Développement Web - Back-end

Antoine TIREL

Premier trimestre 2025

Table des matières

1 Définitions

1.1 Application web

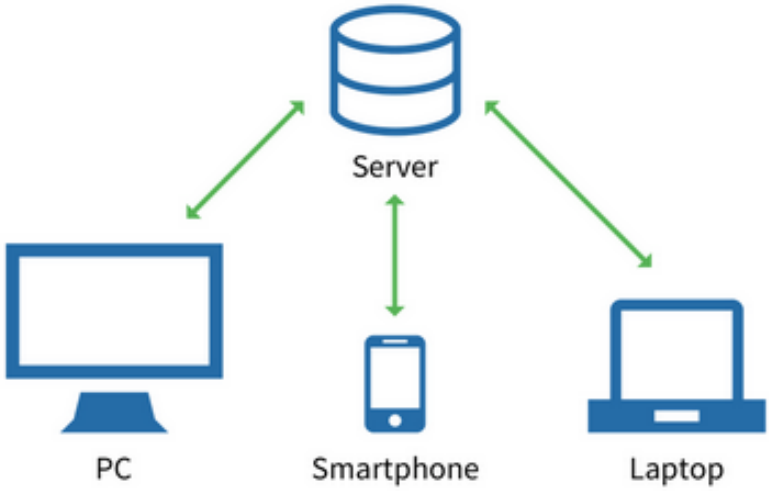
“En informatique, une application web est une application manipulable directement en ligne grâce à un navigateur web et qui ne nécessite donc pas d’installation sur les machines clientes, contrairement aux applications mobiles."
Wikipédia

“Une application Web est un logiciel qui s’exécute dans votre navigateur Web. Les entreprises doivent échanger des informations et fournir des services à distance. Elles utilisent des applications Web pour se connecter aux clients de manière pratique et sécurisée."
Amazon

“Une application web est un logiciel accessible et exécuté sur un site web via le navigateur de l’utilisatrice ou de l’utilisateur. Les entreprises utilisent des applications web pour fournir une large gamme de fonctionnalités dans leur navigateur à leur clientèle. Il n’est donc pas nécessaire de télécharger et d’installer des logiciels."
OVH

Une application web est donc un logiciel applicatif, accessible et traduisible par tous les navigateurs ~~sauf Internet Explorer~~, ne nécessitant aucune installation sur la machine client.
Une application web fonctionne traditionnellement sur le modèle client serveur, bien qu’il puisse exister une quantité de couches intermédiaires plus ou moins manipulables et visibles.

1.1.1 Une petite modélisation



1.2 Histoire

Le **Worl Wide Web** est un système hypertexte public fonctionnant sur Internet, inventé par le CERN en 1990 permettant d'afficher du texte mis en forme sur un navigateur écrit en Objective-C et pouvant être utilisé en C.

1.3 Technologies

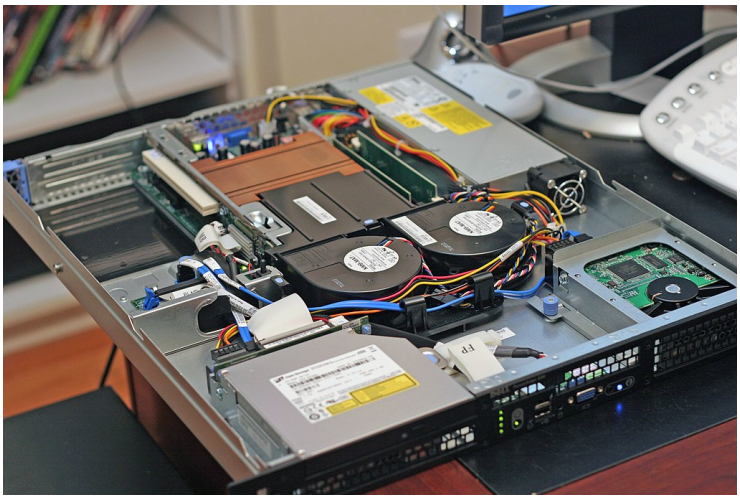
- **HTTP** ou Hyper Text Transfer Protocol pour les intimes
  - Permet de transférer des documents hypertextes
  - Aujourd'hui pratiquement disparue, remplacé par sa version sécurisée **https**
  - Utilise par défaut le port 80, là où le https utilise le port 443
  - http et https utilisent bien le même protocole, et le https est souvent nommé http over TLS pour préciser que le protocole ne change pas, mais qu'il dispose juste d'une surcouche d'encryption des données.
- **HTML** ou Hyper Text Markup Language
  - Permet d'écrire des document hypertexte
  - Ces documents peuvent inclure des images, du texte, du binaire
- **TCP/IP** Transfer Control Protocol/Internet Protocol
  - Protocole de "livraison" des données
  - Basé sur un principe de connexion double
  - Le serveur écoute en permanence, gère les demandes de connexion en les validant et en assurant le résultat
  - Peut dans certains contextes, être remplacé par **UDP**, particulièrement dans les cas où les pertes de données sont négligeables.
- **MIME** standard d'envoi de mail

1.4 Modèle Client Serveur

Le modèle Client Serveur consiste à découper une application en deux couches principales, chacune ayant sa responsabilité, les deux devant fonctionner conjointement pour fournir une application Web fonctionnelle.

1.4.1 Serveur Web

Un serveur web est une machine qui écoute, réceptionne et traite des requêtes. Il est, en général, composé d'un OS, d'un serveur d'écoute, d'une BDD.



[rod24]

1.4.2 Catégories de serveur

- **Statique**
  - Renvoie des données en lecture seule
  - Laisse la responsabilité des modifications du navigateur au client
  - Est, de fait, beaucoup plus sécurisé, les données du serveur ne pouvant pas être modifiées par le client
- **Dynamique**
  - Peut modifier les données
  - Permet de séparer les responsabilités et d'alléger le réseau

1.4.3 Client

Gère l'affichage ou la récupération des données, peut tout aussi bien être un navigateur, qu'une interface CLI qu'un outil dédié.

