu');
  if (null != $nouv){ // Nouveau Jeu
    $session->set('mastermind',new Mastermind(4)); // créé
    $jeu=$session->get('mastermind',null); // à supprimer
  }
}
return $this->render('mastermind.twig',[

  'jeu' => $jeu,
  'message' => $message
]);
}

/***
 * @Route("/sessionClear")
*/
function sessionClear(SessionInterface $session) : Response {
  $session->clear();
  return new Response("Session effacée");
}
}

function facto($n){
  if ($n <= 1) {
    return 1;
  } else {
    return $n*facto($n-1);
  }
}

function arrayToString($item, $key, &$r){
  $r .= $key . ":" . $item . ",";
}

```

(3)

(4)