ÉQUIPEMENTS

Architecture web des systèmes d'information

Résumé

Le projet "Équipements", réalisé en binôme, consistait à développer un site web de gestion du matériel universitaire avec Angular, Express et MongoDB.

Théo BONTEMPS, Dorian DESCAMPS

theo.bontemps@etud.u-picardie.fr, dorian.descamps@etud.u-picardie.fr

1. Introduction

Le projet "Équipements" a été réalisé dans le cadre de la matière Architecture Web des SI. Le projet, réalisé en binôme, avait pour objectif principal de développer un site web via les frameworks Angular et Express et le tout avec base de données MongoDB. Ce site devait permettre la gestion du matériel au sein de d'une université.
Ce rapport présente un compte rendu détaillé de notre travail, en expliquant les idéologies de développement, les choix techniques effectués, ainsi que les fonctionnalités implémentées dans le site.

2. Code : Quelques bonnes pratiques

2.1. Backend

2.1.1. Objets

Les objets sont au cœur de l'application backend, représentant les entités utilisées dans le site comme User, Loan, Equipment, Organization, etc. Chaque entité est modélisée par une classe dédiée, encapsulant les propriétés et les méthodes pour gérer les états et les comportements associés. Par exemple, la classe Loan contient des méthodes pour insérer, mettre à jour, supprimer et formater les données d'un prêt, assurant une gestion cohérente et centralisée des opérations liées aux prêts.

Exemple

```
class Loan {
  …

  async isRunning() {
    …
  }

  async insert() {
    …
  }

  async update() {
    …
  }

  async delete() {
    …
  }

  format() {
    …
  }

…
```

2.1.2.Schémas

Les schémas de validation sont utilisés pour vérifier que les requêtes reçues par l'application respectent une structure prédéfinie, garantissant ainsi l'intégrité des données et la sécurité. Express-validator est utilisé pour définir des règles de validation qui s'appliquent aux champs des requêtes. Cela permet de s'assurer que les données entrantes sont correctement formatées et respectent les contraintes de validation avant d'être traitées par l'application.

Exemple

```
…
bodyPassword: (prefix, isOptional = false) => createValidationRule(
    [
      body(`${prefix}password`)
        .isString()
        .withMessage('PASSWORD_MUST_BE_A_STRING')
        .isLength({ min: parseInt(process.env.USER_PASSWORD_MIN_LENGTH, 10) })
        .withMessage('PASSWORD_TOO_SHORT'),
    ],
    isOptional,
  ),
…
```

2.1.3.Authentification

Les jetons d'authentification sont utilisés pour sécuriser les communications entre le client et le serveur. L'application génère des jetons sécurisés à l'aide de bibliothèques comme crypto, en utilisant des techniques de cryptographie telles que la génération de valeurs aléatoires. Cela permet de s'assurer que seuls les utilisateurs authentifiés peuvent accéder aux ressources.
Les mots de passe sont hachés en en utilisant des sels générés eux même avec des méthodes cryptographiques. Cela garantit la sécurité des comptes utilisateurs en cas de fuite de données.

Exemple

```
static generateTokenValue() {
  return crypto
    .randomBytes(process.env.TOKEN_LENGTH / 2)
    .toString('hex');
}

static generateSaltValue() {
  return crypto
    .randomBytes(process.env.SALT_LENGTH / 2)
    .toString('hex');
}

static hashPassword(password, salt) {
  return crypto
    .createHash('sha512')
```

```
    .update(password)
    .update(
      crypto
        .createHash('sha512')
        .update(salt)
        .digest('hex'),
    )
    .digest('hex');
}
```

2.1.4. Intégrité des données

En plus des vérifications sur les schémas des requêtes garantissant la qualité des données, chaque route provoquant un changement en base s'assure que celui-ci ne provoquera pas de corruption de données. Il n'est alors par exemple pas possible de supprimer un utilisateur qui aurait des prêts.

Exemple

```
if (await Loan.userExists(user)) {
  return res
    .status(409)
    .send({ errors: [{ msg: 'USER_HAS_LOAN' }] });
}
```

2.2. Frontend

2.2.1. Auth Guards

Les classes UserAuthGuard et AdministratorAuthGuard démontrent l'utilisation de gardes de routes pour contrôler l'accès aux différentes parties de l'application en fonction de l'authentification et des rôles des utilisateurs. Cela assure que seules les personnes autorisées peuvent accéder à certaines routes, renforçant ainsi la sécurité de l'application.

