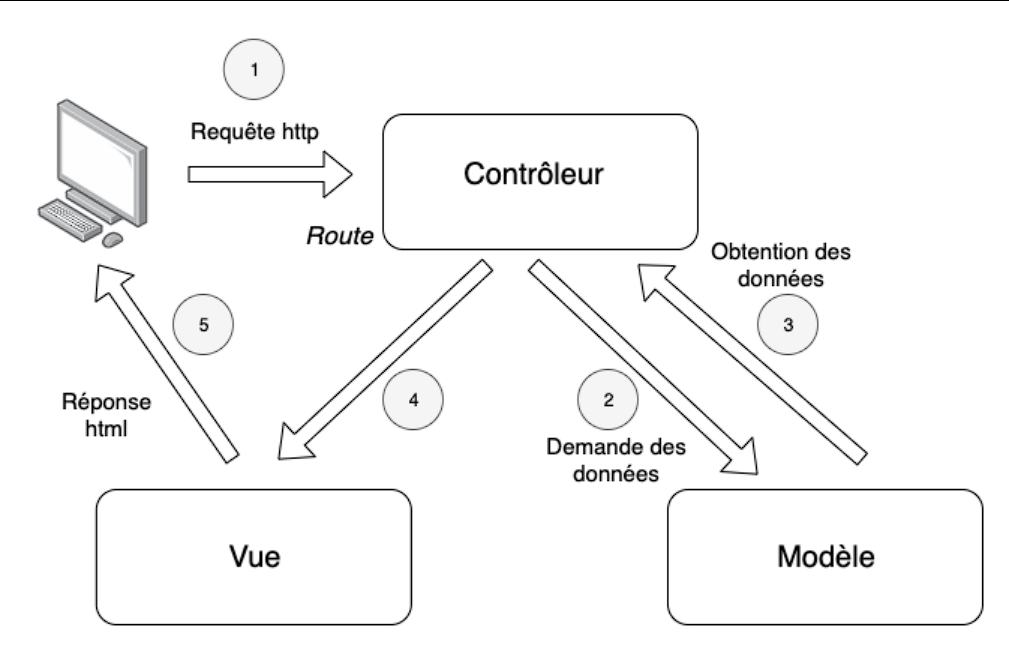
Fonctionnement des requêtes d’un serveur

Schéma du fonctionnement classique MVC (Modèle / vue / controleur)



Le MVT représente une **architecture** orientée autour de trois pôles : le **modèle**, la **vue** et le **template**. Elle s'inspire de l'[architecture MVC](https://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-contr%C3%B4leur), très répandue dans les frameworks web. Son objectif est de séparer les responsabilités de chaque pôle afin que chacun se concentre sur ses tâches.

**Modèle**

Le modèle interagit avec la base de données. Sa mission est de chercher dans une base de donnée les items correspondant à une requête et de renvoyer une réponse facilement exploitable par le programme.

Les modèles s'appuient sur un ORM (*Object Relational Mapping*, ou Mapping objet-relationnel en français). A quoi cela sert-il ?

La consultation d'une base de données relationnelle se réalise par un langage appelé SQL (*Structured Query Language*). Sa syntaxe est très différente de Python ! Par exemple, la requête suivante renverra tous les items de la table content :

SELECT \* FROM content

Ceux-ci seront dans un format que vous devrez alors interpréter manuellement avant de pouvoir l'utiliser dans votre programme. Il s'agit d'une étape assez fastidieuse qui n'est pas nécessaire.

Un **ORM traduit les résultats d'une requête SQL en objets Python** avec lesquels vous pouvez interagir. De même, il permet d'écrire une requête SQL en Python. Un peu comme un traducteur automatique !

Par exemple, la requête suivante renverra tous les items de la tablecontent : Content.objects.all()

Dans un projet Django, chaque application contient un document models.py qui réunit les différents modèles utilisés.

Nous verrons dans le chapitre suivant quels seront les modèles de notre programme. Mais vous pouvez d'ores et déjà y réfléchir ! 😉

**Template**

Un template est un fichier HTML qui peut recevoir des objets Python et qui est lié à une vue (nous y reviendrons). Il est placé dans le dossier templates.

Concrètement, un template peut interpréter des variables et les afficher. Par exemple, nous pouvons "donner" la variable tom="Tom" au template index.html et ce dernier l'affichera à la place du prénom.

Les *templates* sont très communs dans le développement web mais également dans d'autres disciplines. Avez-vous déjà utilisé des thèmes dans Power Point ou Wordpress ? Ou bien les gabarits dans InDesign ? La logique est exactement la même. D'ailleurs, la traduction française du mot "template" est "gabarit", même si le premier terme est plus utilisé dans le langage courant. C'est pourquoi j'utiliserai les deux sans distinction.

**Vue**

La vue joue un rôle central dans un projet structuré en MVT : sa responsabilité est de recevoir une requête HTTP et d'y répondre de manière intelligible par le navigateur.

Concrètement, une vue reçoit une requête HTTP qui peut ressembler à celle-ci :

HEAD /search?query=vert+trèfles+velours HTTP/1.1

Host: chapelier-fou.com

User-Agent: curl/7.54.0

Accept: \*/\*

Elle renvoie une réponse HTTP semblable à la suivante :

HTTP/1.1 200 OK

Date: Tue, 01 Aug 2017 14:36:27 GMT

Server: WSGIServer/0.2 CPython/3.6.2

Content-Type: text/html

X-Frame-Options: SAMEORIGIN

Content-Length: 11702

<!DOCTYPE html>

<html lang="fr">

...

