SEPTEMBER 10, 2021 - LILLE & ONLINE

API PLATFORM CONFERENCE

Unified permission management between API & client

MARION AGÉ

LES-TILLEULS.COOP CTO

@marion_age

présentation

Marion AGÉ

Directrice technique & co-gérante
@ Les-Tilleuls.coop

Développeuse back-end & e-commerce
Plus récemment convertie à Vue.js

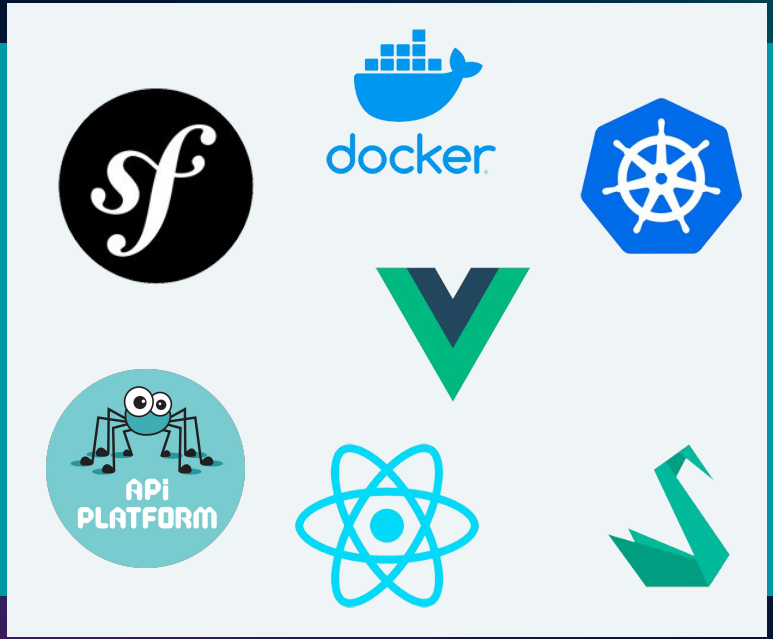
Jamais loin de mon vélo 🚴‍♀️

API PLATFORM CONFERENCE

Les-Tilleuls.coop

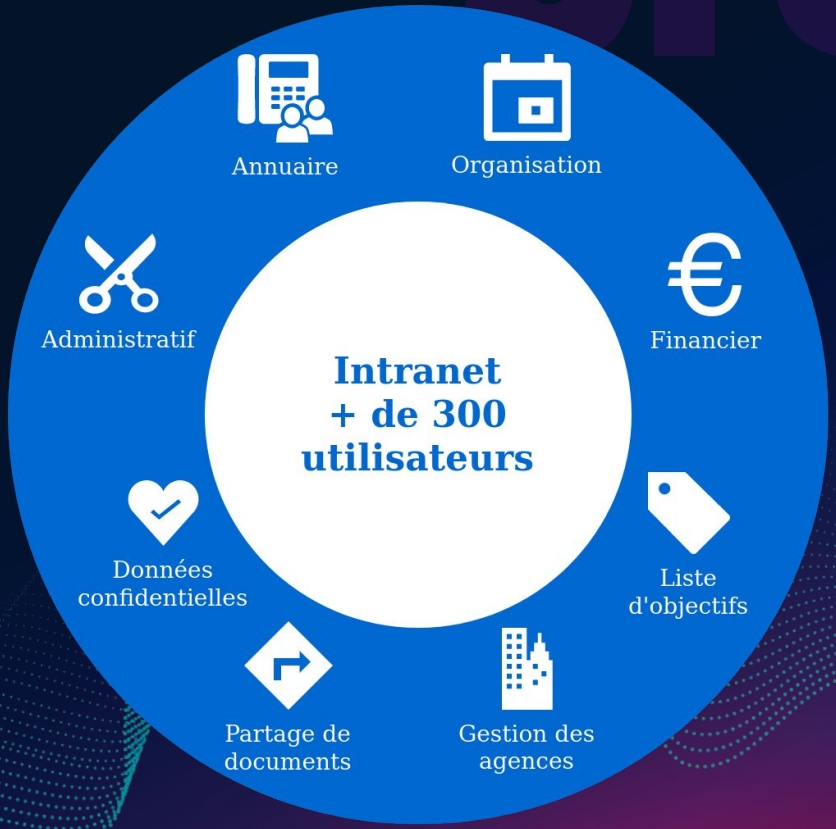
- ✔ **Scop auto-gérée** depuis 2011 🎂
- ✔ Fonctionnement **démocratique & égalitaire**
- ✔ **+ de 50** personnes
- ✔ Lille, Paris, Nantes, Lyon…