Exemples :

- **Navigateurs**
  - Chromium
  - Firefox
  - Google Chrome
  - Opera
  - Safari
- **Outils CLI**
  - curl
- **Outils d'envoi de requête**
  - Postman
  - Bruno
  - Wireshark du du dudu

1.4.4 Client VS Serveur

Avec tout ça, on voit émerger deux clans, appelés les fronteux et les backeux, si on excepte les fullstack et les Ops/Architectes.

Dans l'équipe du back

- Le côté caché d'une application
- Assure l'intégrité des données
- Gère la plupart des calculs
- Met à disposition du consommateur
- Est responsable du "pourquoi ça marche"
- Bosse sur la même techno depuis avant ma naissance

Dans l'équipe du front

- Le côté sexy de l'application
- Assure le suivi des contraintes de visuel
- Fait du CSS (et se soigne ensuite)
- Est un artiste abstrait
- Est responsable du "à quoi ça ressemble"
  - Est-ce que la mise en page garde l'œil ?
  - Est-ce qu'un lecteur d'écran peut parcourir la page ?
  - Pourquoi ça ne marche pas ?
- Doit suivre le dernier truc à la mode

Dans l'équipe des full-stack

— Fait du front et du back
— Est de fait meilleur, parce que plus facile à vendre
— Mais est en fait moins bon, parce que pas spécialiste

1.4.5 Où il est un peu question de déploiement

Un code développé sur une machine ne sert pas à grand chose en lui-même, à part pour briller en soirée mondaine. Pour assurer la mise à disposition des applications web, on fait appel aux Ops (IT ops pour les anciens) qui sont responsables du déploiement des applications.
Le déploiement consiste en la mise à jour d'un serveur afin qu'il embarque les dernières modifications réalisées et puisse répondre aux besoins du client final, celui qui paie.
Historiquement, le déploiement était réalisé par des équipes dédiées, ce qui menait souvent au problème du "ça marche sur ma machine".
Ce problème a été résolu avec l'apparition de nouveaux outils de déploiement plus proches du développement qui ont donné naissance aux DévOps, ayant à la fois les notions d'opérations et de développement.

1.5 Langage et frameworks

On l'a vu dans les précédentes sections, une application web ne permet que l'affichage de pages HTML. De fait, un serveur web ne fait qu'envoyer des pages HTML ou du contenu qui sera ensuite traduit pour devenir une page HTML.

Afin de simplifier le travail de développement, et de permettre une grande versatilité, un cadre a été ajouté sur la plupart des langages. Ce cadre (frame en anglais) est la base du concept de framework, qui peut se résumer en une bibliothèque permettant de réaliser l'intégralité des actions attendues par un serveur et un client.

2 Parlons back end

2.1 Pour la culture

2.1.1 PHP

Le langage PHP est un des tous premiers créés et mis à disposition pour la rédaction de pages sites et applications web.
De fait, il est l'un des plus vieux et des plus décriés aujourd'hui, et vous n'aurez aucun mal à en voir du mal partout.
Le langage est particulièrement réputé pour ses failles de sécurité historiques et sa relative non fiabilité.
Toutefois, il représente plus de 75% des sites internet en ligne, et dont 62% sont sur des versions connues pour leur vétusté et leurs failles de sécurité.
Aujourd'hui, le langage n'a pas beaucoup de popularité mais est tout de même maintenu à jour et suivi par une communauté active lui assurant la même pérennité que la plupart des autres langages utilisés pour faire du développement web.

2.1.2 Les dinosaures

Comme vu dans la partie historique du web, le world wide web date de 1990, et est donc sorti après plusieurs langages comme le C, le C++ ou en même temps que le Python.
Si le C et le C++ ont rapidement été dépassés et remplacés par le PHP, qui reprend beaucoup de points de leur syntaxe, Python a mis à disposition deux frameworks : un minimaliste Flask et un lourd Django.

2.1.3 Le monde magique de Microsoft

Dans le monde merveilleux de Bill Gates où tout le monde n'utilise que Windows et où les autres systèmes d'exploitation sont bannis, que tous les serveurs tournent sur Windows Server, un framework permettant de faire du développement web en C# (ou en F# pour les gens bien), un langage inspiré de Java, Dotnet.
Dotnet possède pour lui l'avantage d'être soutenu par Microsoft, ce qui lui assure une vie possiblement aussi longue que Windows, C# étant également utilisé dans la plupart des modules de Windows.

2.2 Java

À l'origine, Java est un langage orienté objet, un des premiers à complètement se détacher du C développé par Sun Microsystems.
Sa facilité d'accès, sa nouveauté, ainsi que la JVM lui assurent une grande popularité, et son apparition peu après le web permet également une prise de décision rapide avec la création de la plateforme Java et de ses nombreuses sous-divisions, principalement Java SE (Standard Edition) et Java EE (Entreprise Edition), deux services ayant accès à des bibliothèques Java, gérant les transaction, les serveurs web applicatifs, la concurrence.
Aujourd'hui, Java est le langage le plus répandu dans l'industrie en France pour du développement back-end, en particulier sur les nouveaux projets et les projets lourds ayant démarré après la grande histoire du PHP.

2.2.1 De la compilation et un peu d'histoire

Java est un langage compilé, dont la compilation produit un fichier .class, qui tourne dans la JVM, une machine virtuelle Java, ce qui permet au code d'être exécuté sur n'importe quel système d'exploitation.
Cette particularité de Java date de la version 1 sortie en 1996.
Historiquement, Java suivait un process de release lent, et ne sortait une nouvelle version que tous les deux ans en moyenne, chaque release étant propriétaire et fermée.
En 2006, Sun crée OpenJDK et ouvre le développement de Java en le rendant open-source, peu de temps avant la sortie de Java 6, ce qui amène de nombreuses personnes à collaborer à la création de Java 7, qui ne sortira que 5 ans plus tard.
Java 7 devient rapidement le langage le plus utilisé pour les projets web, et si vous rencontrez un projet de cette époque ou plus vieux, fuyez. Le langage est également adopté par Android pour le développement de ses applications, et, aujourd'hui encore, un grand nombre d'applications Android sont des fichiers class transformés.
En 2014 sort Java 8, la version immortelle, qui apporte de nombreuses fonctionnalités encore utilisées aujourd'hui et qui est la plus vieille compatible avec la totalité des bibliothèques qui seront utilisées dans ces cours et TD.
Cependant, par soucis de rétro-compatibilité, Java traîne une énorme partie de son historique, ce qui lui vaut de voir apparaître de plus en plus d'alternatives, qui reprennent toutefois sa base.

