

Table des matières

1.	Liste des compétences du référentiel qui sont couvertes par le projet	3
2.	Résumé du projet en anglais	4
3.	Cahier des charges du projet.....	5
	3.1. Contexte du projet	5
	3.2. Fonctionnalités attendues	5
	3.3. Résultats attendus	6
	3.4. Public cible	6
4.	Gestion de projet	7
	4.1. Diagramme de Gantt	7
	4.2. Tableau Scrum	8
5.	Spécifications fonctionnelles du projet.....	11
	5.1 Gestion d'utilisateurs	11
	5.2 Gestion du matériel	13
	5.3 Réservation et location des équipements	14
	5.4 Gestion du planning en temps réel	16
	5.5 Gestion de notifications	17
6.	Spécifications techniques du projet	18
	6.1 Base de données	18
	6.2 Back-end avec SpringBoot	19
	6.3 Front-end avec Angular	21
	6.4 React Native	22
	6.5 Architecture 3 tiers	23
7.	Réalisations.....	26
	7.1 Conceptualisation	26
	7.2 L'API avec Spring Boot	35
	7.3 Front end avec Angular	46
8.	Présentation du jeu d'essai	50
9.	Description de la veille	53
10.	Description d'une situation de travail	56

10.1 Déploiement de la base de données	57
10.2 Déploiement du Backend	58
10.3 Déploiement du Frontend	59

1. Liste des compétences du référentiel qui sont couvertes par le projet

1. Maquetter une application
2. Développer une interface utilisateur de type desktop
3. Développer des composants d'accès aux données
4. Développer la partie front-end d'une interface utilisateur web
5. Développer la partie back-end d'une interface utilisateur web
6. Concevoir une base de données
7. Mettre en place une base de données
8. Développer des composants dans le langage d'une base de données
9. Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement
10. Concevoir une application
11. Développer des composants métier
12. Construire une application organisée en couches
13. Développer une application mobile
14. Préparer et exécuter les plans de tests d'une application
15. Préparer et exécuter le déploiement d'une application

2. Résumé du projet en anglais

As my final project I've chosen Locparc, the **equipment rental management app**, the reason behind my choice are the project's specifications and its completeness (everything seen in class was covered by the project's specifications). The project consists of making a web and mobile app which takes care of equipment renting/lending for IFA and MNS primarily. The two establishments have lots of equipment that is used during saloons, open-door days, classes inside the establishments and loans to interns.

The app lets the administrator create user accounts, know the location of every piece of material in the database as well as its condition, the items that are in need of repair, items sent to a maintenance facility, items that are available for lease or lending, items leased and the person to whom they were leased, and the admin receives notifications for expired item licenses. The admin can also access the history of leased items. The above mentioned functionalities are available in real time.

The users of the app can create and send requests for leasing available items and send it to the administrator to be validated. They can follow the state of their request in real time as well as have access to the planning where the available items are listed in real time as well.

In conclusion Locparc aims to optimize the management of the equipment park in real time, plan preventive maintenance operations, track rental history and facilitate reservation requests for optimal use of each equipment.

3. Cahier des charges du projet

En tant que stagiaire en formation à l'IFA et MNS, j'ai à vous présenter le cahier des charges d'un projet qui me passionne car j'ai beaucoup appris en le réalisant.

J'ai développé une application de gestion et de location de matériel pour mon centre de formation. C'était une super opportunité pour moi de mettre en pratique les compétences acquises au cours de l'année et de réaliser un projet concret.

Le cahier des charges qui m'a été fourni par le centre de formation est minutieusement détaillé. Le choix de projet a été fait en étudiant plusieurs cahiers des charges de plusieurs projets (huit, plus précisément), et j'ai choisi Locparc car il me semblait le plus complet et que je trouvais le fait de faire un logiciel de gestion de matériel très utile en termes de ce qu'on peut voir en entreprise.

Contexte du projet :

L'IFA (Institut Français des Affaires) et MNS (Metz Numeric School) disposent d'un parc de matériel utilisé dans diverses situations, comme des salons, des formations, des prêts à des stagiaires, etc. Pour mieux gérer ce parc, ils ont besoin d'une application qui leur permettra de suivre en temps réel chaque matériel et de connaître son état. Ils souhaitent (optionnellement) inclure la gestion des licences logicielles.

Mon objectif ultime était de mettre en ligne l'application web Locparc avant le 31 juillet 2023 à 08h29. J'ai également dû partager mon projet via Github à Franck Quirin, le futur super administrateur de Locparc et administrateur de MNS.

Dans ce projet, le client, monsieur Franck Quirin, a contribué dans le développement de la maquette en exprimant ses idées et en donnant ses conseils sur l'expérience utilisateur. Il a également validé les choix concernant le graphisme, l'ergonomie, les contenus et les techniques utilisées. Bien entendu, tout cela en respectant le cahier des charges et les délais fixés. Il m'a également fourni les contenus de base tels que les textes, les logos, les images, etc.

Fonctionnalités attendues :

Locparc doit répondre à plusieurs objectifs importants.

Tout d'abord, les utilisateurs pourront louer du matériel, faire des demandes de location qui seront validées (ou pas) par l'administrateur plus précisément, les emprunteurs pourront signaler tout événement lié au matériel, comme un dysfonctionnement, une panne, une demande de retour, une prolongation de prêt, ou même une nouvelle demande de prêt. Ensuite, les responsables ont une interface de

gestion complète pour suivre l'état du parc, consulter les matériels en location et ceux qui n'ont pas été restitués. De plus, l'application doit permettre une gestion de planning pour visualiser la localisation des matériels ainsi que les réservations passées et futures. Enfin, un système d'alerte devra signaler tout retard dans le retour du matériel ou toute nouvelle demande émanant d'un utilisateur.