👷‍♀️👷🏾 jobs@les-tilleuls.coop



API PLATFORM CONFERENCE

Le projet



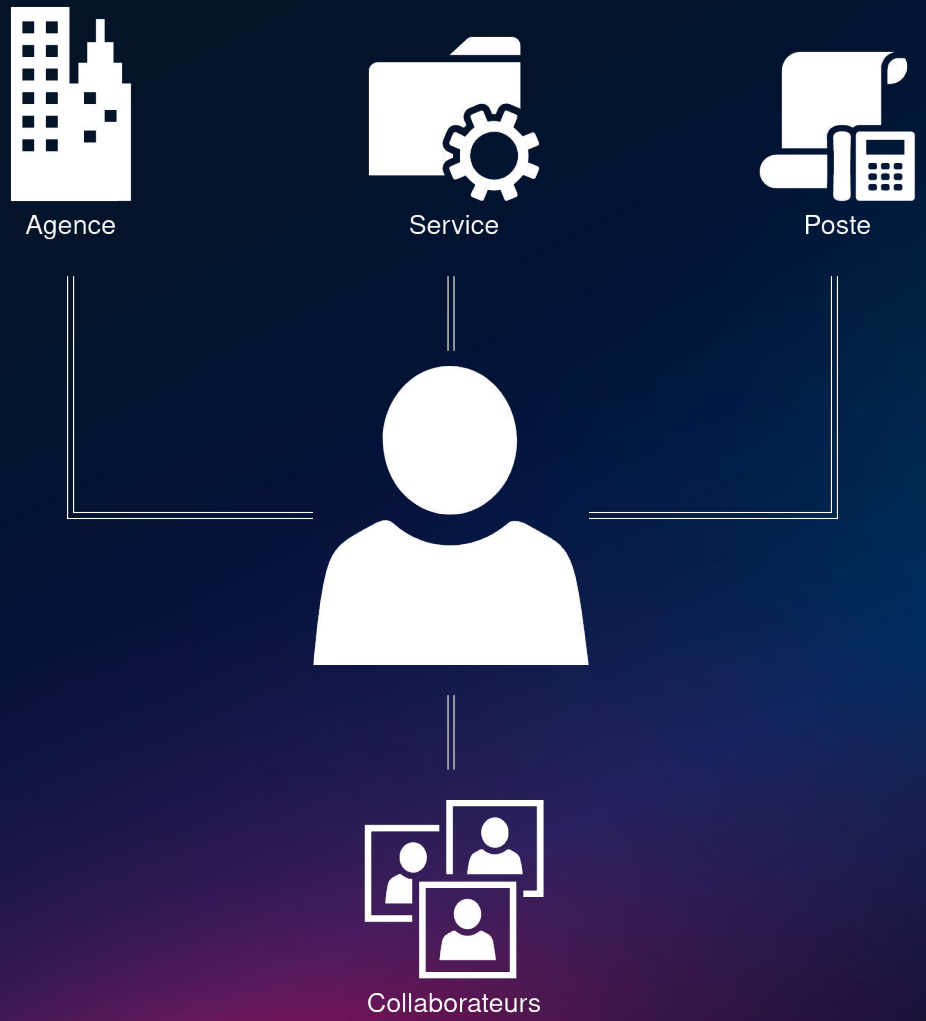
Un **intranet** partagé

Utilisé par **plus de 300** personnes

Présence de données personnelles et **confidentielles**

Des droits très différents en fonction de chacun·e par **besoin** ou par **sécurité** des données

API PLATFORM CONFERENCE



API PLATFORM CONFERENCE

Les fonctionnalités

ACCÈS SÉCURISÉ

L'API et le frontend ne sont accessibles qu'à des utilisateurs **authentifiés**.

SYSTÈME DE PERMISSIONS

Mise en place d'un système de **permissions** permettant des **règles** métier "complexes" d'accès aux ressources.

