Architecture RESTful

https://github.com/heig-vd-devprodmed-course/heig-vd-devprodmed-course

Visualiser le contenu complet sur GitHub à cette adresse.

L. Delafontaine, avec l'aide de GitHub Copilot.

Ce travail est sous licence CC BY-SA 4.0.

Plus de détails sur GitHub

Cette présentation est un résumé du contenu complet disponible sur GitHub.

Pour plus de détails, consulter le contenu complet sur GitHub ou en cliquant sur l'en-tête de ce document.

Objectifs (1)

- Décrire les principes fondamentaux d'une architecture REST.
- Différencier les architectures REST et RESTful.
- Décrire quand et pourquoi utiliser une architecture RESTful pour développer des services web.

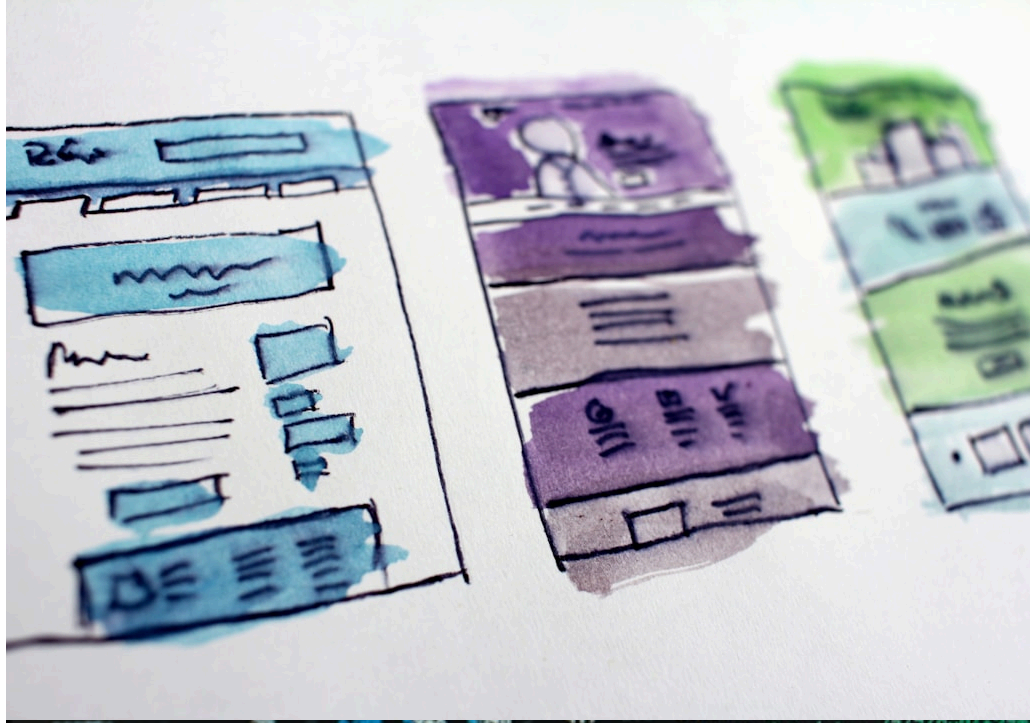
Objectifs (2)

- Mettre en œuvre une architecture RESTful dans une application web avec le framework Laravel.

Il s'agit du dernier contenu du cours !

MVC, REST et RESTful

- Jusqu'à présent, nous avons étudié l'architecture MVC (Model-View-Controller).
- Il existe d'autres architectures pour développer des applications web, notamment :
  - REST.
  - RESTful.



Jusqu'à présent… architecture MVC

- Architecture MVC : routes, contrôleurs, vues, modèles, authentification.
- Adaptée pour des applications web classiques avec interface graphique.
- Retourne du HTML complet, gère les sessions via cookies ⇒ votre navigateur peut comprendre et afficher les réponses.
- Pas adaptée pour des clients qui ne peuvent pas gérer de l'HTML et gérer les sessions (applications mobiles, scripts en ligne de commande).