Résultats attendus :

Grâce à Locparc, MNS et IFA obtiennent plusieurs résultats significatifs. Tout d'abord, un suivi en temps réel des équipements, quel que soit l'activité, en termes de localisation, d'état, de planning de réservation et de documentation. Cela leur permettra d'être plus efficaces dans la gestion de leurs stocks et d'éviter les pertes ou les oublis.

Public cible :

Le cahier des charges stipule que l'application s'adresse principalement aux entreprises et aux associations qui prêtent du matériel à leurs collaborateurs. Mais il y a également une deuxième cible, les gestionnaires des stocks et les associations caritatives constituent une cible secondaire.

4. Gestion de projet

Nous étions dans une équipe de deux personnes au début du projet, c'était au moment là où nous avons conçus le planning pour le développement et la conception de locparc et nous sommes décidés sur l'utilisation du cadre de travail Scrum afin de maximiser notre efficacité.

Nous avons d'abord commencé avec la conception de la base de données. Avec la méthode Scrum nous avons appris à mettre en place des sprints, qui sont là dans le but d'atteindre le Product Goal (Objectif de produit, qui va de soi).

1. Diagramme de Gantt

Nous nous sommes également servis du diagramme de Gantt (un outil de gestion de projet) pour mettre en place des sprints. Nous nous sommes décidées sur une durée de deux semaines (hors week-ends) pour un sprint à cause des cours, des obligations de chacun et pour se donner une marge confortable afin de ne pas commettre d'erreurs que devons rectifier plus tard à cause du manque de temps.

La troisième personne nous a rejoint vers la fin de la conception de la base de données. Nous l'avons familiarisé avec notre idée de conception pour Locparc et elle a rattrapé le retard assez rapidement.

Voici à quoi ressemblaient les sprints pour la conception.

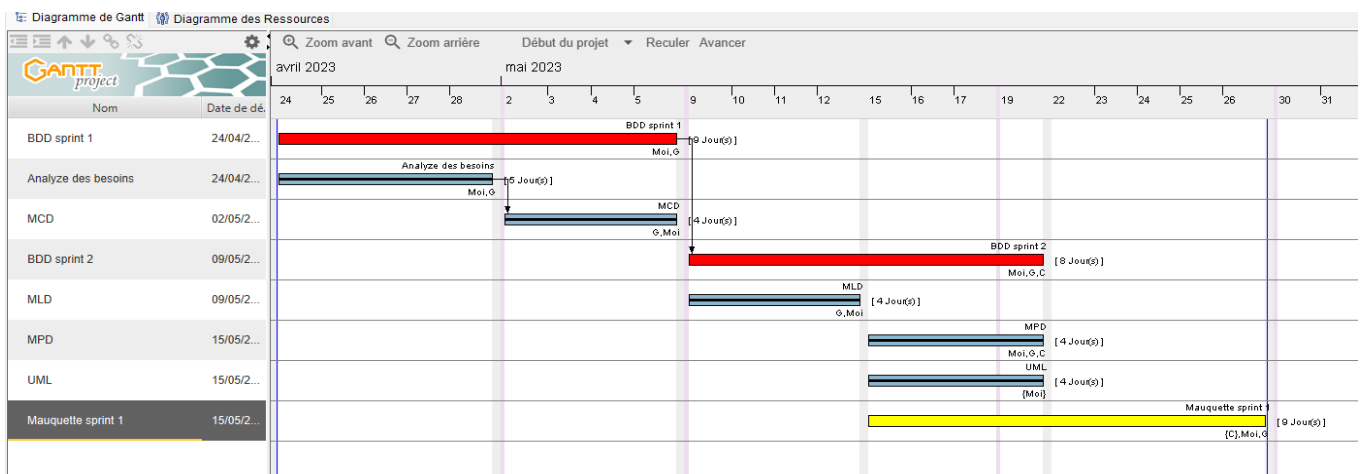


Figure 1. Diagramme de Gantt

Le choix d'utiliser Gantt seulement pour la conception de la base de données été basé sur notre utilisation de tableau Kanban qui nous avons préférés et où nous avons notés les User Stories qu'on n'était pas capable de transcrire aussi clairement dans Gantt.

Gantt étant un bon outil a été quand même rapidement remplacé par le tableau Kanban (qu'on a utilisé comme un Scrum Board).



Figure 2. Exemple d'un simple Scrum Board

2. Tableau Scrum

Une fois la conception de la base de données terminée, nous avons créés des User Stories qui sont des descriptions des fonctionnalités faites du point de vue d'un utilisateur dans notre Scrum Board. Un Scrum Board est un tableau qui contient le travail à faire, le travail en cours et un travail accompli.

Notre version a quelques champs de plus.

- La colonne Lexique: Nous avons créé un lexique qui nous a permis de définir les priorités des tâches qui sont à accomplir. Pour le lexique nous avons utilisé des codes couleurs avec les formes différentes spécialement faites pour des personnes daltoniennes. Etant une personne daltonienne ça m'a été d'une grande utilité dans la distinction des priorités.

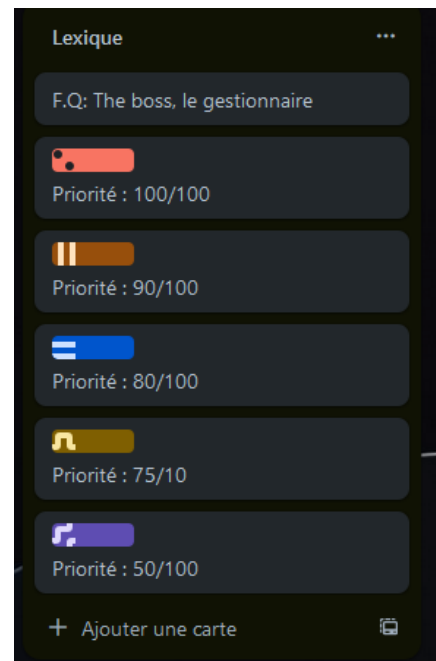


Figure 3. Le lexique de Locparc avec les formes pour daltoniens

- La colonne des User Stories avec les priorités définies pour chaque tâche avec une étiquette correspondante à une forme et une couleur depuis le lexique. On peut également cliquer sur une carte pour voir d'éventuels détails et nous pouvons également les modifier. Nous avons utilisés les images de couverture pour identifier l'auteur des cartes.