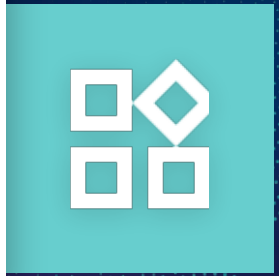
Les fonctionnalités

INTERFACE D'ADMINISTRATION

Une **interface graphique** permet aux **administrateurs** de gérer les permissions en fonction des règles établies.

INTERFACE PERSONNALISÉE EN FONCTION DES DROITS

L'**interface graphique** de l'intranet est **personnalisée** automatiquement pour chaque personne en fonction des **droits** qui lui sont octroyés.



Les fonctionnalités

SÉCURISATION DE L'API

L'API est sécurisée automatiquement : accès aux **ressources**, **opérations**, **filtrage** des données...

MISE À JOUR EN TEMPS RÉEL DES INTERFACES

L'**interface graphique** des utilisateurs connectés est mise à jour automatiquement lorsque les permissions sont modifiées, **sans rechargement**.

API PLATFORM CONFERENCE

Première étape :

Sécurisation de l'API

Fonctionnalités **API**	
	Gérer les permissions selon des règles administrables
	Sécuriser les opérations
	Filtrer automatiquement les collections
	Appliquer des groupes de sérialisation automatiques

Les fonctionnalités **natives**

Directives de contrôle d'accès global

```yaml
# config/packages/security.yaml

security:
    access_control:
        - { path: ^/docs, roles: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/foo, roles: ROLE_FOO }
        - { path: ^/, roles: IS_AUTHENTICATED_FULLY }
```

Les fonctionnalités **natives**

Contrôle d'accès au niveau des ressources & opérations

```
#[ApiResource(
    collectionOperations: [
        'get',
        'post' => ['security' => "is_granted('ROLE_ADMIN')"],
    ],
    itemOperations: [
        'get',
        'put' => [
            'security' => "is_granted('ROLE_ADMIN') or object.owner == user"
        ],
    ],
    security: "is_granted('ROLE_USER')",
)]
class Book {}
```

Les fonctionnalités **natives**

Accès restreint aux propriétés

```php
class Book
{
    //...

    /**
     * @var string Property viewable and writable only by users with ROLE_ADMIN
     */
    #[ApiProperty(
      security: "is_granted('ROLE_ADMIN')",
      securityPostDenormalize: "is_granted('UPDATE', object)"
    )]
    private $adminOnlyProperty;
}
```

```php
final class CurrentUserExtension implements QueryCollectionExtensionInterface,
  QueryItemExtensionInterface {
    // public function applyToCollection(// ...): void
    // public function applyToItem(// ...): void

    private function addWhere(QueryBuilder $queryBuilder, string $resourceClass): void
    {
        // $user = $this->security->getUser() ...
        $rootAlias = $queryBuilder->getRootAliases()[0];
        $queryBuilder->andWhere(sprintf('%s.user = :current_user', $rootAlias));
        $queryBuilder->setParameter('current_user', $user->getId());
    }
}
```

1. Gérer les règles de permissions

Chaque **fonctionnalité** qui nécessite des droits spécifiques est associée à un **code unique** et autant de **règles** que nécessaires, administrables.

Fonctionnalité = Permission + PermissionRules

- ✔ Une permission **sans règles** est accessible à tous
- ✔ Un utilisateur peut être éligible à plusieurs règles, il faut donc établir des **priorités**

```php
#[ORM\Entity]
#[ApiResource]
class Permission
{
    #[ORM\Id, ORM\Column, ORM\GeneratedValue]
    private ?int $id = null;

    #[ORM\Column(unique: true)]
    private ?string $code;

    #[ORM\OneToMany(mappedBy: 'permission', targetEntity: PermissionRule::class)]
    private Collection $rules;
}
```

```php
#[ORM\Entity]
#[ApiResource]
class PermissionRule
{
    #[ORM\Id, ORM\Column, ORM\GeneratedValue]
    private ?int $id = null;

    #[ORM\ManyToOne(inversedBy: 'rules')]
    #[ORM\JoinColumn(nullable: false)]
    private Permission $permission;

    #[ORM\ManyToOne]
    private ?Organization $organization = null;

    #[ORM\ManyToOne]
    private ?Service $service = null;

    #[ORM\ManyToOne]
    private ?Position $position = null;

    #[ORM\Column(type: 'boolean', options: ['default' => false])]
    private bool $manager = false;

    #[ORM\Column(type: 'simple_array', nullable: true)]
    private ?array $filters = [];

    #[ORM\Column(type: 'simple_array', nullable: true)]
    private ?array $groups = [];
}
```

2. Gérer les **droits** et le **filtre** automatique sur les opérations

On établit une **règle de nommage** des permissions liées aux opérations pour que le moteur de recherche de permissions connaisse les règles à chercher sur chaque opération de l'API.