Dans le futur peut-être… architecture REST (1)

Une application **REST** respecte les principes suivants :

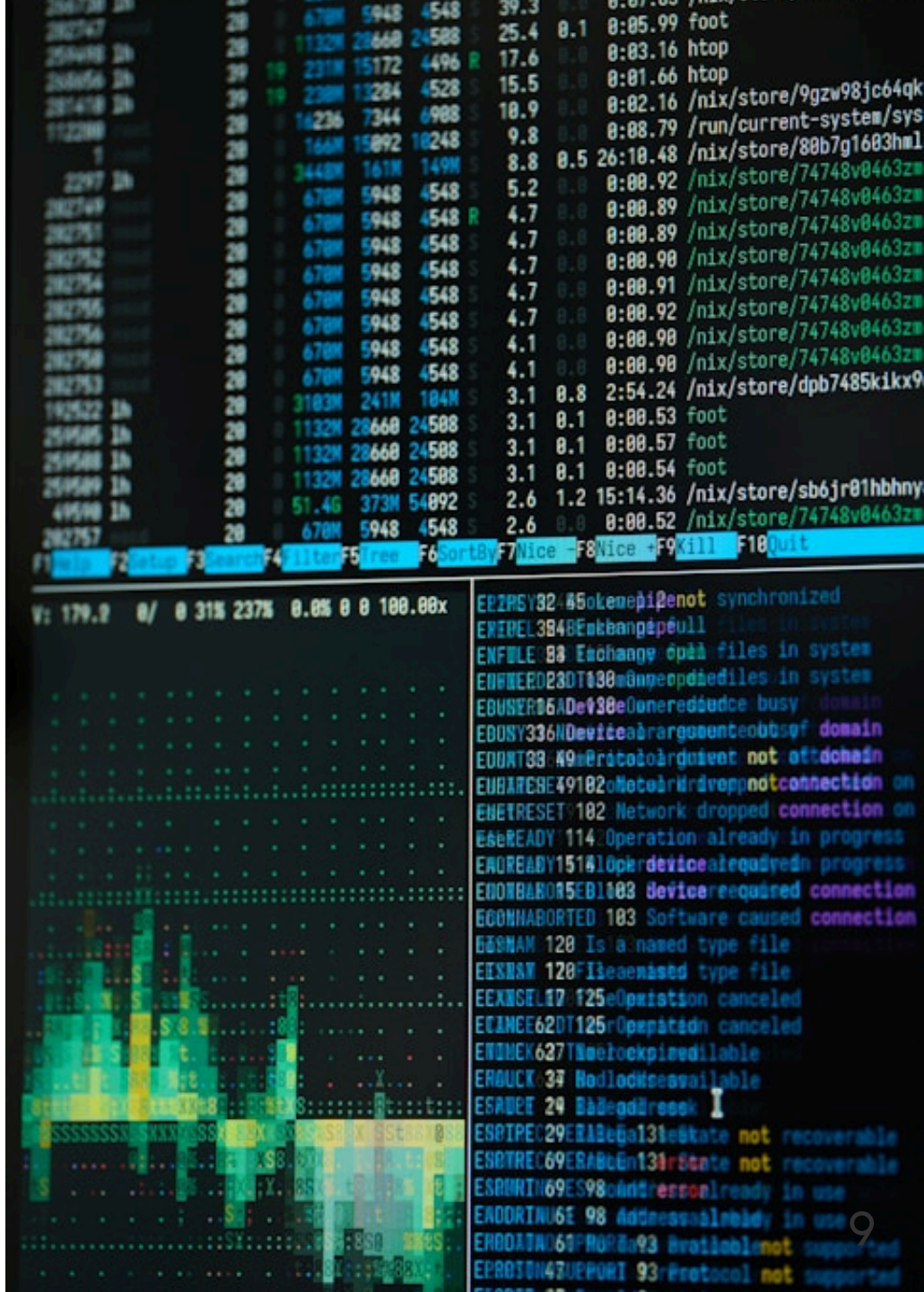
1. **Client-serveur** : un client consomme les services d'un serveur.
2. **Sans état** : chaque requête contient toutes les informations nécessaires pour authentifier la personne. Le serveur ne stocke aucune information de session.
3. **Cacheable** : les réponses peuvent être mises en cache.