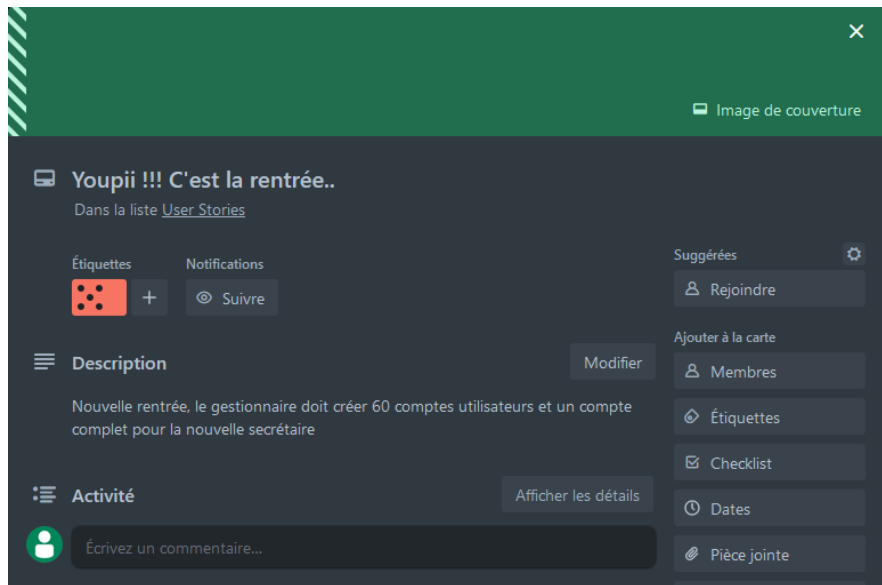


Figure 4. La vue « d'intérieur » d'une carte User Story

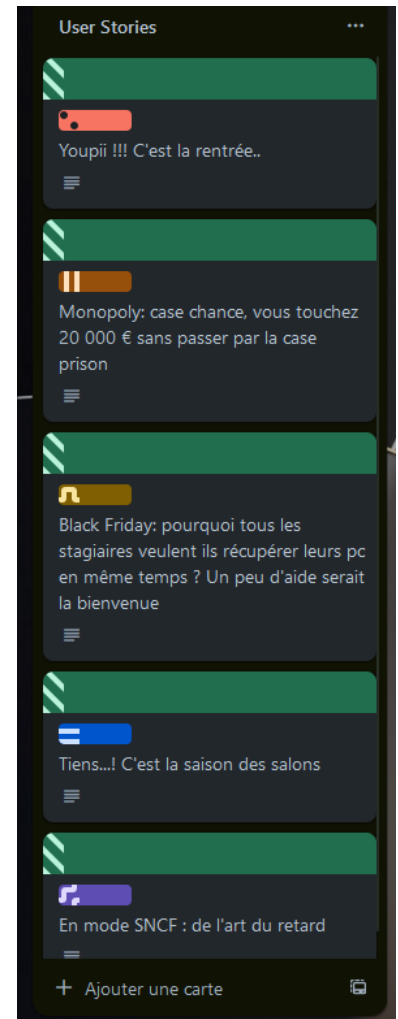


Figure 5. La colonne des User Stories

- La colonne Conception qui définit les fonctionnalités globales. Nous avons utilisé cette colonne pour définir des fonctionnalités à faire lors des sprints.
- La colonne A faire, pour des tâches à faire lors des sprints. Les premières cartes dans cette colonne correspondent aux cartes de la colonne Conception qu'on déplace et servent à indiquer que les tâches à faire lors du sprint actuel doivent être liées aux premières cartes.
- La colonne En cours pour des tâches qui sont en train d'être faits.
- Et dernièrement la colonne Terminé pour des tâches terminés.

