FICHE MEMO

maquettage

papier/crayon
balsamiq
adobe XD
charte graphique
plan du site

html

W3C

<meta charset=UTF-8> affiche les caractères unicode.

Différence entre HTML4 et HTML5 => sémantique (SEO , text-to-speech)
=> après analyse du web (id,class) plusieurs mots-clés récurrents ont été retenus.

video
audio
localStorage
webSockets
canvas
svg

formulaire:

Avec html3 a été introduite la notion de formulaire ce qui a permis d'envoyer des données au serveur.
<form action"url" methos="post">
<input type =
text
textearea
number
file
submit
checkbox
hidden

https://developer.mozilla.org/fr/docs/orphaned/Web/Guide/HTML/HTML5

avec HTML5 d'autres types ont été rajoutés
email
date
datetime
tel
… etc …

https://developer.mozilla.org/fr/docs/Web/HTML/Element/Input

serviceWorkers
localStorage
webSockets
canvas
svg

css:

il peut y avoir des question avec **media-queries**
(genre: je change de couleur quand la résolution est inférieure à 900px)

HTML->5 polices par défaut
possibilité de charger en CDN les polices de google
uploader des polices et les utiliser avec la clé font-family de css3

bootstrap:

développé par twitter
basé sur flex
les 12 colonnes de grid (rows, col-12)

autres possibilités :

grid avec media-queries

autres frameworks responsive : foundation, bulma , tailwind …etc…

Javascript

JS : langage interprété, créé par netscape début des années 90.

Basé sur les évènements.
Reçois des événements du navigateur ET de votre application javascript (<script>)

Un classique: différence entre const, let et var

intérêts de Javascript côté client.

Permet d'ajouter de la logique à notre page internet (HTML et CSS n'étant que des languages de description).

Le moteur Javascript se trouve par défaut dans tous les navigateurs.
V8 Chrome, SpiderMonkey dans Firefox

À part le scripting classique (variables, tableaux une ou plusieurs dimensions, iterations, calculs …etc…) qu'on retrouve dans tous les langages interprétés.

1) **SELECTION : ACCEDER AUX ELEMENTS DU DOM** par leur tag (<p> <header>) et/ou par des attributs (id, class)

Permet d'accéder au Document Object Model représentant une page WEB

2) **ECOUTER LES EVENEMENTS** (du script et du navigateur)

Des "écouteurs d'évènements" peuvent être connecté à un ou des éléments du DOM en fonction de leur type d'élément (div, p, input) , classe et/ou id .

AddEventListener

onload
click
dble-click
mouseover
…etc…

Lorsque qu'un évènement se produit sur un élément l'action correspondante est déclenchée.

3) MODIFICATION DES ELEMENTS SUITE À un évènement

```
document.getElementById("unid").innerHTML = "contenu"
```

4) CREATION D'ELEMENT

changement de class cet cet élément ou d'autres
masquage/affichage d'un élément
chargement d'images
changement de look des éléments
redirection sur une autre page

5) AJAX

Asynchronous Javascript Xml

Javascript accède à une URL distante en mode pull (GET) ou en mode push (POST).
Permet de lire des données accessibles à partir d'une URL ou envoyer des données à une URL distante.

XMLHTTPRequest

Jquery:

$.ajax , $.post , $.get

fetch

axios

POO

Classe / Objet

encapsulation

private - protected - public

polymorphisme par héritage simple

Soit la classe véhicule_de_transport (notre d'abstraction, notion généraliste) et ses 2 classes filles qui en héritent voiture et avion .

véhicule_de_transport
-> voiture extends vehicule_de_transport
-> avion extends vehicule_de_transport

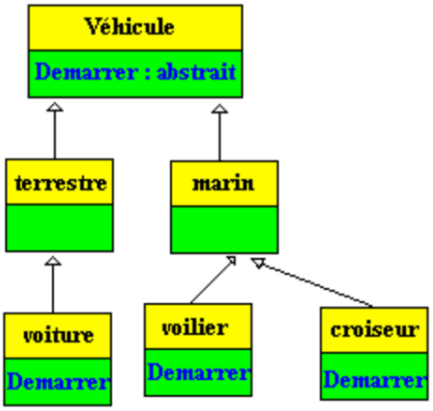
Le véhicule de transport prend plusieurs formes (polymorphe)

autre exemple:

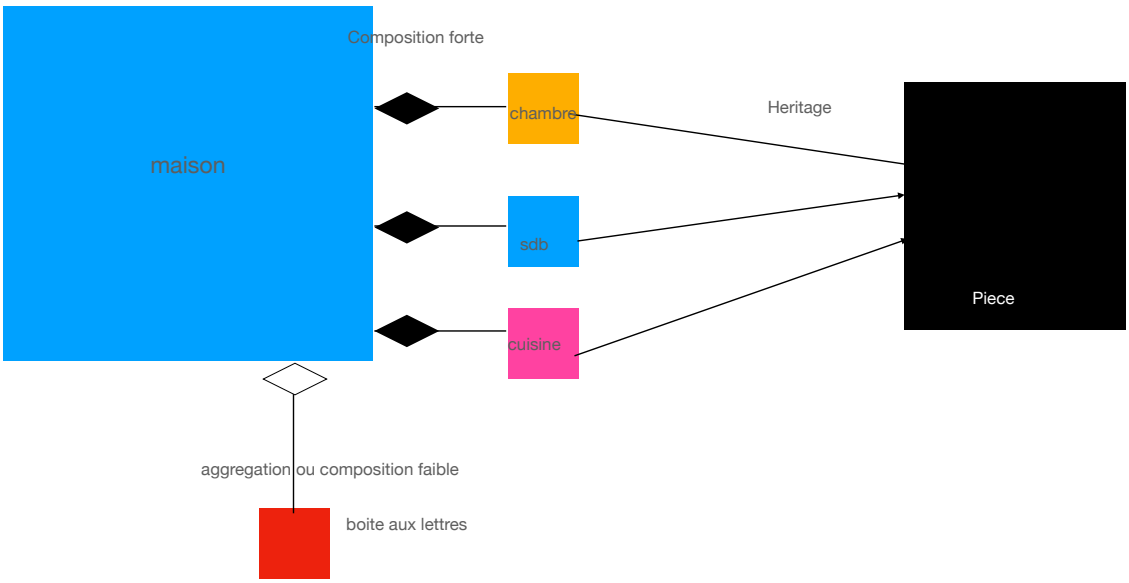
https://tutowebdesign.com/poo-php.php

polymorphisme par classe abstraite

Une classe abstraite est une classe qui ne peut pas être instanciée, (pas d'instanciation d'objets)
Une classe abstraite peut contenir des méthodes déjà implémentées.
Une classe abstraite peut contenir des méthodes non implémentées.
Une classe abstraite est héritable.
On peut construire une hiérarchie de classes abstraites.
Pour pouvoir construire un objet à partir d'une classe abstraite, il faut dériver une classe non abstraite en une classe implémentant toutes les méthodes non implémentées.



composition / aggregation (composition faible)



SYMFONY

Modele MVC

Le MVC permet d'effectuer la séparation des préoccupations afin de mieux organiser le code. Ça évite le code "spaghetti".

Modèle: Doctrine , Objet Request Manager qui fait la liaison entre les objets et la base de données.

Avec Doctrine on ne fait pas de requêtes SQL mais un constructeur de requêtes:

```
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
```

Vue: template (*ici moteur twig*)

Controleur: routage et exécution d'une action sur ce routage. (routage = ce que l'on saisi dans la barre d'adresse ou dans l'URL d'un appel Ajax.

exemples de routage par annotations

*/**

** @Route("/modifmovie/{id}", name="modifmovie")*

**/*

public function modifierMovie(Movie $movie=null, Request $request, EntityManagerInterface $em) {

...actions... exemple ci-dessous

```
return $this->render('auth/index.html.twig',[
      "form" => $form->createView()
  ]);
```

}

*/**

** @Route("/creationmovie", name="creationmovie")*

*/

depuis php8 les annotations

#[Route("/", name: "association")]

entities -> classe

attributs en mode private
getters et setters

Repository -> Objet Request Manager qui permet de "mapper" les objets (instances des entités) avec les tables de la base de données.