Dans le futur peut-être… architecture REST (2)

4. **Interface uniforme** : l'interface entre client et serveur est standardisée (= qui suit les standards : URL, méthodes HTTP, données JSON, …).
5. **Système en couches** : plusieurs couches (base de données, logique métier, authentification, etc.).
6. **Code à la demande** (optionnel) : le serveur peut envoyer du code exécutable au client.

Architectures REST/RESTful

- Respecter parfaitement REST peut être difficile, voire impossible.
- On parle donc d'**architecture RESTful** : au plus proche des principes REST, mais certains aspects peuvent être adaptés.
- Laravel permet de développer des applications RESTful via une API.



API

- Une **API** (*"application programming interface"*) est un ensemble de règles permettant à différentes applications de communiquer.
- Elle fournit des fonctionnalités ou des données de manière standardisée.
- Utilisée par des applications mobiles ou des scripts en ligne de commande qui ne peuvent pas comprendre/afficher de HTML.
- Sépare le front-end du back-end.
- **Votre application peut être utilisée par différents types de clients (web, mobile, scripts) via la même API (universel !).**

Format des données

- Grâce au protocole HTTP, les clients peuvent définir le format des données qu'ils souhaitent recevoir du serveur en utilisant les en-têtes de la requête.
- Par exemple, un client peut demander des données au format JSON en incluant l'en-tête `Accept: application/json` dans sa requête HTTP.
- Si le serveur peut fournir les données dans le format demandé, il les renverra avec l'en-tête `Content-Type: application/json`.
- Le format JSON est largement utilisé pour les API RESTful.

Structure d'une API RESTful (1)

Conventions d'une API RESTful :

- Les ressources sont identifiées par des **URL uniques**.
- Les opérations utilisent les **méthodes HTTP** standard.
- Les réponses sont en **JSON**.
- L'authentification utilise des **tokens** (JWT, tokens d'API, etc.).
- Les erreurs utilisent des **codes d'état HTTP** appropriés.

Structure d'une API RESTful (2)

Exemple pour une ressource `posts` :

Méthode	URL	Description	Réponse
`GET`	`/api/posts`	Liste des posts	200
`POST`	`/api/posts`	Créer un post	201, 400
`GET`	`/api/posts/{id}`	Détails d'un post	200, 404
`PUT/PATCH`	`/api/posts/{id}`	Modifier un post	200, 404
`DELETE`	`/api/posts/{id}`	Supprimer un post	204, 404

Versionner une API RESTful

- Il est recommandé de **versionner** une API RESTful.
- Permet aux clients de continuer à utiliser une version stable pendant le développement de nouvelles fonctionnalités.
- Convention : inclure la version dans l'URL.

```
# Version 1
GET /api/v1/posts

# Version 2
GET /api/v2/posts
```

Tester une API RESTful

Pas d'interface graphique → outils spécifiques nécessaires :

- Bruno (recommandé).
- curl.
- Insomnia.
- Postman.