Exemple : permission "`api_users_get_collection`"

Grâce à une **extension** API Platform, on cherche les règles régissant l'accès à la route pour l'utilisateur courant :

- ❌ Si l'utilisateur n'a pas l'autorisation => error **403**
- ✔️ S'il a l'accès, on rend la collection, en **filtrant** éventuellement les résultats

```
{
    "@id": "/permission_rules/34",
    "@type": "PermissionRule",
    "permission": "/permissions/23",
    "position": "/positions/46",
    "manager": false,
    "filters": [
        "organization",
        "service"
    ],
    "groups": [
        "user:admin"
    ]
}
```

```php
#[\Attribute(\Attribute::TARGET_CLASS | \Attribute::IS_REPEATABLE)]
class FilterPermission
{
    private const USER_VALUES = [
        'organization' => 'user.getOrganization()',
        'manager' => 'user.getId()',
         // ...
    ];

    public string $name;
    public string $field;
    public string $value;

    public function __construct(string $name, ?string $field = null, ?string $value = null)
    {
        $this->name  = $name;
        $this->field = $field ?: $name;
        $this->value = $value ?: self::USER_VALUES[$name] ?: '';
    }
}
```

```php
#[ApiResource]
#[FilterPermission(name: 'organization')]
#[FilterPermission(name: 'manager')]
class User {}



#[ApiResource]
#[FilterPermission(name: 'organization', field: 'id')]
class Organization {}
```

```php
private function addWhere(QueryBuilder $queryBuilder, string $resourceClass): void
{
    // ...
    $rule = $this->permissionManager->getPermissionForRoute($routeName);
    if (!$rule || empty($rule->getFilters())) {
        return;
    }

    // ...
    foreach ($rule->getFilters() as $filter) {
        // ...
        // retrieve field & value for current user
        $queryBuilder
            ->andWhere(sprintf('%s.%s = :filter_%s', $rootAlias, $field, $filter))
            ->setParameter(sprintf('filter_%s', $filter), $value);
    }
}
```

```php
public function getPermissionForRoute(string $route): ?PermissionRule
{
    if (null === $permission = $this->repos->findOneBy(['code' => $route])) {
        return null;
    }

    return $this->findRule($permission);
}

protected function findRule(Permission $permission): PermissionRule
{
    $priority = -1;
    $matchedRule = null;
    foreach ($permission->getRules() as $rule) {
        if (!$this->isEligible($rule) || $rule->getPriority() <= $priority) {
            continue;
        }

        $matchedRule = $rule;
        $priority = $rule->getPriority();
    }

    if (!$matchedRule) {
        throw new AccessDeniedException();
    }

    return $matchedRule;
}
```

3. Appliquer des **groupes de sérialisation** dynamiques

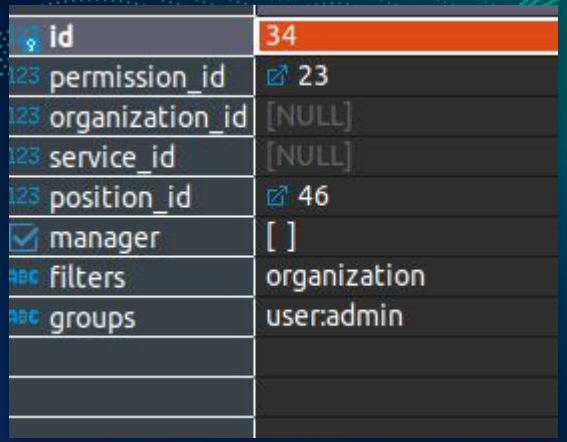
On s'appuie sur un **Context Builder** pour récupérer les éventuels **groupes de sérialisation dynamiques** à ajouter, configurés dans la règle appliquée.

```php
public function createFromRequest(// ...): array
{
    // ...
    if (
        (null === $rule = $this->permissionManager->getPermissionForRoute($routeName))
        || empty($rule->getGroups())
    ) {
        return $context;
    }

    $context['groups'] = array_merge($context['groups'], $rule->getGroups());

    return $context;
}
```

```php
#[ApiResource(
    normalizationContext: ['groups' => ['user']],
)]
class User
{
    // ...
    #[Groups("user:admin")]
    private ?User $manager = null;
}
```




```json
{
    "@id": "/users/89",
    "@type": "User",
    "email": "directeur_lille@foo.com",
    "firstname": "Neoma",
    "lastname": "Vandervort",
    "organization": "/organizations/16",
    "service": "/services/30",
    "position": "/positions/46",
    "manager": "/users/88"
},
```