Dans les classes *src/repository* on retrouve la déclaration:

pour une entité Association

```
use App\Entity\Association;
```

Dans les classes *Repository* on retrouve la déclaration:

```
use App\Repository\AssociationRepository;
```

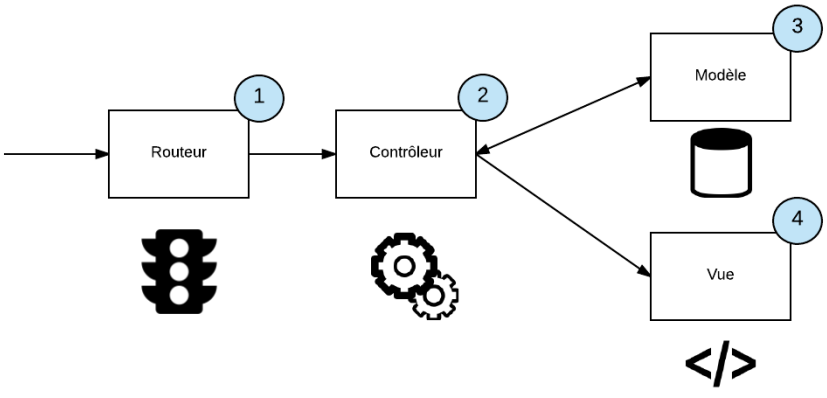
Symfony en mode CLI permet de générer rapidement

les entités
les relations entre ces entités (one-to-many , many-to-many, …etc…)
les composants d'accès à ces entités
la base de données
les formulaires de saisie.

Par contre les templates twig se font manuellement à partir d'une base.

principes MVC:

toutes les actions passent par le contrôleur, le modèle et le contrôleur n'affichent JAMAIS d'informations, le contrôleur ne modifie jamais directement la base de données.



node:

Basé sur les évènements !!!

Asynchrone

même langage JS côté client et côté serveur

rapide : asynchrone => **NODE n'attend pas.**

1 seul thread, pas de programmation concurentielle

grosse communauté de développeurs

grande quantité de modules (**npm**) , drivers pour toutes les BDD

Framework web **express** (http - routage - templating).

Plusieurs moteurs de template (Embedded Javascript - Mustache - handlebars … etc…)