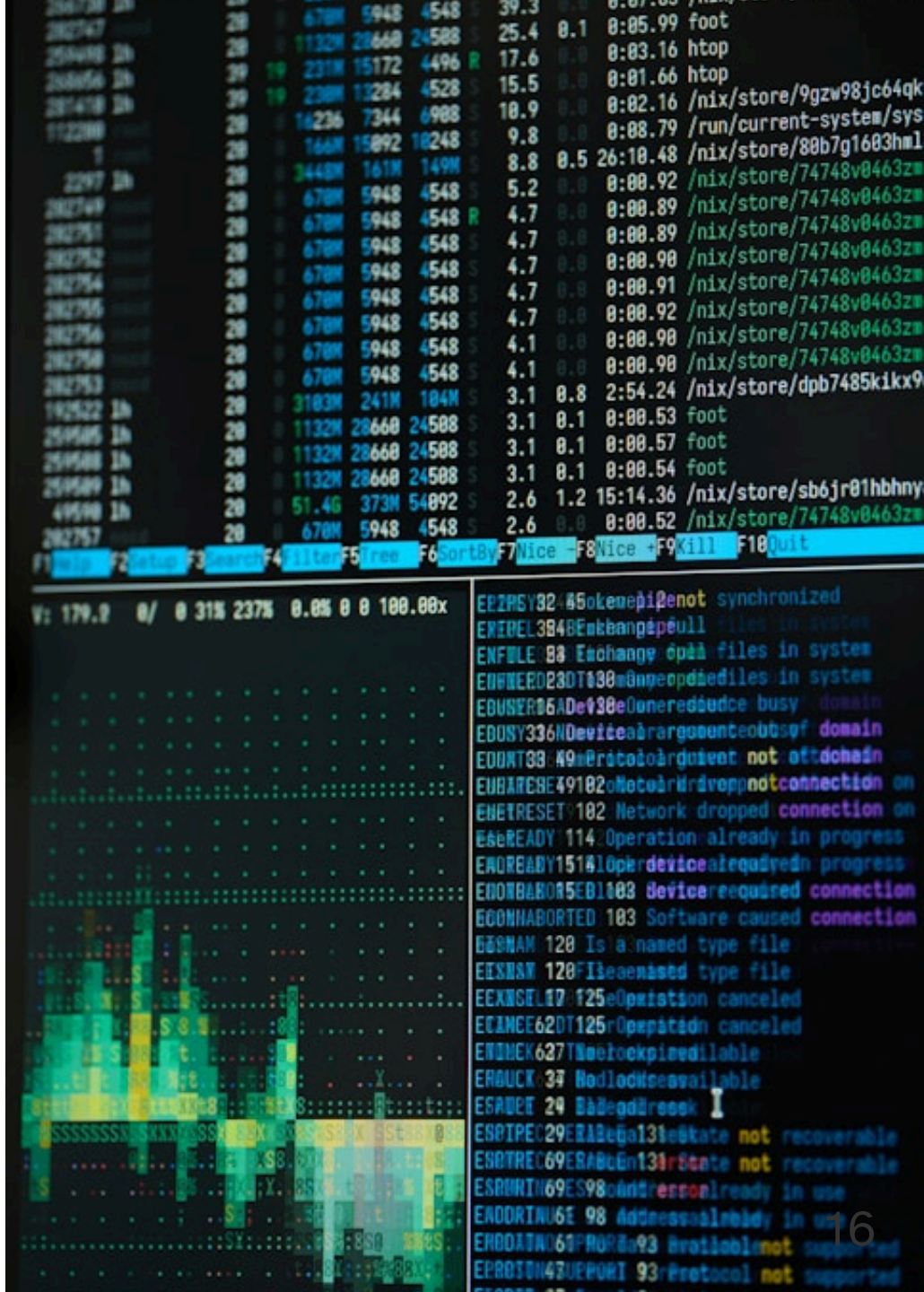
```
# Exemple avec curl pour tester l'API RESTful de notre application Laravel
curl -s -H "Authorization: Bearer <token>" -H "Accept: application/json" \
  http://localhost:8000/api/posts
```

Développer une API RESTful avec Laravel

Notre application Laravel suit déjà en grande partie ces principes.

Il suffit de l'adapter pour qu'elle soit pleinement RESTful.