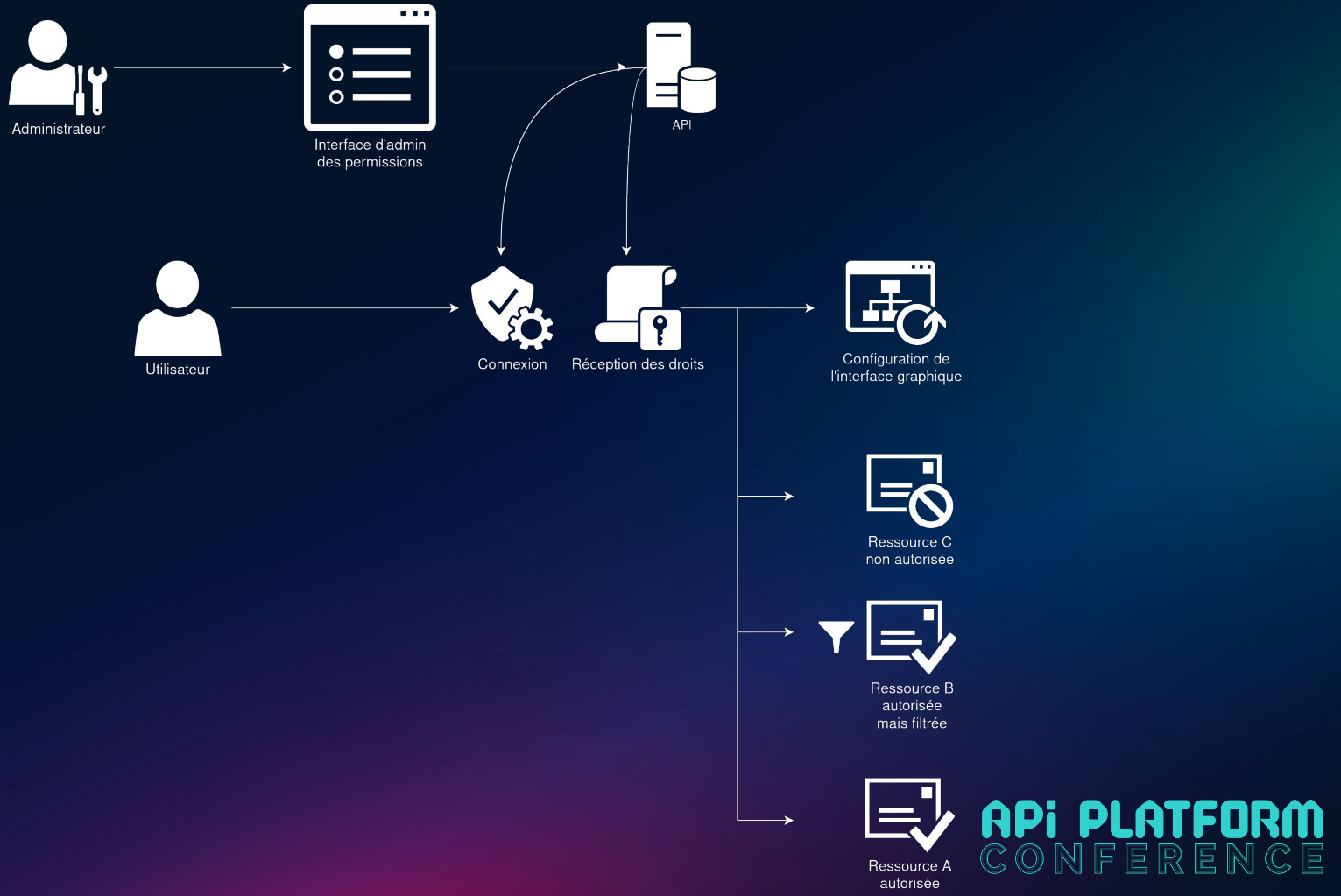
API PLATFORM CONFERENCE

Fonctionnalités **API**	
✓	Gérer les permissions selon des règles administrables
✓	Sécuriser les opérations
✓	Filtrer automatiquement les collections
✓	Appliquer des groupes de sérialisation automatiques