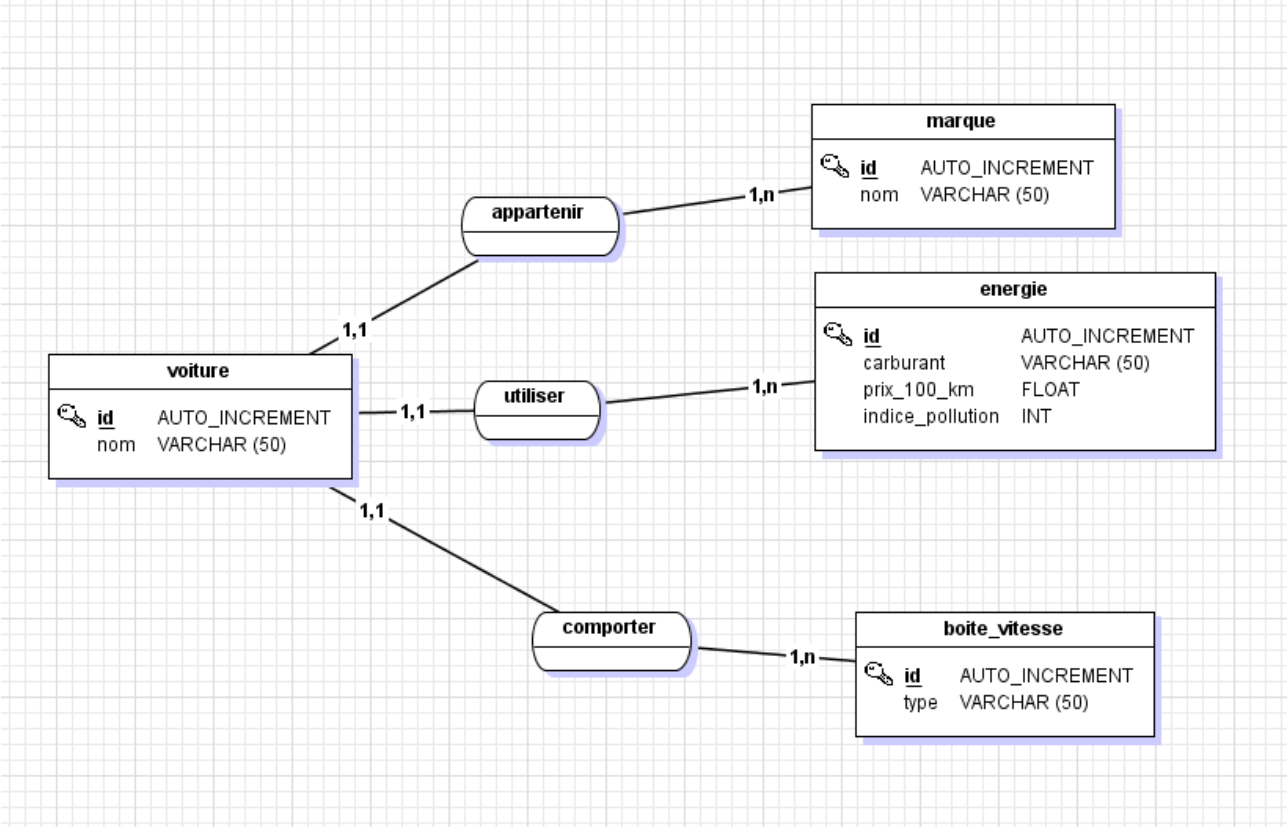
socket.io

connection permanente
push du serveur vers le client et inversement

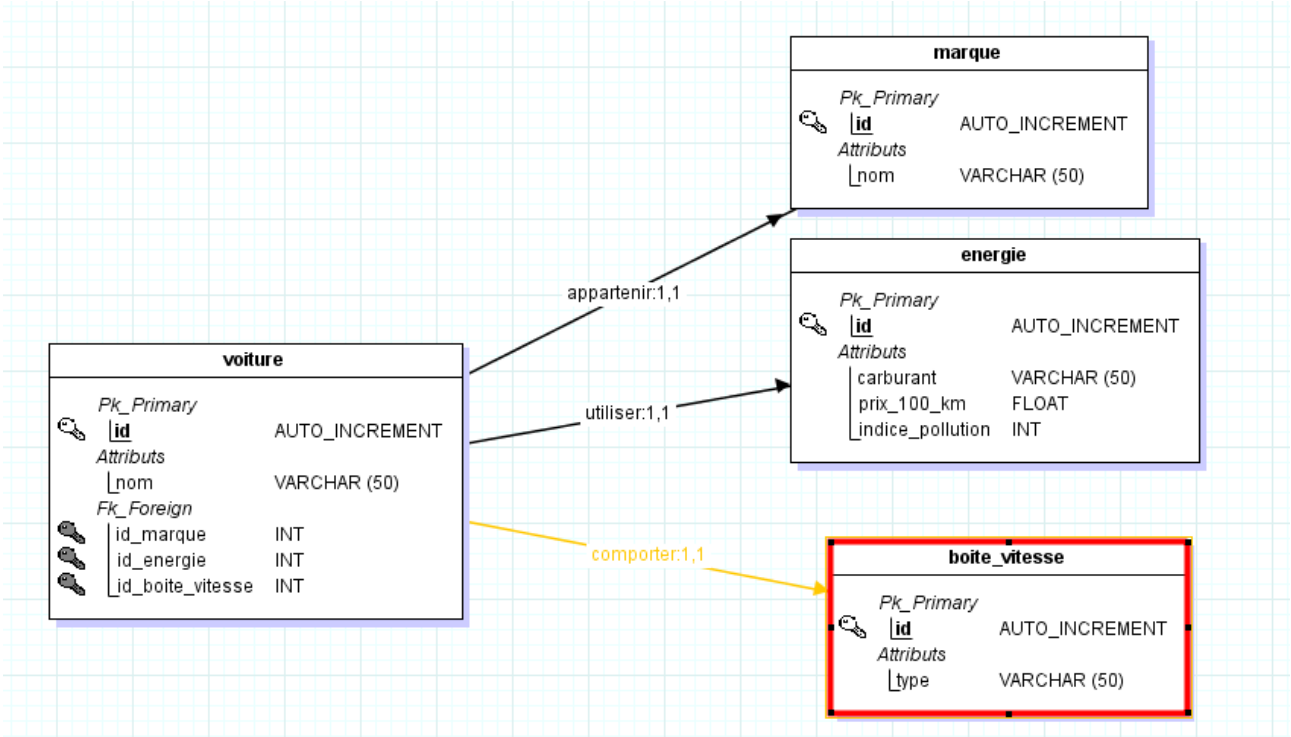
BDD Relationelle (Mysql/MariaDB)