2.3 Kotlin

Le monde du web évoluant et le langage Java vieillissant, de nombreuses alternatives au langage apparaissent, principalement pour en simplifier l'écriture trop verbeuse.
Parmi ces alternatives, JetBrains décide de proposer Kotlin, un langage de programmation orienté objet dont la compilation produit des fichiers .class, lisibles par les jvm.

Sa facilité et son efficacité, avec notamment une bien meilleure gestion des exceptions que Java au moment de sa sortie conduit Android à privilégier le développement d'applications en Kotlin plutôt qu'en Java, et que Google place officiellement Kotlin comme langage par défaut pour Android Studio et le playstore.

3 Parlons Web

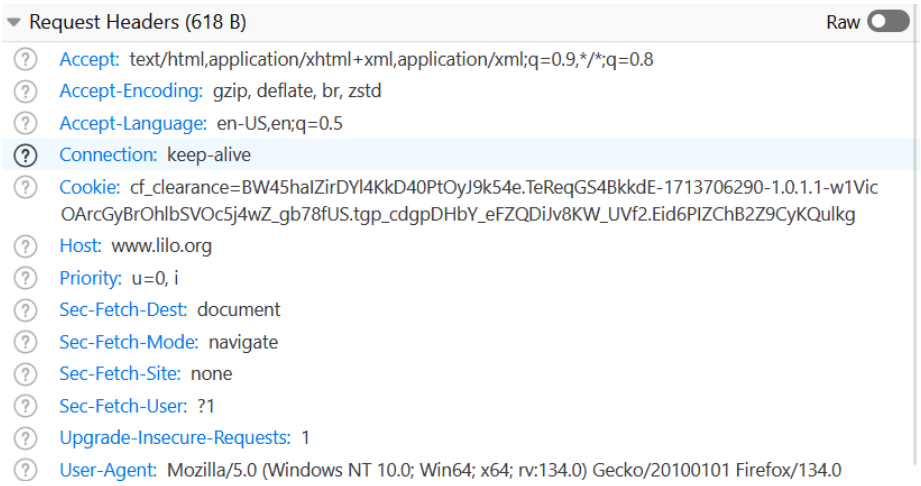
3.1 HTTP

HTTP est le protocole d'échange pour le Web, mais concrètement, comment ça marche ?
HTTP fonctionne selon le principe de requêtes pour communiquer À une URL dédiée sont associées des requêtes, définies par des verbes

- GET, HEAD -> pour récupérer des informations
- POST -> pour créer des informations
- PUT, PATCH -> pour modifier des informations
- DELETE -> pour supprimer de l'information

Une commande est accompagnée d'un contenu divisé en deux parties :

1. HEADERS (en-tête) Contient toujours ou presque l'authentification, ainsi que des informations d'usage, typiquement le navigateur, la cible
2. BODY (corps) contenu réclamé par l'utilisateur



Les commandes donnent lieu à différentes réponses, auxquelles sont associées un code :

- 1XX -> sache que
- 2XX -> succès
- 3XX -> va voir là-bas
- 4XX -> Erreur client
- 5XX -> Erreur serveur

Un HEADER et un BODY

3.2 Un peu de théorie

Soit f une application, on dit que f est idempotente lorsque f(f(a)) = f(a), ce qui ne veut pas pour autant dire que f(a) = a comme par exemple avec la fonction valeur absolue.
Par abus de langage, on dit qu'une requête HTTP est idempotente lorsque son application en série ne modifie pas davantage les données sur le serveur qu'une simple application.
Les requêtes HTTP GET, PUT, HEAD et DELETE sont toutes dites idempotentes : appliquée en série, elles doivent avoir le même effet indéfiniment.
Un exemple :
J'ai un serveur avec les données suivantes :
Liste des prix Nobel de Physique
Je décide de supprimer Albert Einstein de la liste :

DELETE /nobel/albert_einstein

Si j'exécute cette requête en boucle, Albert Einstein n'étant plus dans la liste, il n'est pas supprimé et ma liste contient bien la même chose qu'après mon premier appel.

3.3 URL

Source : MDN

Une URL est composée de plein de blocs détaillés ci-après

- Scheme -> schéma Protocole de transfert permettant d'accéder à la ressource
  - http
  - https
  - mailto
  - ftp
- Authority -> La concaténation du nom de domaine et du numéro de port Le numéro de port est optionnel dans le cas où on utilise les protocoles http ou https.
- Path ou chemin
- Paramètres -> une série de clé/valeurs séparés par une esperluette
- Ancre -> une sous-adresse dans un document

Avec un exemple :
https ://fr.wikipedia.org/wiki/Réseau_Polytech#Écoles

3.4 Il reste un peu de protocole, je vous en remet ?

3.4.1 REST

Une API REST, ou en bon françois académicien Interface de Programmation d'Application à Transfert d'État REprésentationnel.
On a donc un protocole assurant l'interopérabilité lors de la communication. Il n'y a pas d'état, les données sont dynamiques et on peut donc en mettre en cache.
Une API REST communique en format MIME (JSON, XML, HTML, etc.)
Un exemple (qui se trouve être par le plus grand des hasards le TD)

- GET /Pokemon/id
- POST /Pokemon
- PUT /Pokemon/id
- PATCH /Pokemon/id/availability
- DELETE /Pokemon/id

Avantages

- Performant
- Accessible
- Profite du HTTP
- Largement utilisé
- Indépendant du langage

Désavantages

- Une synchronisation entre le client et le serveur est nécessaire

— Risque de surcharge

3.4.2 SOAP

SOAP ou Simple Object Access Protocol
La logique tient ici dans le “simple”, on se limite à la transmission de messages, sous forme d’enveloppe, en envoyant du HTTP et du XML.