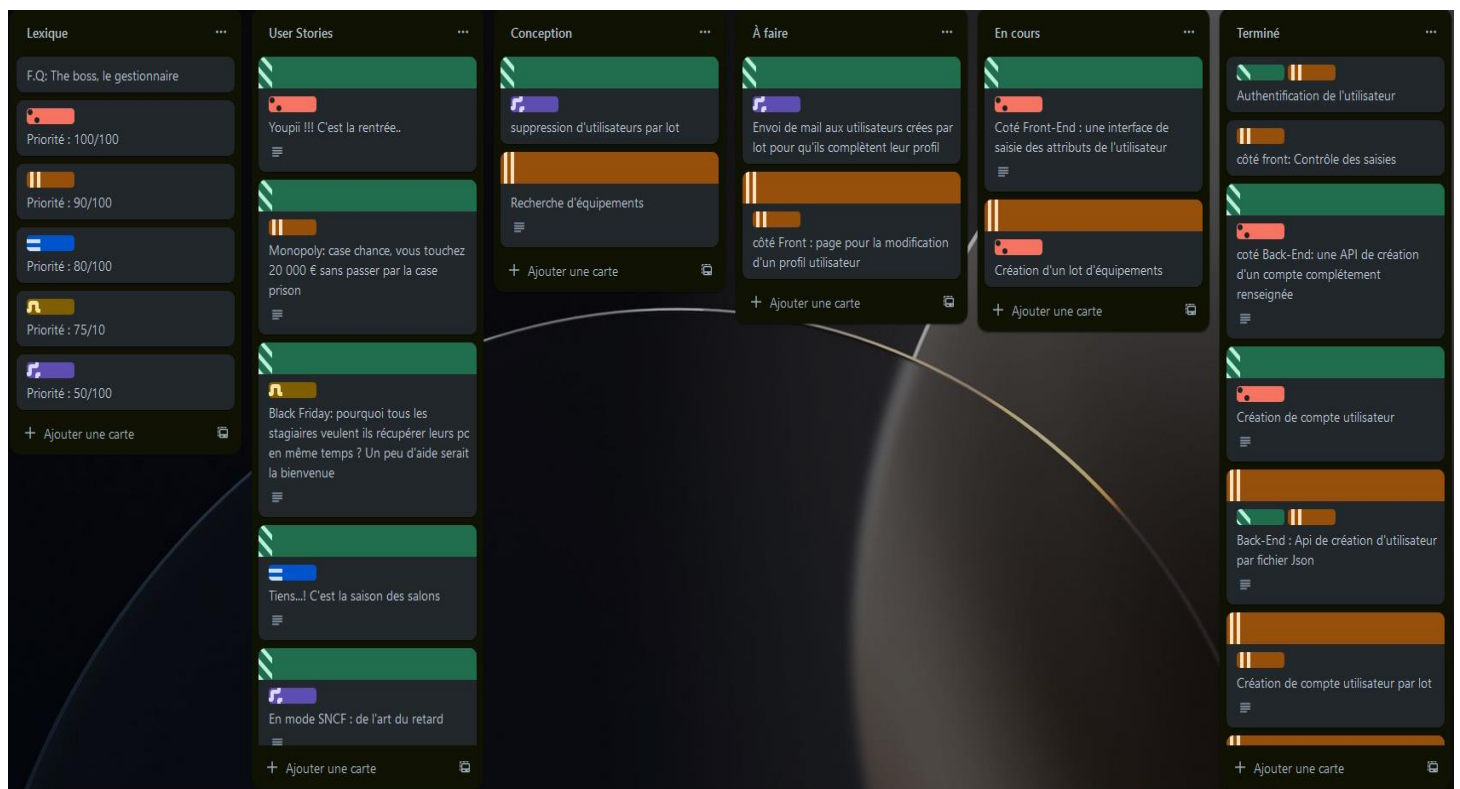


Figure 6. Le tableau Scrum de Locparc

5. Spécifications fonctionnelles du projet

Dans cette partie je vais vous présenter des spécifications fonctionnelles de Locparc.

Les spécifications fonctionnelles incluses dans Locparc sont les suivantes :

1. *Gestion d'utilisateurs* : droit d'accès, droit d'ajout, de modification ou de suppression de matériel ou d'autres utilisateurs, droit de location et de réservation etc.
2. *Gestion du matériel* : ajout d'équipements avec leurs spécifications comme le nom, la date d'arrivée du matériel, la marque, le modèle, catégorie et sous-catégorie etc., dire si le matériel est en maintenance, le coût des réparations...
3. *Réservation et location des équipements* : les utilisateurs peuvent réserver le matériel en fonction de sa disponibilité et surtout en fonction de la validation du gestionnaire
4. *Gestion du planning en temps réel* : affichage d'un planning en fonction de la disponibilité du matériel en prenant en compte les réservations et locations en cours.
5. *Envoi de notifications* : envoyer les notifications aux utilisateurs concernant la fin de la période de location, envoyer également les notifications aux gestionnaires les informant du dépassement éventuel de la date de fin de location ou du dépassement de la garantie de l'équipement.

1. *Gestion d'utilisateurs*

1.1 Droit d'accès :

Les visiteurs →

Les visiteurs dans le contexte présent sont les personnes sans un compte utilisateur. J'ai décidé de ne pas leur accorder accès à l'application Locparc. Cette décision était prise en prenant en compte le cahier des charges et la première cible de l'application Locparc qui est l'IFA et MNS. Les visiteurs n'ont accès qu'à la page de connexion qui les force à s'authentifier avant d'accéder à Locparc.

Les utilisateurs →

Les utilisateurs ont accès à l'application et son contenu une fois connectées, ainsi que le planning où ils peuvent faire des demandes de réservations ou de locations de matériel auprès du gestionnaire. Le compte de chaque utilisateur sera créé par le gestionnaire.

Les loueurs →

Les loueurs ont accès à l'application et son contenu une fois connectées, ils peuvent également valider les demandes de réservations ou de locations des utilisateurs. Les comptes des loueurs sont aussi créés par le gestionnaire.

Les gestionnaires →

Le premier gestionnaire est créé par le super-utilisateur et il peut ensuite créer d'autres gestionnaires ou loueurs ainsi que les utilisateurs. Les gestionnaires ont leur propre interface utilisateur où ils peuvent accepter ou refuser les demandes de location ou réservation des utilisateurs.

1.2 Droit de lecture, création, modification, suppression d'utilisateurs/matériel :

Les utilisateurs →

Les utilisateurs peuvent consulter le matériel (lire sa description, voir la photo de l'équipement) voir également sa disponibilité et éventuellement faire une demande de réservation dans le cas d'un usage futur ou de location dans le cas dans usage immédiat. Les utilisateurs peuvent modifier leurs informations, ainsi que leurs mots de passe qui leur sont données par le gestionnaire.

Les loueurs →

Les loueurs peuvent consulter le matériel, ils peuvent faire des demandes de réservations ou de location auprès du gestionnaire et ils peuvent accepter ou refuser les demandes de réservations ou locations, mais ils n'ont pas le droit de créer, modifier ou supprimer des équipements ni de comptes d'utilisateurs. Les loueurs peuvent également changer leurs informations ainsi que leurs mots de passe.

Les gestionnaires →

Les gestionnaires ont le droit de créer, consulter, modifier ou supprimer un équipement ainsi qu'un utilisateur, un loueur ou un autre gestionnaire. Les gestionnaires n'ont pas accès aux mots de passe.