avec JMerise ou https://www.looping-mcd.fr/

Relations entre les tables et cardinalité :



Modele Conceptuel de Données (Merise)
Modele Logique de Données
Modele physique de données -> SQL de création



Transactionnel PHP/PDO

```
try
{
    //on lance la transaction
    $pdo->beginTransaction();
    //nos 3 requêtes moches

    $pdo->query('SELECT * FROM machin WHERE bidule = 'truc'');
    $pdo->query('INSERT INTO machin SET bidule = 'truc', chose = 'moi');
    $pdo->query('UPDATE machin SET nombre = nombre + 1');

    // si jusque là tout se passe bien on valide la transaction
    $pdo->commit();
     echo "Tout s'est bien passé.";
}
catch(Exception $e) //en cas d'erreur
{
    //on annule la transition
    $pdo->rollback();
      // bug !!!!
      echo 'Erreur : '.$e->getMessage().'<br />';
      //on arrête l'exécution s'il y a du code après
      exit();
}
```

NOSql

Mongodb

Base de données orientée documents
Semi-structuré en json { } , pas de schéma
langage d'échange JSON -> machine and human readable
JSON proche de l'objet Javascript
JSON => { "nom":"marcel" }
Objet Javascript => { nom: "marcel' }
replicaset (tolérance aux pannes)
sharding (passage à l'échelle pour supporter la montée en charge, énormément de connections, une collection pourra se trouver sur plusieurs replicaSet)

possibilité d’utiliser les schéma avec **mongoose**.
facilite de creation de donnée et de recherche de collections

Autres BDD noSQL : couchDB , Cassandra, Redis (très rapide car en mémoire) , hadoop ((big data), elasticsearch.

Transactionnel mongodb

```
const session = client.startSession()
await session.withTransaction(async () => {
  await collection.insertOne(doc1, { session })
  await collection.insertOne(doc2, { session })
})
session.commitTransaction()
session.endSession()
```

Tests:

notions de Jest / phpunit
écrire des fonctions pour tester des fonctions

Authentification

cryptage du password dans la base de données
(php: password_hash / node: module bcrypt)
Sessions
JWT - Json Web Token

sécurité

pour le PHP :
Attaque XSS (ne pas laisser du javascript s'exécuter dans des champs input. htmlspecialchars)
injection SQL : les requêtes préparées (PDO prepare)
helmet

https (cryptage asymétrique)

déploiement

transfert de notre application sur un hébergement
ftp / sftp (filezilla, WinSCP …etc…) pour déposer votre application sur un serveur distant.
github
docker
cloud (heroku pour node par exemple / Atlas pour mongoDB)

MVC

séparation des préoccupations pour mieux organiser le code.

SEO (Référencement naturel)

Rajouter le titre dans ma liste de recommandations

RGPD (loi européenne)