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
               xmlns:m="http://www.example.org">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice>
      <m:StockName>T</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Source : wikipedia

Avantages

— Extensible
— Profite du XML
— Indépendant du langage

Désavantages

— Le XML c’est verbeux
— ~~Mort~~

3.4.3 GraphQL

Le GraphQL, ce n'est pas du SQL, même s'il a le même Q et le même L.
On suit toujours la logique HTTP, mais uniquement avec des POST, et on effectue des queries et des mutations.

```
{
  hero {
    name
    friends {
      name
    }
  }
}
```

Query du client...

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        }
      ]
    }
  }
}
```

...réponse du serveur

Avantages

- Le client choisit ce qu'il requête
- Adapté aux architectures avec contraintes

Désavantages

- Le client DOIT choisir
- Pas de cache côté serveur
- n'utilise pas HTTP en entier

3.5 Langages d'échange

3.5.1 JSON

JavaScript Object Notation Le JSON est un format standard d'échange entre logiciels, il a l'avantage d'être lisible par un humain.
Le JSON est le format le plus répandu pour les échanges REST.

```json
{
    "glossary": {
        "title": "example glossary",
        "GlossDiv": {
            "title": "S",
            "GlossList": {
                "GlossEntry": {
                    "ID": "SGML",
                    "SortAs": "SGML",
                    "GlossTerm": "Standard Generalized Markup Language",
                    "Acronym": "SGML",
                    "Abbrev": "ISO 8879:1986",
                    "GlossDef": {
                        "para": "A meta-markup language, used to create markup languages such as DocBook.",
                        "GlossSeeAlso": ["GML", "XML"]
                    },
                    "GlossSee": "markup"
                }
            }
        }
    }
}
```

Le JSON est construit suivant le principe clé/valeur, à chaque clé est associé une valeur qui peut être soit une string, soit un nombre, soit un booléen, soit un tableau de tout ça.

3.5.2 XML

Extensible Markup Language, ou en bon français, langage de balise extensible. Super verbeux et plus présent pour des raisons historiques qu'autre chose.

```xml
<!DOCTYPE glossary PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
 <glossary><title>example glossary</title>
  <GlossDiv><title>S</title>
   <GlossList>
    <GlossEntry ID="SGML" SortAs="SGML">
     <GlossTerm>Standard Generalized Markup Language</GlossTerm>
     <Acronym>SGML</Acronym>
     <Abbrev>ISO 8879:1986</Abbrev>
     <GlossDef>
      <para>A meta-markup language, used to create markup
languages such as DocBook.</para>
      <GlossSeeAlso OtherTerm="GML">
      <GlossSeeAlso OtherTerm="XML">
     </GlossDef>
     <GlossSee OtherTerm="markup">
    </GlossEntry>
   </GlossList>
  </GlossDiv>
 </glossary>
```

3.5.3 YAML

Yet Another Markup Language
Le plus récent, le moins verbeux, le plus sexy, celui qu'on utilise pour nos dépendances en Kotlin