Les mots de passe sont générés et cryptés automatiquement au moment de la création des comptes.

2. Gestion du matériel

2.1 Catégorisation du matériel :

Avant d'ajouter le matériel le gestionnaire doit d'abord savoir sous quelle catégorie et sous-catégorie le matériel se situe. Si la catégorie du matériel n'existe pas, le gestionnaire peut la créer et ainsi pour les sous-catégories. Une fois le classement par catégorie et sous-catégorie effectué, le gestionnaire peut également classer l'équipement par marque et modèle. Cette classification par catégorie et par modèle aide dans la rapidité de la recherche du matériel.

2.2 Description du matériel :

Tout matériel ne nécessite pas une description, mais il est fortement conseillé au gestionnaire d'en ajouter une ce qui rend l'expérience utilisateur d'autant meilleure. Cette description du matériel peut également servir comme un manuel d'utilisation ou une description de l'état de l'équipement tout simplement.

2.3 Maintenance du matériel :

Ce qui est compris par maintenance du matériel peut être la maintenance préventive comme la mise à jour des licences, du système d'exploitation, de logiciels etc. Ou la maintenance corrective comme le casse d'un composant de l'équipement.

Locparc permet d'envoyer le matériel en maintenance si besoin. La maintenance peut être effectuée dans un centre de réparation tout comme dans les locaux de IFA ou MNS. Le prix éventuel d'une réparation peut être renseigné et stocké dans la base de données de Locparc.

Une fois marqué « en maintenance » l'équipement est enlevé du planning et n'est plus disponible pour la réservation ni pour la location. Si l'équipement était réservé avant d'être retiré du planning, l'utilisateur ou les utilisateurs concernés sont contactés par le gestionnaire les informant de la nouvelle situation et une éventuelle modification du contrat s'il y a bien eu un.

2.4 Historique du matériel :

Qui était le locataire ? Qui a autorisé la location ? Combien de fois l'équipement a été loué ? Combien de fois l'équipement était envoyé en maintenance ? Coût total des réparations de l'équipement ?

Locparc répond à toutes ces questions grâce à l'historique de l'équipement qui peut être consulté par le gestionnaire.

2.5 Gestion des stocks :

Locparc a une gestion des stocks où les utilisateurs peuvent consulter les équipements, leur disponibilité, la quantité disponible pour chaque équipement comme dans les figures illustrées ci-dessous :



Figure 1. Disponibilité d'un appareil photo Canon



Figure 2. Indisponibilité d'un appareil photo Canon

3. Réservation et location des équipements

3.1 Validation de demande de réservation ou de location :

La validation des demandes de réservations ou de locations peut être effectué par les gestionnaires ou des loueurs. Les demandes sont acceptées en se basant sur le principe FIFO (First In First Out) ou premier venu premier servi, qui fonctionne comme une queue dans un restaurant.

Les gestionnaires et les loueurs doivent donner un motif dans le cas d'une demande refusé. Les motifs possibles peuvent être l'indisponibilité de l'équipement qui n'est pas visible dans le planning, un oubli de renseignement d'une indisponibilité dans le planning suite à un envoi en maintenance, etc.

3.2 Durée de réservation de location :

La réservation de la location doit avoir une période définie de début (date et l'heure) et de fin (date et l'heure). Cette période n'a pas de durée maximale car les étudiants de l'IFA ou MNS peuvent faire leurs études pendant plusieurs années dans les établissements et que les établissements peuvent s'entre prêter du matériel.

Le contrat de location peut être prolongé sur demande.

3.3 Retour du matériel :

Le retour du matériel doit être effectué avant la date et l'heure de fin de contrat. Le gestionnaire, après inspection détaillée remet le matériel comme disponible dans le planning. Dans le cas d'un retard de retour du matériel le gestionnaire décide de la pénalité en fonction de l'équipement loué et la raison du retard. Dans le cas des endommagements la garantie déposée avant la location peut être retiré s'il s'agit d'une location payante. Les pénalités peuvent être financiers ou celles définies au préalable pendant la rédaction du contrat de location.

3.4 Contrat de location :

Le contrat de location peut être rédigé par le gestionnaire, loueur ou une personne désignée comme responsable (directeur de l'établissement, avocat, notaire etc.).

Le contrat de location doit contenir la date et l'heure de début de location ainsi que la date et l'heure de fin de location, le nom et prénom de l'utilisateur ainsi que le nom et prénom du gestionnaire ou loueur qui a donné l'approbation pour cette location. Le contrat doit également contenir le motif ou événement de la location ainsi que l'adresse de l'utilisation de l'équipement. Le contrat peut être modifié avec l'accord du gestionnaire dans la mesure de la disponibilité du matériel.

Un utilisateur ne peut pas faire de réservations qui sont plus de deux semaines ou 14 jours dans le futur. Un utilisateur qui a un contrat en cours et souhaite prolonger son contrat est prioritaire par rapport à l'utilisateur qui fait une demande de location ou de réservation, cependant un utilisateur ne peut pas prolonger son contrat si une réservation se trouve dans les dates de la prolongation souhaitée. Les utilisateurs peuvent modifier leur contrat avec l'accord du gestionnaire. Une fois le contrat modifié, le gestionnaire met à jour les disponibilités du matériel.

4. Gestion du planning en temps réel

4.1 Définition de planning :

Le planning est composé de trois parties. La première partie est une liste de location en cours. La deuxième partie consiste d'une liste de réservation à venir et la troisième et dernière partie est composé d'une liste de demandes qui n'ont pas encore été validées, triées par ancienneté, faisant en sorte de respecter le principe FIFO.

4.2 Accès au planning :

Les utilisateurs n'ont pas accès au planning de Locparc. Le planning est destiné aux gestionnaires ou d'éventuels loueurs créés par les gestionnaires.