2.2.2. Programmation réactive et observables

L'utilisation de RxJS et des observables permet de gérer les flux de données asynchrones et les événements de manière réactive. Les observables fournissent un moyen puissant pour émettre, écouter et transformer des flux de données, tandis que les opérateurs pipe, map, tap et catchError facilitent la composition et la gestion des opérations asynchrones et des erreurs. Cette approche permet un contrôle précis sur le traitement des données et l'enchaînement des opérations, améliorant ainsi la réactivité et la robustesse de l'application.

Exemple

```
changePassword(
  email_address: string,
```

```
  old_password: string,
  new_password: string,
): Observable<void> {
  const body = {
    user: { email_address, old_password, password: new_password },
  };

  return this.http.post<void>(
    `${environment.API_BASE_URL}/account/password`,
    body,
  );
}
---
this.accountService
    .changePassword(email, oldPassword, newPassword)
    .pipe(
      tap(() => {
        alert('PASSWORD_CHANGED');
        window.location.reload();
      }),
      catchError((error) => FrontendService.catchError(error)),
    )
    .subscribe();
```

2.2.3. Validation des formulaires

L'utilisation de ReactiveFormsModule pour créer et gérer des formulaires complexes avec des validations personnalisées assure que les données saisies par les utilisateurs sont correctement validées avant d'être envoyées au serveur. Les messages d'erreur en temps réel améliorent l'expérience utilisateur.

Exemple

```
this.formGroup = this.formBuilder.group({
  loanType: ['', Validators.required],
  organization: [''],
  room: ['', Validators.required],
});

this.formGroup.get('loanType')?.valueChanges.subscribe((loanType) => {
  if (loanType === 'ORGANIZATION') {
    this.formGroup.get('organization')?.setValidators(Validators.required);
  } else {
    this.formGroup.get('organization')?.clearValidators();
  }

  this.formGroup.get('organization')?.updateValueAndValidity();
});
```

2.2.4. Services

La logique métier et les opérations liées aux données sont déplacées dans des services dédiés, comme UserService, pour maintenir les composants légers et focalisés sur la gestion de l'interface utilisateur.

Exemple

```
export default class UserService {
  constructor(private http: HttpClient) {}

  getOrganizations(): Observable<any> {
    …
  }

  getLoans(): Observable<any> {
    …
  }

  …
}
```

2.2.5. Gestion centralisée des erreurs

La gestion des erreurs est centralisée via une méthode dédiées nommée FrontendService.catchError, assurant une gestion uniforme et réutilisable des erreurs à travers l'application. Cette fonction évite également la duplication excessive de code.

Exemple

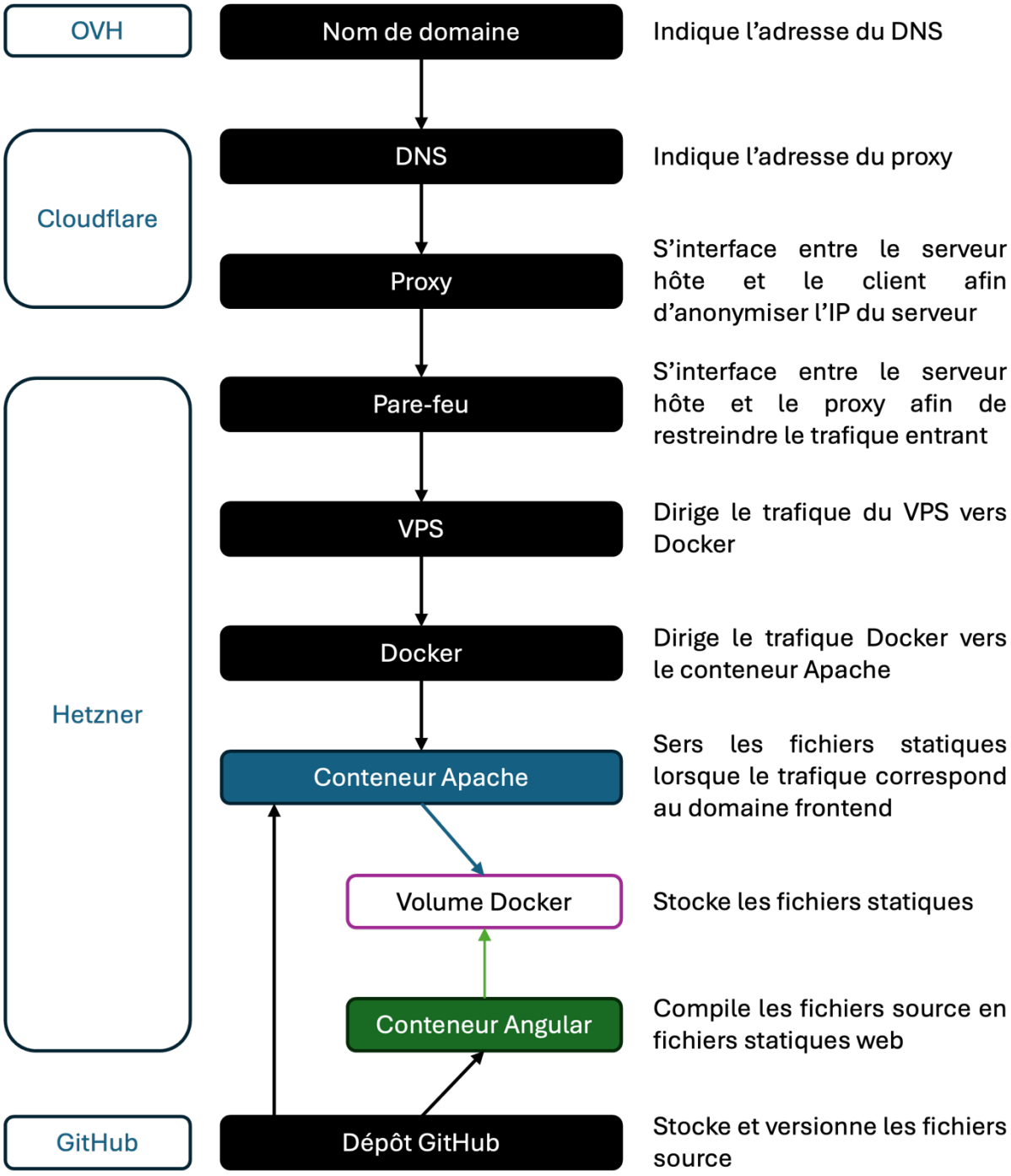
```
static catchError(error: any): Observable<never> {
    const message = error.error?.errors?.[0]?.msg || 'NOT_HANDLED_ERROR';
    alert(message);

    return of();
  }
```