```yaml
glossary:
  title: example glossary
  GlossDiv:
    title: S
    GlossList:
      GlossEntry:
        GlossTerm: Standard Generalized Markup Language
        Acronym: SGML
        Abbrev: ISO 8879:1986
        GlossDef:
          para: |-
            A meta-markup language, used to create markup
            languages such as DocBook.
          GlossSeeAlso:
            GlossSeeAlso: ''
            GlossSee: ''
```

4 Interfaces de communication

4.1 Swagger

Swagger est un outil permettant d'exposer les points d'entrées (endpoints) des REST API d'un serveur.
Il permet, grâce à la spécification OpenAPI de décrire en JSON ou en YAML les retours des endpoints, de se baser sur la documentation, le type de données pour exposer les contrats d'interface et permet ainsi de se libérer visuellement du langage du serveur.

5 Architecture

Une application web, que ce soit côté back-end ou front-end, est séparée en plusieurs sous-parties afin de faciliter la lecture et éviter les fichiers de 5000 lignes (eh oui, de tels fichiers existent encore de nos jours, et ils sont moins rares que les ours polaires)
Afin de réaliser une application web utilisable et pouvant évoluer, il est nécessaire de suivre certaines règles, que l'on pourrait appeler de la logique, mais qui souffrent souvent d'interprétation.
Les classes sont regroupées en packages, chacune apportant son ensemble de logique, ce qui permet de construire un ensemble formant un bloc efficace.

5.1 SOLID

Dans un serveur web, il est important de respecter le principe de séparation des rôles ou SRP (single responsability principle).

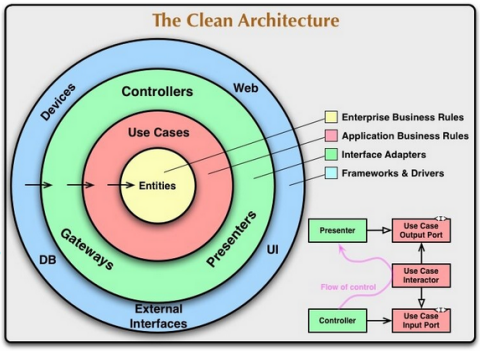
Il appartient à chaque classe d'avoir sa responsabilité, sa fonctionnalité dans l'application et sa représentation de données.
Open/Closed principle (le "O" SOLID)
Une entité (classe) doit être fermée à la modification mais ouverte à l'extension.
Une classe en programmation objet se définit par ses attributs et ses méthodes, tous les attributs doivent être modifié uniquement à travers des opérations prédéfinies, et ne doit pas être mis à jour.
Principe de substitution de Liskov
Toute instance d'une classe C doit pouvoir être substituée par une instance d'une classe D héritant de C
Interface Segregation principle
Tout client se connectant au serveur doit utiliser une interface dédiée à son usage, et aucun ne doit avoir à faire à une interface générale
Dependency inversion principle
Les entités doivent dépendre d'abstractions, pas d'implémentations.

5.2 Clean Architecture

Architecture proposée par Robert Martin
Respecte les principes SOLID

Séparation en couches

Composant de bas niveau ne dépend pas des composants de haut niveau



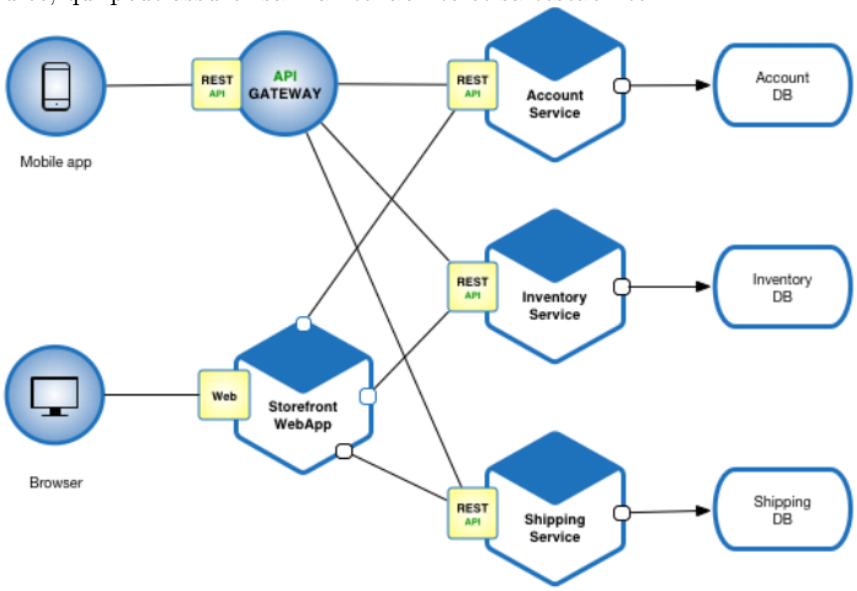
On découpe l'application en suivant les principes SOLID.
On manipule des entities, qui définissent de manière absolue les règles métier, les paramètres obligatoires et optionnels, les informations concrètes à sauvegarder dans l'application.
Dans ce découpage, on retrouve un bloc appelé "controllers/presenters/gateways" qui correspond à la partie en contact direct avec le web, donc la partie avec laquelle l'utilisateur interragit concrétement.
Cette architecture est une règle de bonne pratique, elle est applicable sur tous les projets, dans tous les langages (même le C (même le PHP j'vous jure)), elle est indépendante du framework, toutes les bibliothèques de composants sont construites autour de cette logique, toutes les aides en ligne sur Stackoverflow partent de ce principe, toutes les données de code d'entrainement de ChatGPT suivent ce principe.

5.3 MVP contre MVC

Les modèles MVP et MVC sont les modèles les plus courant d'architecture pouvant être trouvés au sein d'un projet.
Model View Controller
Le modèle définit les données et permet leur stockage, il s'agit du niveau le plus éloigné de l'utilisateur final, le client. Il suit une logique purement métier et n'est pas exposé du tout à aucun niveau de l'application.
La Vue (view) est la partie avec laquelle l'utilisateur final interagit, c'est la partie jolie de l'application, elle possède une partie de logique métier pour permettre l'échange avec le
Contrôleur (controller) qui est la partie gérant et définissant les interactions métier, ainsi que la validation des règles métier.
Model View Presenter