4.3 Affichage côté utilisateur :

Comme précisé plus haut, les utilisateurs n'ont pas accès au planning. Les utilisateurs ont néanmoins, accès aux équipements disponibles sur la page des équipements qu'ils peuvent ajouter à leurs paniers.

Liste d'équipements

Appareils photo



Appareil photo CANON
Test standard CIPA, batterie et
carte mémoire incluses
19,99€ prix par jour

Quantité disponible

Sélectionnez la quantité



Appareil photo CANON
Test standard CIPA, batterie et
carte mémoire incluses
16,99€ prix par jour

Quantité disponible

Sélectionnez la quantité



Appareil photo CANON v3
Test standard CIPA, batterie et
carte mémoire incluses
22,99€ prix par jour

Quantité disponible

Sélectionnez la quantité

Projecteurs

5. *Gestion de notifications*

5.1 Type de notifications :

Les utilisateurs reçoivent des notifications concernant le dépassement de la date de fin de contrat, la validation ou le refus de leur demande de réservation/location ou l'annulation de leurs réservations validées.

Les gestionnaires reçoivent les notifications dans le cas d'un dépassement de la date de fin d'un contrat (où le gestionnaire peut envoyer un mail rappelant la personne concernée du dépassement de la date de fin de son contrat de location). Ils reçoivent également des notifications d'annulation de réservation et d'expiration de garantie d'un équipement.

5.2 Moyens utilisés pour l'envoi de notifications :

Email : les utilisateurs ainsi que les gestionnaires reçoivent les notifications de dépassement de la date de fin de contrat. Les utilisateurs reçoivent également des notifications de validation ou de refus de demandes de locations/réservations par email.

6. Spécifications techniques du projet

1. Base de données

Pour ma base de données j'ai fait le choix d'utiliser le langage SQL (Structured Query Language) car ma base de données est une base de données relationnelle, ce qu'il signifie que des tables sont reliées entre elles grâce aux clés étrangères qui sont les clés primaires d'autres tables. En effectuant des jointures de tables nous pouvons obtenir des objets A composés d'objets B (par exemple : un utilisateur qui a une adresse où l'adresse est une table à part).

Le choix d'une base de données relationnelle était évident au moment de conception de Locparc, parce que les bases de données relationnelles organisent les données en se basant sur le monde réel en permettant la séparation de données en tableaux qui, elles, représentent des entités du monde réel. En faisant ainsi il est plus facile de comprendre les relations entre les données et les manager ensuite.

En plus, en appliquant de bonnes pratiques de conception comme la normalisation, j'ai augmenté l'efficacité de ma base de données en rendant mes données atomiques (chaque champ contient une donnée qui ne peut plus être divisé en d'autre données), j'ai réduit la redondance de mes données en séparant les données qui peuvent se répéter dans leur propres tables (comme des adresses par exemple). J'ai également réduit des données orphelines, qui naissent quand une donnée liée à une autre est supprimé sans que l'autre donnée le soit, en créant des contraintes de clés étrangères (ce qu'il signifie que toutes les données liées doivent être supprimées) et ainsi garantissant l'intégrité des données.

Finalement, l'approche structurée des bases de données relationnelles utilisant des tables, des contraintes permet d'instaurer l'intégrité des données, une meilleure lisibilité ainsi que maintenabilité.

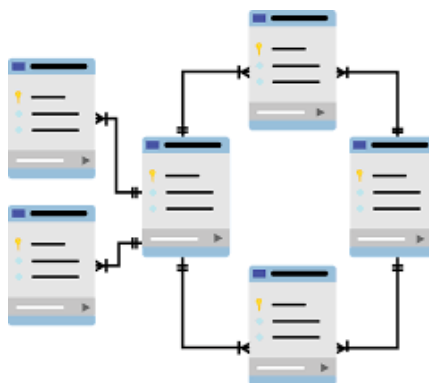


Figure 1. Une representation de base de données relationnelle

Au sujet du SGBDR (Système de Gestion de Bases de Données Relationnelles), j'ai fait le choix d'utiliser MySQL qui lui utilise SQL comme langage, avec des

fonctionnalités avancées qui lui sont propres. MySQL est un SGBDR connu pour ses performances et sa fiabilité et c'est pour ces raisons et parce que je le connais bien, j'ai décidé de l'inclure dans mon projet.



Figure 2. Logo de MySQL, le SGBDR choisi

J'ai ensuite choisi phpMyAdmin comme le GUI (Graphical User Interface) afin de mieux gérer mes bases de données MySQL. Avec phpMyAdmin je peux écrire des requêtes SQL, facilement consulter des données de mes tables, les trier rapidement dans l'ordre que je veux, avoir un affichage plus plaisant que celui de la ligne de commandes, etc.



Figure 3. Logo do PMA, le logiciel web de gestion pour mysql

2. Back-end avec SpringBoot

Au sujet de mon backend, j'ai créé une API (Application Programming Interface) avec Spring Boot (un framework Java très utilisé pour la création d'application Java) qui alimente mon front créé avec Angular et éventuellement l'application mobile créé avec React Native.

J'ai fait le choix d'utiliser Spring Boot grâce aux avantages qu'il offre :

1. L'autoconfiguration : en se basant sur les dépendances fournies dans le fichier pom.xml, Spring Boot configure notre application automatiquement et effectue la mise en place des dépendances sans que nous ayons à nous concentrer sur les configurations spécifiques qui nécessitent une véritable expertise dans le Framework Spring (Spring Boot étant une extension de Spring) qui peut prendre des mois voire des années à acquérir alors qu'il s'agit d'une simple mise en place. Une mise en place qui nous permet seulement de commencer à développer notre application. Grâce à l'autoconfiguration Spring Boot réduit énormément le temps de configuration d'un projet et m'a permis de rapidement commencer à développer mon projet.
2. Inversion de contrôle (Inversion of Control) et l'Injection de Dépendances (Dependency Injection) : il est essentiel de reconnaître les principes fondamentaux du Spring Framework, qui sont l'inversion de contrôle (IoC) et l'injection de dépendance (DI).
Ces principes ont joué un rôle important dans le développement de l'API pour Locparc en améliorant la flexibilité et la maintenabilité de mon backend.