Deuxième étape :

Gestion des permissions côté frontend

Fonctionnalités **Frontend**	
	Interface de gestion des permissions (type CRUD)
	Authentification auprès de l'API
	Réceptionner les permissions de l'utilisateur connecté
	Adapter le menu aux permissions courantes
	Configuration du router
	Affichage différencié en fonction des droits


Administrateur
Interface d'admin
des permissions
API
Utilisateur
Connexion
Réception des droits
Configuration de
l'interface graphique
Ressource C
non autorisée
Ressource B
autorisée
mais filtrée
Ressource A
autorisée
API PLATFORM
CONFERENCE

1. Authentification et récupération des permissions

Authentification obligatoire pour accéder aux parties protégées de l'application frontend.

Authentification par **token JWT**

Création d'un **Cookie** httpOnly, secure, à expiration courte + mise en place d'un **refresh token**.

```php
public function onAuthenticationSuccessResponse(AuthenticationSuccessEvent $event)
{
    // ...

    $data = $event->getData();
    $data['user'] = $userData;
    $data['permissions'] = $this->permissionManager->getAllPermissions();

    // ...

    $event->setData($data);
}
```

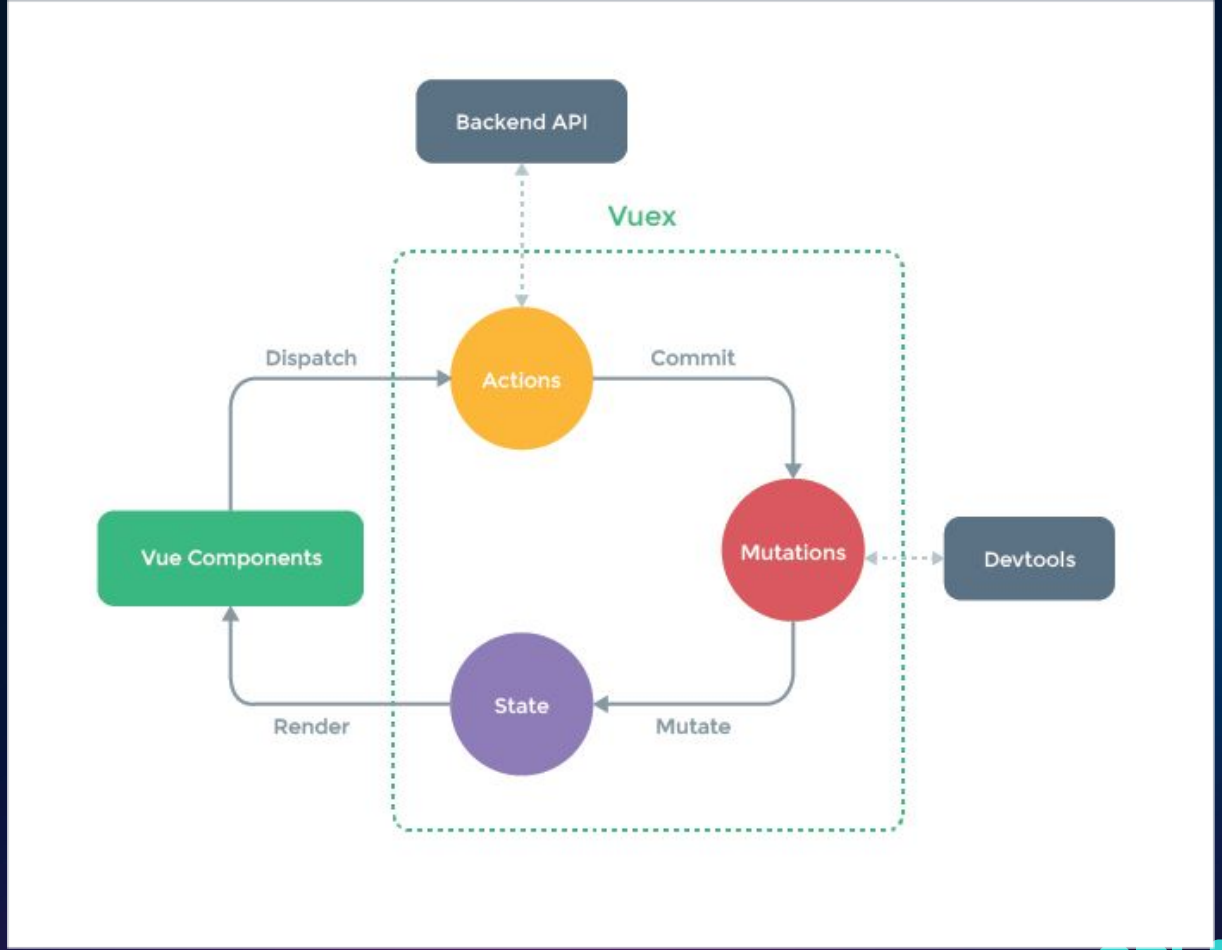
```
{
    "user": {
        "@context": "/contexts/User",
        "@id": "/users/88",
        "@type": "User",
        "firstname": "Tianna",
        "lastname": "Ziemann"
    },
    "permissions": [
        "api_users_get_item",
        "api_users_get_collection",
        "do_something_fun",
        "other_permission_code",
    ],
    "refresh_token": "xxx"
}
```