Le modèle MVP reprend la même logique modèle et vue, mais la vue voit ses responsabilités limitées à l'affichage, et perd toute logique métier, elle délègue toute cette partie au
Presenter, la présentation, qui envoie les données directement mises à jour pour l'affichage et ne nécessitant aucune forme de transformation métier, uniquement d'affichage.

5.4 Architecture en microservices

Comme le nom le laisse entendre, l'idée est de séparer les responsabilités entre différents services d'échelle réduite.
Concrètement : un microservice est un service indépendant dans son cycle de vie des autres services, non couplés à d'autres services. En clair, c'est un chemin qui répond à une et une seule problématique métier.
Ça permet notamment de gérer chaque problématique métier avec une équipe dédiée, qui peut assurer sa maintenabilité et sa testabilité



Source : What are microservices?

Sur l'image, chaque cube représente un microservice

Cette architecture présente aussi un avantage notable de réponse en cas de problème critique : si un microservice rencontre un problème critique qui empêche son fonctionnement, il n'impacte pas l'ensemble de l'application mais uniquement sa logique métier.

Avec un exemple :

Une banque présente un site internet avec quatre logiques métier :

- Se connecter à son compte pour le consulter.
- Réaliser des simulations d'emprunt.
- Ouvrir des nouveaux livrets d'épargne.
- Un site global permettant de naviguer entre les services et de choisir celui que l'on souhaite utiliser

Chacune de ces fonctionnalités est développée sur un microservice dédié.
Suite à une erreur de calcul, le service de simulation d'emprunt, qui n'a pas prévu que l'utilisateur pouvait rentrer des lettres crash à cause d'une exception non gérée qui fait déborder sa pile mémoire.
Le reste de l'application peut tout à fait continuer à fonctionner et n'est pas impacté par ce crash. Seule l'application principale doit s'adapter et signaler le dysfonctionnement du service de simulation d'emprunt.

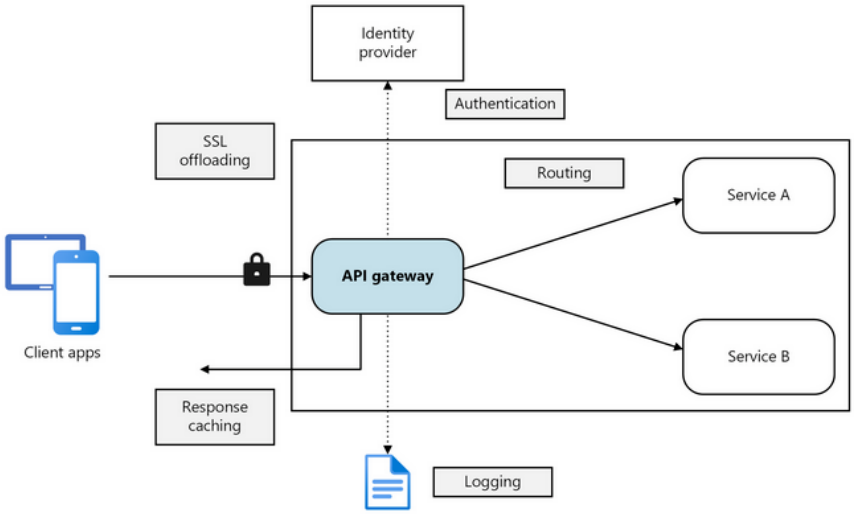
Le développement d'une architecture en microservices présente toutefois quelques désavantages :

- Le projet est plus complexe dans son ensemble
- Chaque service doit pouvoir communiquer avec les autres
- La charge sur le réseau est plus importante

Pour résumer : Une architecture en microservices est utile pour séparer les responsabilités à une échelle projet, elle permet notamment de répartir différents employés à travers différentes équipes, chacune travaillant sur son périmètre.
Elle comporte toutefois l'inconvénient d'alourdir le projet, et le réseau, et doit donc être utilisée lorsque ses avantages dépassent ses inconvénients.

5.5 API Gateway

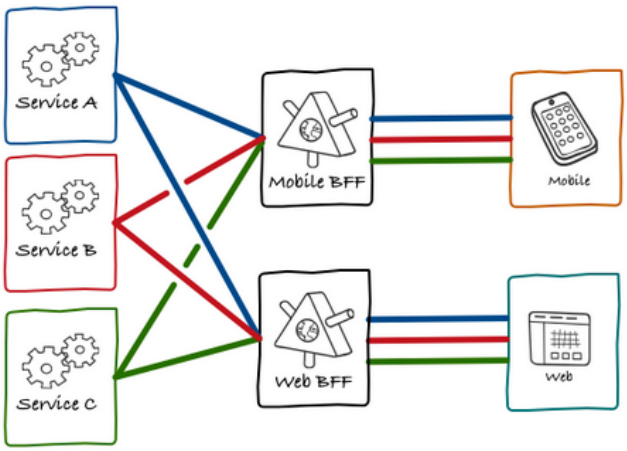
L'API Gateway est un moyen de diversifier les services en centralisant le point de contact pour le client en une "porte" d'entrée : la Gateway.



On voit rapidement le premier avantage, qui est la limitation de l'entrée à un seul point, la gateway. On peut également ajouter que l'on peut modifier les microservices ou leur paramètres sans impact visible pour le client.
Mais aussi une complexité encore augmentée qui augmente également le temps de réponse des requêtes.

5.5.1 Back For Front

Le Back For Front est un cas particulier d'API Gateway.
L'idée est ici de grouper les services vers une gateway qui sera dédiée à un client, ou à un type de client.
Un cas classique d'utilisation est celui d'un serveur statique répondant à différentes requêtes en fonction du support utilisé, un dédié au support mobile, l'autre au support tablette et enfin un dédié au support bureau.



6 Services

Dans une application web, en découpage classique, le gros du métier est géré par les services, qui contrôlent, calculent et construisent les réponses.
Suivant le SRP (single responsability principle) et ID (inversion de dépendance), chaque service doit gérer son propre objectif métier : un service pour l'authentification, un service pour gérer les utilisateurs, un service pour gérer les Pokémon, un service pour gérer les personnages...
Pour ce faire, on sépare les services de la construction de ses dépendances, ça permet de construire du code faiblement couplé et beaucoup plus maintenable.
On a ainsi un injecteur de dépendances, qui se charge de déterminer, à l'exécution, quel service injecter, quelle version injecter et quel bloc exécuter.