L'Inversion de Contrôle transfère la responsabilité de l'instanciation et de la gestion des objets de l'API elle-même vers le framework. Au lieu de créer manuellement des objets et de gérer leurs dépendances, Spring Boot se charge de ces tâches. Cette inversion du contrôle permet un couplage faible entre les composants, ce qui favorise la modularité, facilite les tests et satisfait des critères de qualité d'un logiciel.

L'injection de dépendances (DI) est liée à l'Inversion de Contrôle et elle est une caractéristique clé de Spring. Avec l'injection de dépendances les objets dans mon backend reçoivent leurs dépendances de sources externes plutôt que de les créer en interne. Cette source externe, appelée injecteur de dépendances ou conteneur, injecte les dépendances requises dans les objets au moment de l'exécution. L'injecteur de dépendances reconnaît les dépendances grâce aux annotations de Spring Boot.

3. Communauté et fonctionnalités : Spring Boot bénéficie d'une large et active communauté, proposant une documentation étendue, des tutoriels et des ressources variées. Il existe des fonctionnalités supplémentaires pour travailler avec des bases de données relationnelles notamment.

L'écosystème dynamique garantit un soutien continu et une amélioration constante pour le développement de l'API pour Locparc et la documentation de Spring Boot est complète et bien structurée, qui nous offre des explications détaillées et des exemples pratiques qui m'ont guidé dans l'utilisation efficace du framework.



3. *Front-end avec Angular*

Le choix qui s'est présenté à moi pour le développement de mon front-end était celui de Angular vs React.

React étant une bibliothèque que je vais devoir éventuellement apprendre et maîtriser (rien que pour le fait qu'on peut créer des applications mobiles avec React Native qui est pareil mais il utilise des composants natifs au lieu de composants web), me semblait être un choix raisonnable, il est bien répandu et demandé et la courbe d'apprentissage n'est pas si raide que celle d'Angular alors qu'on peut faire la même chose. Mais il présentait un gros inconvénient pour moi, il n'est pas structuré.

Pendant mes recherches j'ai trouvé que la chasse pour la « bonne » bibliothèque était assez laborieuse et une fois trouvé souvent compliqué à adapter (n'ayant pas un bon niveau de maîtrise de React). Ceci dit je me suis vite découragé face au temps qu'il me restait pour terminer mon projet, ou même les fonctionnalités demandées.

J'ai choisi Angular comme le framework pour mon projet Locparc. Le choix était simple à faire parce que j'aime bien structurer mes projets et Angular m'a permis de faire ça. En plus le framework utilise TypeScript, du JavaScript mais mieux (parce que c'est fortement typé) !

J'ai une préférence pour Angular car tout est simple à trouver (si on veut une fonctionnalité, angular material est là pour nous aider), la documentation est très compréhensive ce qui rend le travail avec Angular très agréable sans parler de son CLI (command line interface).

L'interface de ligne de commande d'Angular nous permet de créer des composants, des services, des guards et plus encore avec une seule ligne. Nous pouvons également les supprimer en une seule ligne ainsi que les renommer, alors que tout ce processus requière une configuration manuelle en React ! Rien que pour le CLI j'aurais choisi Angular.

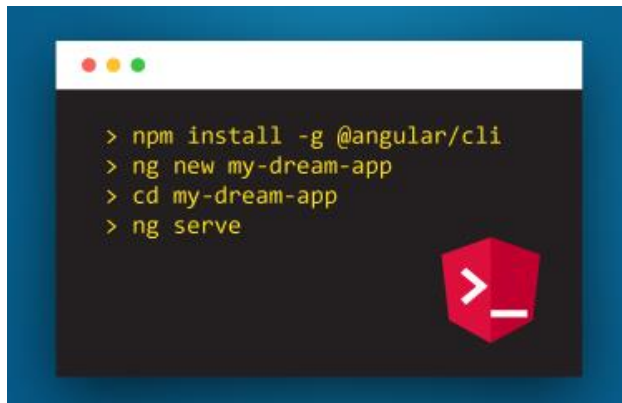


Figure 4. Le CLI d'Angular qui crée un projet et le démarre en quatre lignes

4. *React Native pour le client mobile*

J'ai entrepris de développer une application en utilisant le framework React Native. Cependant, malgré ma motivation et mon enthousiasme, j'ai rencontré un obstacle majeur qui m'a empêché de réaliser pleinement mon projet : le manque de temps.

Entre les cours, le backend du projet, le frontend du projet, l'apprentissage de la programmation orienté objet ainsi que l'architecture modulaire, et les autres engagements personnels, il était extrêmement difficile de trouver suffisamment de temps pour me consacrer pleinement au développement de mon application en React Native. Chaque journée était une course contre la montre, et malheureusement, le développement de l'application mobile a été mis au second plan.

Mon incapacité à développer l'application en raison du manque de temps a été source de frustration et de déception. Cependant, je me suis rendu compte que cette expérience m'a également appris des leçons précieuses. J'ai appris à mieux gérer mon emploi du temps, à hiérarchiser mes tâches et à être plus réaliste quant aux projets que je peux entreprendre dans le cadre de mes contraintes de temps.

Ceci dit, il se peut que j'aie un début d'application d'ici septembre s'il je trouve du temps qui est très précieux en ce moment.

5. Architecture 3 tiers

Mon but pour Locparc était de créer une application où les règles métier ne sont pas dictés ni par l'interface utilisateur ni par la base de données. Autrement dit je voulais que mon application soit modulaire, où des règles de métier (business rules) n'en dépendaient pas du code « bas niveau » ou des détails.