Qu'est-ce qu'un store ?

L'application frontend récupère les **permissions** de l'utilisateur courant au moment de la connexion et les garde en **mémoire**.

Store : espace de stockage **global** de l'état (**state**) de l'application :

- ➔ **réactif**
- ➔ **immutable** de façon directe (**commit** => **mutations**)



source : https://vuex.vuejs.org/

```
const store = {
    state: {
        // ...
    },
    getters: {
        // ...
    },
    actions: {
        // ...
    },
    mutations: {
        // ...
    },
};
```

```
const store = {
    state: {
        permissions: [],
        user: null,
    },
    getters: {
        can: (state) => (code) => state.permissions.includes(code),
        isLogged: (state) => !!state.user,
        permissions: (state) => state.permissions,
        user: (state) => state.user,
    },
    actions: {},
    mutations: {},
};
```

```
const store = {
  state: {},
  getters: {},
  actions: {
    async LOGIN_REQUEST({ commit }, user) {
      commit('LOGIN_REQUEST');
      const res = await new AuthApi().login(user);
      commit('LOGIN_SUCCESS', res.data);

      return res;
    },
    async REFRESH_TOKEN({ commit }) {
        // ...
    },
  },
  mutations: {},
};
```

```
const store = {
  state: {},
  getters: {},
  actions: {},
  mutations: {
    LOGIN_REQUEST(state) {
      state.user = null;
      setRefreshToken(null);
    },
    LOGIN_SUCCESS(state, resp) {
      state.user = resp.user;
      state.permissions = resp.permissions;
      setRefreshToken(resp.refresh_token);
    },
  },
};
```

2. Définition du Menu en fonction des droits

```
<nav>
  <div>
    <router-link :to="{ name: 'homepage' }">
      Home
    </router-link>

    <router-link
      :to="{ name: 'users' }"
      v-if="can('api_users_get_collection')"
    >
      Users
    </router-link>

    <router-link
      :to="{ name: 'organizations' }"
      v-if="can('api_organizations_get_collection')"
    >
      Organizations
    </router-link>
  </div>
</nav>
```

2. Définition du Menu en fonction des droits

```vue
<script setup lang="ts">
import { useStore } from "vuex";

const store = useStore();
const can = (code: string) => store.getters['auth/can'](code);
</script>
```

3. Sécurisation du router

Le **router** doit vérifier les permissions de l'utilisateur courant **avant de valider la navigation**.