```
class MouseWithBall()
class WirelessKeyBoard()

class Computer(
    val keyboard: WirelessKeyBoard = WirelessKeyBoard(),
    val mouse: MouseWithBall = MouseWithBall()
)
```

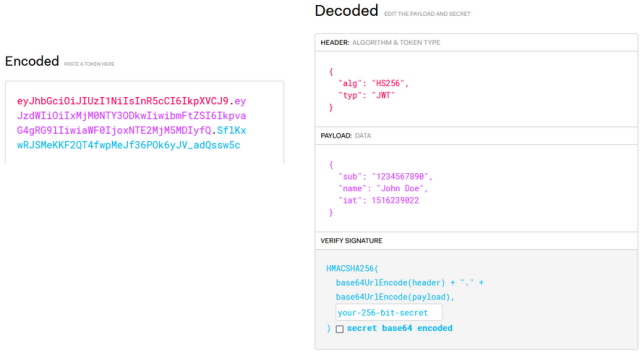
Sur cet exemple, on peut voir que Computer n'utilise que des souris à boule et des claviers sans fil.
Si demain on décide d'avoir un ordinateur uniquement compatible avec les souris infrarouges, on doit alors réécrire toute la classe Computer pour l'adapter.
On regroupe ensuite les services par catégories, en fonction de ce dont on a besoin.
On distingue ainsi les services dits **Singleton**, créés une et une seule fois et partagés à travers toute l'application. C'est le cas de base et le cas de la majorité des services rencontrés.
Les services dits **Scoped**, sont créés et manipulés par une instance d'exécution. En clair, dans une application Web, chaque client qui se connecte à mon serveur, ou chaque exécution parallèle du contrôleur a accès à sa version du service. Un tel cas est particulièrement utile si vos services doivent dépendre de différentes règles utilisateur.
Et enfin les services dits **Transient**, éphémères, où chaque exécution, chaque service, chaque contrôleur possède sa propre version du service.
Dans une application web utilisant Java ou Kotlin, on laisse une grande partie de la responsabilité de la gestion des services à...

6.1 Spring

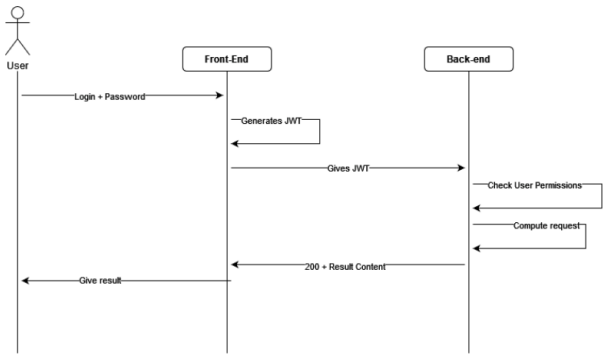
Spring est un cadriciel (framework) permettant de définir l'infrastructure d'une application Java (et Kotlin par extension), enrichi par SpringBoot, qui permet notamment de gérer toutes les interactions web.
Lors du TP 2, on a pu voir l'annotation @RestController.
Cette annotation indique à Spring que la classe courante est un contrôleur dans le modèle MVC, et qu'elle expose donc plusieurs points d'entrée, suivant le modèle REST. Les points d'entrée sont eux-mêmes annotés (@GetMapping, @PostMapping...) et permettent d'aiguiller les appels REST sur chaque méthode associée.
Pour produire une application, Spring réalise un amalgame (bundle) des différentes déclarations faites dans le projet et les fusionne afin de générer une application fonctionnelle, en suivant une Configuration, qui peut être modifiée grâce à l'usage de l'annotation @Configuration.
Un exemple qui se couple avec l'annotation @Profile, qui permet de définir un type d'environnement et ainsi d'avoir des fichiers annotés Configuration pour les profils de test, de build, de qualification, de production...
Une fois les points d'entrée ouverts, on passe sur des services, définis avec l'annotation @Service.
Cette annotation permet de marquer une classe comme étant un service, même si elle n'a qu'un usage purement pratique, étant elle-même un agrégat de plusieurs autres annotations.
Elle peut en revanche être complétée par l'annotation @Scope, qui comme son nom l'indique, permet de spécifier l'échelle à laquelle notre service sera actif. Cette annotation utilise par défaut la valeur singleton, mais peut par exemple prendre pour valeur un package, auquel cas le service est créé une et une seule fois pour ce contexte-là, mais est recréé dans tous les autres contextes.

6.1.1 Authentification

Vous l'aurez sûrement remarqué, mais on n'a pas beaucoup parlé de sécurité dans le cours, alors remédions-y.
Pour authentifier un utilisateur, un client se connectant au serveur, on utilise le principe de jetons décrits en json, **JWT** (ou Java Web Token)



Un JWT est généralement généré par le front-end (la partie client d'une application web) et utilisé par le back-end afin de déterminer qui est connecté et qui à le droit à quoi.



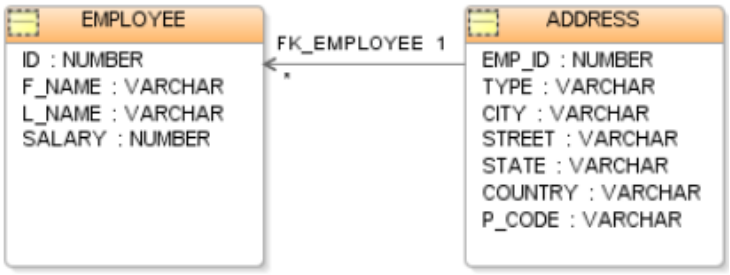
Dans nos projets et TP, toute la partie de traduction et de gestion d'envoi et réception des tokens est gérée par Spring, et ne nécessite aucun traitement supplémentaire.

7 Base de données

Afin d'assurer le stockage pur des données, on utilise des bases de données, que l'on va interroger à l'aide de requêtes.
Les bases de données servent essentiellement au stockage, pour se faire, elles sont intégralement optimisées pour le requêtage et le stockage de données.
En tant qu'unités de stockage, elles permettent de stocker des objets suivant le même principe que le json, à savoir, des fichiers contenant du texte, des nombres, ou du binaire.

7.1 Base de données relationnelles

Une base de données relationnelle stocke l'information sous la forme d'un tableau, où chaque colonne, nommée **table** peut posséder différents types de relations avec les autres tables de l'application.
Chaque table est définie par un nom qui permet au système de gestion de bases de données (SGBD) de repérer la colonne de l'information recherchée, et un attribut d'identifiant, qui permet de trouver la ligne.
L'identifiant de la table est appelé **Clé primaire** (primary key), il doit être typé et unique pour chaque élément stocké dans la table.
Une table peut aussi contenir une **Clé étrangère** (foreign key), à savoir une donnée dont la valeur correspond à la valeur de la clé primaire d'une autre donnée dans une autre table.



7.2 SQL

SQL ou Structured Query Language est le langage le plus courant pour réaliser des requêtes sur les SGBD relationnelles.
Pour ce faire, SQL utilise une syntaxe basée sur des mots-clés.

SELECT permet de récupérer des informations dans une table, spécifiée avec **FROM**, suivant des critères qui peuvent être énumérés avec **WHERE**. Le caractère ***** est utilisé comme joker pour indiquer tout.
Exemples :
SELECT * **FROM** Employees
Renvoie la liste de toutes les informations de tous les employés.
SELECT Nom, Age **FROM** Employees
Renvoie la liste des noms et des âges de tous les employés.
SELECT * **FROM** Employees **WHERE** age $>$ 50
Renvoie la liste de toutes les informations des employés âgés de plus de 50 ans.

INSERT INTO ...**VALUES** ...permet d'ajouter une ligne dans le tableau, c'est à dire une donnée dans une table, en spécifiant les valeurs que doivent prendre les champs.
Exemple : À partir de la table suivante :

ID	FIRST_NAME	LAST_NAME	SALARY
123	Emma	Carena	1000

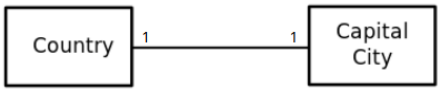
Si on réalise l'opération :
INSERT INTO Employees (ID, FIRST_NAME, LAST_NAME, SALARY) **VALUES** (456, 'Harry', 'Cover', 3000)
On obtient la table ci-après :

ID	FIRST_NAME	LAST_NAME	SALARY
123	Emma	Carena	1000
456	Harry	Cover	3000

7.2.1 Relations

Les bases de données relationnelles se définissent par différents types de relations entre les tables, explicitées par les clés primaires et étrangères.

One to One