Laravel met des outils à notre disposition pour cela.



Différencier les routes MVC des routes API

	Routes MVC	Routes API
Fichier	`routes/web.php`	`routes/api.php`
Préfixe	*(aucun)*	`/api`
Usage	Interface utilisateur par un navigateur (pages HTML et sessions)	Consommation par des clients qui ne supporte pas HTML et les sessions
Middleware	`auth`	`auth:sanctum`

Laravel Sanctum pour l'authentification des API RESTful

Laravel Sanctum est un package Laravel qui permet de gérer l'authentification via des tokens pour les API RESTful.

Il permet de :

- Générer des **tokens d'authentification** pour les utilisateur.trices.
- Gérer les **permissions et les rôles** associés aux tokens.
- Permettre de **révoquer** ou **renouveler** les tokens.

Créer les tokens pour les utilisateur.trices

- Afin de permettre à des clients de s'authentifier via des tokens, il faut d'abord créer ces tokens pour les utilisateur.trices.
- Pour cela, il faut :
  - Proposer une interface pour que les utilisateur.trices génèrent des tokens ⇒ Nouveau domaine/ressource dans l'application : **gestion des tokens**.
  - Nécessite des vues, routes et contrôleurs dédiés.
- Une fois les tokens créés, les clients peuvent les utiliser pour s'authentifier auprès de l'API RESTful.

Utiliser les tokens pour authentifier les requêtes API (1)

Inclure le token dans l'en-tête `Authorization` avec l'en-tête `Accept: application/json` pour indiquer que le client souhaite recevoir une réponse au format JSON.

```
# Exemple avec curl pour tester l'API RESTful de notre application Laravel
curl -s \
  -H "Authorization: Bearer <token>" \ # Le token est utilisé ici
  -H "Accept: application/json" \      # Le client souhaite recevoir du JSON
  http://localhost:8000/api/posts      # Requête à l'API RESTful
```

Utiliser les tokens pour authentifier les requêtes API (2)

- Sanctum vérifie automatiquement la validité du token.
- Si valide : la requête est authentifiée.
- Sanctum associe automatiquement la personne authentifiée à la requête.
- Permet de contrôler l'accès aux différentes fonctionnalités de l'API en fonction des permissions associées au token en utilisant les gates et les policies de Laravel.

Gérer les permissions et les rôles des utilisateur.trices avec les tokens d'authentification

- Des **permissions** (*abilities*/scopes) peuvent être associées à chaque token.
- Permet de contrôler l'accès aux différentes fonctionnalités de l'API en fonction des permissions associées au token.

```php
// Vérifie si le token a la permission `posts:create`
$request->user()->tokenCan('posts:create');
```

Conclusion

- L'architecture **RESTful** est adaptée pour des services web consommés par des clients légers (mobiles, scripts en ligne de commande, applications SPA).
- **Laravel Sanctum** gère l'authentification via des tokens.
- Les tokens peuvent porter des **permissions** pour contrôler l'accès à l'API.

Questions

Est-ce que vous avez des questions ?

Feedback

Le formulaire de feedback vous **permet de partager votre retour** sur le cours *"DévProdMéd"*.

Il ne prend **que quelques minutes** et est **anonyme**.

Les résultats seront discutés à la fin du cours. **Merci beaucoup !**

À vous de jouer !

- (Re)lire le contenu de cours.
- Faire les exercices.
- Faire le mini-projet.
- Poser des questions si nécessaire.

➡️ Visualiser le contenu complet sur GitHub.

N'hésitez pas à vous entraidez si vous avez des difficultés !

Sources

- Illustration principale par Richard Jacobs sur Unsplash
- Illustration par Aline de Nadai sur Unsplash
- Illustration par Hal Gatewood sur Unsplash
- Illustration par Lukas sur Unsplash
- Illustration par Nikita Kachanovsky sur Unsplash