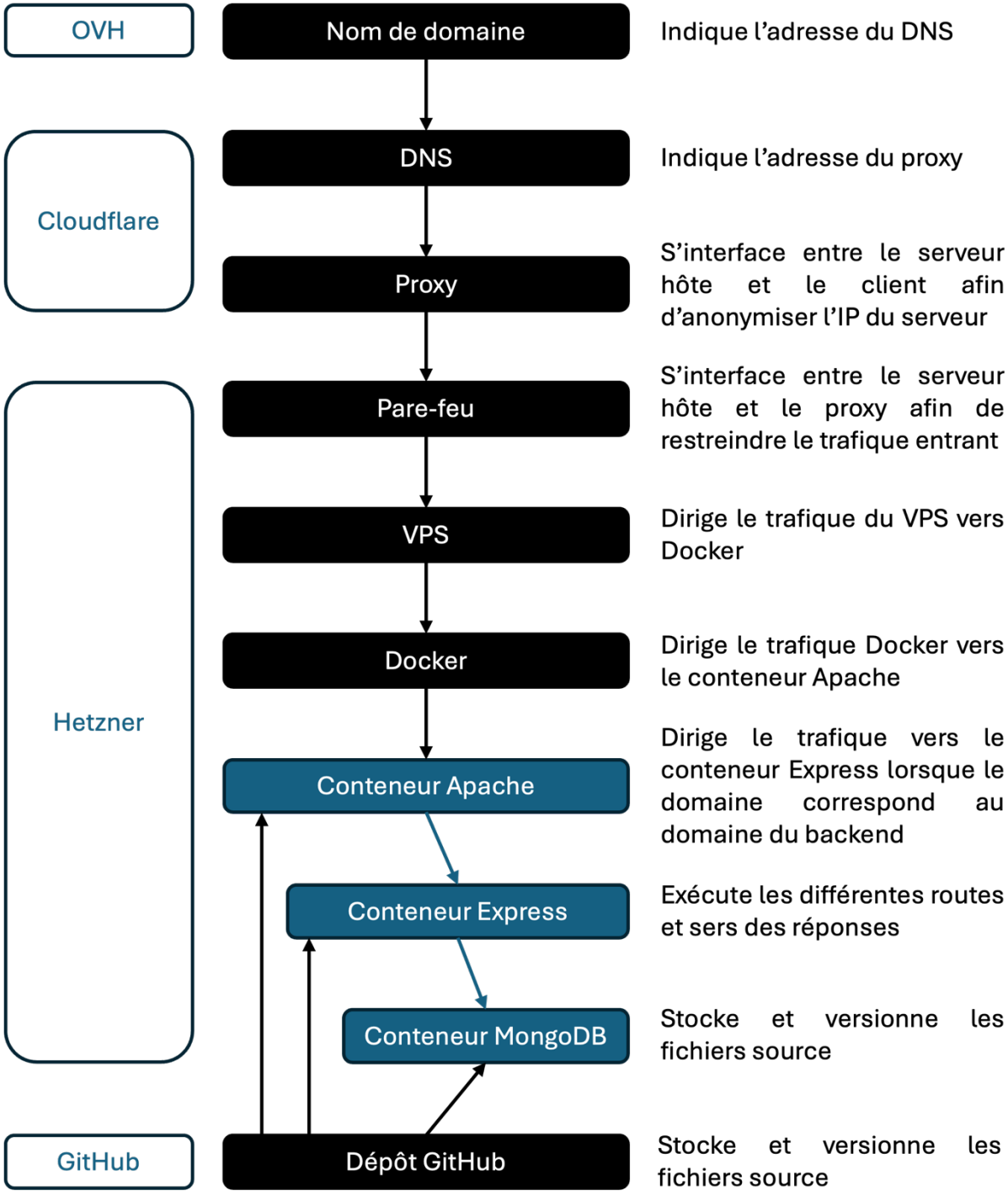
3. Architecture

L'architecture présentée ci-dessous est optimisée pour un environnement de production et est destiné à pouvoir être utilisé en extranet.
Elle met en place de nombreux principes de sécurité tels que l'authentification, la conteneurisation, l'offuscation et le chiffrage.
Elle suit des normes de sécurité élevées en termes de stockage des informations de sécurité, et vérifie la conformité et l'auteur de toutes les requêtes adressées au backend.

3.1. Frontend


OVH
Nom de domaine
Indique l'adresse du DNS
DNS
Indique l'adresse du proxy
Cloudflare
Proxy
S'interface entre le serveur hôte et le client afin d'anonymiser l'IP du serveur
Pare-feu
S'interface entre le serveur hôte et le proxy afin de restreindre le trafique entrant
VPS
Dirige le trafique du VPS vers Docker
Docker
Dirige le trafique Docker vers le conteneur Apache
Hetzner
Conteneur Apache
Sers les fichiers statiques lorsque le trafique correspond au domaine frontend
Volume Docker
Stocke les fichiers statiques
Conteneur Angular
Compile les fichiers source en fichiers statiques web
GitHub
Dépôt GitHub
Stocke et versionne les fichiers source

3.2. Backend



4. Test

Note : Merci de consulter la procédure jointe pour un test en local.

Le site est disponible à l'adresse https://equipment.bnts.fr.
Les comptes par défaut sont les suivants.

Rôle	Adresse e-mail	Mot de passe
Administrateur	frederic.furst@u-picardie.fr	angularr
Utilisateur	theo.bontemps@etud.u-picardie.fr	angularr
Utilisateur	dorian.descamps@etud.u-picardie.fr	angularr

5. Conclusion

5.1. Fonctionnalités