➔ Certaines routes sont accessibles à tous les utilisateurs
➔ D'autres sont restreintes

=> "**navigation guards**" (intercepteurs) du routeur

```
const routes = [
  {
    path: '/login',
    name: 'login',
    component: () => import('@/views/Login.vue'),
  },
  {
    component: () => import('@/SecuredApp.vue'),
    meta: { requiresAuth: true },
    path: '/',
    children: [
      {
        path: '/',
        name: 'homepage',
        component: () => import('@/views/Homepage.vue'),
      },
      {
        path: '/users',
        name: 'users',
        meta: {
          permission: 'api_users_get_collection',
        },
        component: () => import('@/views/Users.vue'),
      },
      // ...
    ],
  },
];
```

```
export const secureApp = (router: Router) => {
  router.beforeEach(async (to, from, next) => {
    if (!to.matched.some((record) => record.meta.requiresAuth)) {
      return next();
    }

    if (!store.getters['auth/isLogged']) {
        // ...
    }

    if (canAccessRoute(to)) {
      return next();
    }

    return next({ path: '/' });
  });
};
```

4. Fonctionnalités selon les droits

On peut ajouter des **données arbitraires** dans les **meta** du router et s'en servir ensuite dans nos pages et composants.

Les possibilités sont **multiples** pour différencier le rendu : composants distincts, affichage ou non de blocs, d'actions, de textes, ajout de classes dynamiques, appels asynchrones...

```
{
  path: '/users',
  name: 'users',
  meta: {
    permission: 'api_users_get_collection',
    actions: ['api_users_post_collection', 'show_logs', 'do_something_fun'],
  },
  component: () => import('@/views/Users.vue'),
},
```

```vue
<template>
  <div class="buttons">
    <button v-for="action in authorizedActions" :key="action">{{ action }}</button>
  </div>
</template>

<script setup lang="ts">
import { computed } from 'vue';
import { useRoute } from 'vue-router';
import { useStore } from 'vuex';

const store = useStore();
const can = (code: string) => store.getters['auth/can'](code);

const route = useRoute();
const authorizedActions = computed(
  () => route.meta.actions?.filter((action) => can(action)) || []
);
</script>
```

Fonctionnalités **Frontend**	
✓	Interface de gestion des permissions (type CRUD)
✓	Authentification auprès de l'API
✓	Réceptionner les permissions de l'utilisateur connecté
✓	Adapter le menu aux permissions courantes
✓	Configuration du router
✓	Affichage différencié en fonction des droits

Troisième étape :

Mise à jour des permissions en temps réel

- ➔ La mise à jour des permissions côté API déclenche **une mise à jour via Mercure**
- ➔ Mercure **notifie tous les clients connectés** de la mise à jour de leurs permissions
- ➔ L'interface des utilisateurs est **redéfinie de façon transparente** en fonction des nouveaux droits (grâce à la **réactivité**)

```php
#[ApiResource(mercure: true)]
class Permission {}



#[ApiResource(mercure: true)]
class PermissionRule {}
```

```
import { subscribe } from '@/api/mercure';
import { store } from '@/store';

export const useMercure = (cb: Function) => {
  let stateEvent: EventSource|null = null;

  const subscribePermissions = async () => {
    stateEvent = subscribe(async () => {
      await store.dispatch('auth/REFRESH_TOKEN');
      cb();
    });
  }

  const unsubscribePermissions = () => {
    stateEvent = null;
  }

  return {
    subscribePermissions,
    unsubscribePermissions,
  }
}
```

```
// SecuredApp.vue
import { onMounted, onDeactivated } from 'vue';
import { useRoute } from 'vue-router';
import { useMercure } from '@/composables/useMercure';
import { redirectNoAuthorized } from "@/router/security";

const currentRoute = useRoute();
const { subscribePermissions, unsubscribePermissions } =
  useMercure(() => redirectNoAuthorized(currentRoute));

onMounted(() => subscribePermissions());
onDeactivated(() => unsubscribePermissions());
```

Conclusion

- ✔ Système de **permissions** avec des **règles métier** +/- complexes
- ✔ Des règles **administrables** par un non tech qui impliquent :
  - ✓ la sécurisation des **points d'accès** de l'API
  - ✓ un **filtre** automatique des données
  - ✓ l'ajout de **groupes** de sérialisation dynamiques
  - ✓ l'**affichage conditionné** de l'application frontend
  - ✓ la sécurisation des **routes** du frontend
  - ✓ la **mise à jour** des permissions en temps réel

API PLATFORM CONFERENCE

Merci !