```sql
CREATE TABLE Country (
    Id INT PRIMARY KEY NOT NULL,
    Name VARCHAR(255),
    CapitalId INT FOREIGN KEY REFERENCES CapitalCity(Id)
)
```

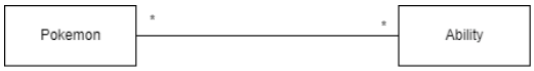
```sql
CREATE TABLE CapitalCity (
    Id INT PRIMARY KEY NOT NULL,
    Name VARCHAR(255),
    InhabitantNumber INT,
    CountryId INT FOREIGN KEY REFERENCES Country(Id)
)
```

One to Many


```sql
CREATE TABLE Blog (
    Id INT PRIMARY KEY NOT NULL,
    Name VARCHAR(255)
)
```

```sql
CREATE TABLE Post (
    Id INT PRIMARY KEY NOT NULL,
    Name VARCHAR(255),
    Content VARCHAR,
    BlogId INT FOREIGN KEY REFERENCES Blog(Id)
)
```

Many to Many




```sql
CREATE TABLE Pokemon (
    Id INT PRIMARY KEY NOT NULL,
    Name VARCHAR(255)
)
```

```sql
CREATE TABLE Ability (
    Id INT PRIMARY KEY NOT NULL,
    Name VARCHAR(255)
)
```

```sql
CREATE TABLE PokemonAbility (
    PokemonId INT FORIEGN KEY REFERENCES Pokemon(Id),
    AbilityId INT FORIEGN KEY REFERENCES Ability(Id)
)
```

7.3 Object Relational Mapper

Un Object Relational Mapper, ou ORM est une interface mise à disposition par une bibliothèque permettant de simplifier grandement les conversions entre les données disponibles dans le serveur et celles stockées en base de données.
Cette approche permet de réduire drastiquement la quantité de code écrite, d'augmenter l'homogénéité du code du serveur (pas besoin d'écrire de SQL explicite) et limite les risques d'erreurs lors des échanges réseau, tout en prenant en charge les problématiques de sécurité et de concurrence.
Elle possède en revanche le désavantage d'ajouter une couche supplémentaire, qui apporte son lot de complexité d'exécution, induit une dépendance à l'ORM et nécessite de garder une connaissance au moins en surface des données concrètement manipulées.

7.3.1 JPA

La Java/Jakarta Persistance Application Programming Interface est l'interface historique mise en place par Java depuis Java 5 pour gérer les données relationnelles dans les serveurs.
Elle se présente suivant la même logique que Spring avec des annotations permettant de donner des rôles aux classes et aux méthodes.

```
@Entity  new *
@Table(name = "pokemon")
data class Pokemon(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    val id: Long? = null,
    val name: String,
    val description: String,
    val generation: Int,
    val type: Type
)
```

8 Bonnes pratiques

La section qui suit ne fait pas partie du cours au sens propre, elle n'est présente qu'à titre informationnel, "pour la culture", et pourrait faire l'objet de question bonus dans l'examen.

8.1 Tester c'est douter

Pour chaque TD, à l'exception du code réalisé sur la plateforme de Kotlin, le code n'a été testé que manuellement, c'est à dire en le faisant tourner et sans jamais vérifier que chaque fonction réalisait vraiment l'objectif souhaité.
Dans une application classique, on réalise toujours des tests unitaires, à savoir des tests qui isolent la méthode du reste de l'application et vérifient son bon fonctionnement.
Dans les environnements Java, on utilise généralement JUnit que vous devez trouver sur absolument tous les projets Java, Kotlin ou tout autre langage produisant des fichiers .class. Cette bibliothèque doit également toujours être couplée avec Jacoco (Java Code Coverage) qui permet de limiter les tests inutiles et de réaliser quelles parties du code ne sont couvertes par aucun test.

8.1.1 Muter c'est confirmer

Je conseille fortement de doubler JUnit avec Pitest, une bibliothèque de tests de mutation. Un test de mutation est un test qui consiste à modifier le code et à vérifier qu'au moins un test plante. Les tests de mutation permettent ainsi d'assurer une couverture bien plus fiable et de réduire encore davantage les tests inutiles. Cependant, ces tests réalisant de nombreuses modifications du code, et nécessitant une recompilation du code avant chaque exécution sont très coûteux, et leur exécution est donc à limiter, par exemple, avant une phase de qualification, ou suivant un rythme prédéfini et régulier.

8.2 Sonar

Tous les projets modernes dépendent d'au moins une bibliothèque externe, à moins de n'avoir tout réécrit eux-même.
Afin de garantir la qualité générale du code, la validité des bibliothèques susmentionnées, ainsi que les risques que pourraient comporter certaines exécutions de méthodes, on utilise des outils d'analyse de code comme Sonar.

8.3 Lombok

Dans un monde où tout le monde n'utilise pas Kotlin, il peut être intéressant de savoir qu'il existe des bibliothèques permettant de faire "comme Kotlin" avec du pur Java.
La bibliothèque Lombok permet en grande partie de compenser les manques de Java, en ajoutant notamment les annotations @Data, permettant d'avoir des

data class, @Builder, et plus particulièrement @Builder(toBuilder=true) qui permet de dupliquer une instance de classe :
MaClasse monInstance1 = new MaClasse(1, "toto", "tyty") ;
MaClasse monInstance2 = monInstance.toBuilder().param2("tutu").build() ;
Est équivalent Java du code Kotlin suivant :
val monInstance1 : MaClasse = MaClasse(1, "toto", "tyty")
val monInstance2 : MaClasse = monInstance1.copy(param2 = "tutu")
Lombok peut également être utilisé dans les projets Kotlin sur les classes n'étant pas des data class.

Références

[rod24] RODZILLA. “Serveur démonté”. In : *Wikipédia* (2024).