La vue réalise également toutes les actions nécessaires pour répondre à la requête :

* si une interaction avec la base de données est requise, la vue appelle un modèle et récupère les objets renvoyés par ce dernier.
* si un gabarit est nécessaire, la vue l'appelle.

Dans un projet Django, les vues de chaque application sont regroupées dans le document views.py.

Chaque vue est associée à une url. Par exemple, l'url de la requête HTTP précédente est la suivante :

http://chapelier-fou.com/search?query=vert+trèfles+velours

Les urls d'un projet sont regroupées dans le fichier urls.py.

Une **vue** est une méthode qui génère le contenu à renvoyer à une requête. Il s'agit d'une méthode qui renvoie une réponse à une requête HTTP.

On appelle **route** un schéma d'url correspondant à une vue. A savoir également qu'une vue peut répondre à plusieurs URL.

**Exemple de la vie, la vraie**

Afin de bien comprendre ce concept, très répandu en informatique, imaginons une situation de la vie réelle.

Michel se rend chez ConsomPlus, un centre commercial immense. Il veut s'acheter un nouveau chapeau vert avec des trèfles pour fêter la Saint-Patrick. Mais pas n'importe lequel ! Il souhaite en acquérir un de bonne fabrique qui vivra vieux.

Le centre commercial compte des centaines de boutiques. Afin de trouver son chemin, Michel se rend à l'accueil.

* Michel : "*Bonjour, je cherche Le Chapelier Fou, un magasin de chapeaux. Pourriez-vous m'indiquer son emplacement ?*"
* La personne de l'accueil : "*Bien sûr ! Le Chapelier Fou se situe au troisième étage, allée de droite, au fond à gauche. Précisément, c'est à l'emplacement 14.*"
* Michel : "*Merci bien !*"

Il se dirige vers l'emplacement indiqué. Michel s'adresse alors au chapelier et lui expose sa requête :

* Michel : "*Bonjour monsieur le Chapelier ! Je cherche un chapeau haut de forme vert avec des trèfles, en velours et assez solide pour supporter quelques années en ma compagnie.*"
* Le chapelier : "*Aucun problème, mon bon Monsieur. Je vais voir ce que j'ai en stock !*"

Ce dernier se rend alors dans l'arrière-boutique à la recherche d'un modèle correspondant aux demandes. Dans l'armoire "Chapeaux haut de forme", à l'étagère "vert", il choisi celui qui porte les étiquettes "velours, trèfle" tout en grommelant "*Tous mes chapeaux survivraient à une guerre nucléaire. Evidemment qu'ils pourront supporter sa compagnie !*"

Il retourne auprès de son client et lui tend le modèle demandé. Michel est ravi : "*C'est parfait ! Exactement ce que je cherchais.*" Le Chapelier emballe alors le chapeau tant désiré dans une immense boite à chapeaux puis, moyennant paiement, tend l'ensemble à Michel.

Ce dernier ressort, heureux, [tout simplement](https://youtu.be/GC5E8ie2pdM?t=1m3s).

**Exemple dans Django**

Et si je vous disais que Django adopte un processus similaire ?

Imaginons à présent que Michel affronte une crise de *flegmatite* aiguë. Au lieu de se déplacer au centre commercial, il se rend sur le site "www.chapelier-fou.com".

Sur la page d'accueil, il voit un champ de recherche. Il entre "vert trèfles velours" et appuie sur entrée.

La page de résultat affiche le chapeau de ses rêves. Il le paye puis se rendra chez le chapelier pour le récupérer (il aime bien discuter avec ce chapelier fou).

Analysons la requête effectuée sur le site :

* Michel appuie sur "entrée". **Cela génère une requête HTTP**.
* la requête arrive sur le serveur. C'est à Django de jouer ! Il cherche dans le fichier urls.py **le schéma correspondant** à l'url. Dans le monde physique, cela correspond au moment auquel Michel s'est présenté à l'accueil.
* Chaque schéma est associé à une **vue**. La requête passe alors dans le fichier *views.py*, plus précisément dans la méthode associée au schéma. Dans le monde réel, cela correspond à l'arrivée de Michel dans la boutique et à son échange avec le chapelier.
* La vue exécute ce qui est demandé. Elle interroge le **modèle** Hat à la recherche d'un chapeau correspondant aux caractéristiques demandées. Cette étape correspond au moment auquel le chapelier se rend dans l'arrière-boutique à la recherche du chapeau demandé.
* La vue continue d'exécuter les instructions. Elle fait appel au **gabarit** adéquat pour formuler sa réponse. Cela correspond au moment auquel le chapelier met le chapeau dans la boite.
* Enfin, la vue renvoie une**réponse HTTP** correcte contenant le code HTML.

Le protocole HTTP est très intéressant ! Avant de continuer, je vous incite fortement à lire la partie [**Une architecture basée sur un protocole que l'on connaît bien : HTTP**](https://openclassrooms.com/courses/construisez-une-api-rest-avec-symfony/une-architecture-pas-un-protocole#/id/r-4280629) du cours de Sarah Khalil.

Bien, j'ai beaucoup parlé et peu codé dans ce chapitre ! Il est temps de remédier à cela et d'appliquer le MVT dans notre projet. Nous voyons cela dans le chapitre suivant !