Le site est complètement fonctionnel. Les fonctionnalités réalisées sont mises en relief en vertes parmi toute celles requises et optionnelles. Une unique fonctionnalité, qui s'avère être optionnelle, n'a pas été réalisée.

5.1.1. Fonctionnalités obligatoires

5.1.1.1. Gestion des comptes utilisateurs

- Administrateur
  - Ajouter un compte utilisateur.
  - Supprimer un compte utilisateur.
  - Modifier son propre compte.
- Utilisateurs
  - Modifier leur propre compte.

5.1.1.2. Gestion du matériel

- Administrateur
  - Accepter une demande d'attribution de matériel.
  - Accepter le retour d'un matériel.
  - Ajouter un matériel dans la base.
  - Retirer un matériel de la base.
- Utilisateurs
  - Consulter les matériels disponibles, par type.
  - Consulter les matériels qui leur ont été attribués.
  - Demander à se faire attribuer un matériel.
  - Demander à rendre un matériel.

5.1.2. Fonctionnalités optionnelles

5.1.2.1. Tableau de bord

- Administrateur
  - Visualiser le nombre de matériels par type.
  - Visualiser les matériels entrés ou sortis depuis une certaine date.
  - Gestion des dates de renouvellement :
    - Ajouter des dates de renouvellement pour certains matériels.
    - Recevoir des alertes pour les matériels à renouveler.
  - Gestion des délais de retour :

- Ajouter des délais de retour pour certains matériels.
  → Réalisé partiellement, une date de retour par défaut est définie.
- Recevoir des alertes pour les matériels à retourner.

- Règles d'attribution de matériel :
  - Ajouter des règles spécifiques sur les utilisateurs pour l'attribution de matériel.

5.2. Quelques captures d'écran