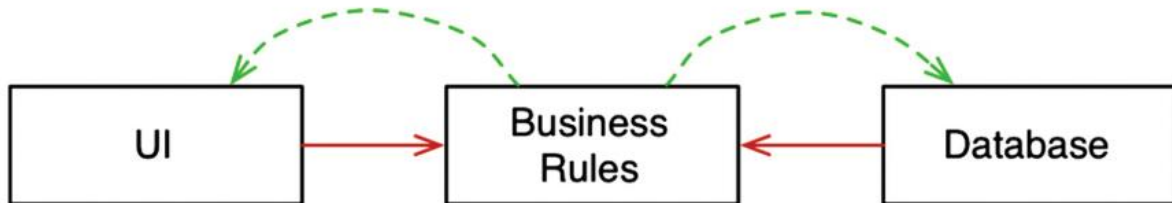


Figure 5. Capture d'écran illustrant une architecture modulaire prise du livre Clean Architecture – Robert C. Martin

L'architecture trois tiers me semblait être une bonne stratégie afin d'attendre une modularité ainsi qu'une extensibilité désirée pour Locparc.

5.1 Le niveau logique :

La logique métier est située dans l'API et est comme le cerveau de l'application. Cette couche est responsable de la manipulation et du traitement des données pour que l'API fasse ce qu'elle est censée faire.

Dans la couche logique les vérifications sont faites afin de s'assurer que les données entrant dans l'API sont correctes et respectent certaines règles. Les règles vérifiées sont les types, ordre d'exécutions et dans les cas où tout se passe correctement l'envoi de statuts OK.

Mon API étant créé en Spring Boot inclue un serveur Tomcat intégré avec l'utilisation de la dépendance suivante :

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-tomcat</artifactId>
  <scope>provided</scope>
</dependency>
```

Le serveur Tomcat intégré est super pratique pendant le développement. Il permet de redémarrer rapidement l'application. Quand je faisais des changements dans mon code, je pouvais voir les résultats tout de suite en redémarrant le serveur Tomcat intégré. Cela m'a fait gagner beaucoup de temps pendant le développement, car je pouvais itérer plus rapidement et travailler de manière plus efficace.

Entre autres j'utilise le spring boot data jpa, spring boot security et une dépendance très utile lors la création des classes Lombok.

Spring boot data jpa facilite l'utilisation de la persistance des données dans les applications Spring Boot. Elle intègre le module JPA (Java Persistence API) pour simplifier la configuration et également l'ORM (object relational mapper)

Hibernate. Hibernate se charge de la conversion des objets Java en instructions SQL pour interagir avec la base de données. Il facilite également la création et la gestion des tables, des relations entre les entités, et des requêtes complexes. Mais le but principal d'Hibernate est de rendre l'application plus modulaire parce qu'en utilisant Hibernate l'application ne dépend plus d'un système de gestion de base de données spécifique et peut être changé facilement.

La dépendance Spring boot security offre des fonctionnalités de sécurité prêtes à l'emploi pour les applications Spring Boot. Elle facilite l'implémentation de l'authentification, de l'autorisation et de la protection des ressources de l'application. En ajoutant cette dépendance avec la dépendance de JWT (json web token) j'ai pu configurer l'authentification de mon application en utilisant des jetons d'authentification.

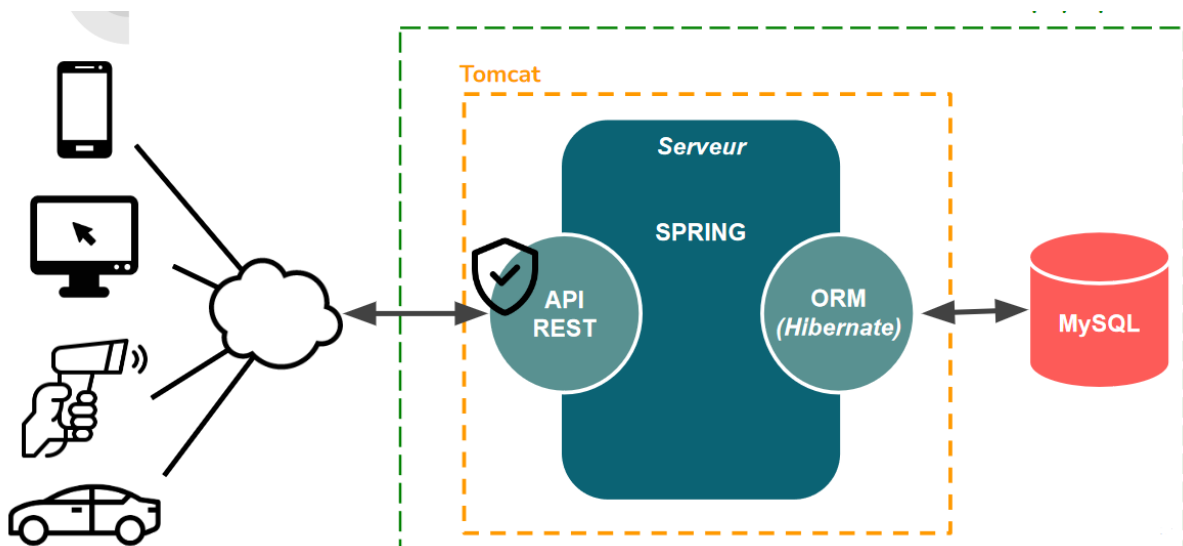


Figure 6. Illustration de fonctionnement d'une API REST

5.2 Le niveau de présentation :

Dans ce palier se trouvent les interfaces utilisateur de mon application, notamment mon frontend avec Angular et éventuellement l'application mobile avec React Native.

Le but principal de la couche de présentation est de fournir une interface utilisateur compréhensible afin que l'utilisateur final puisse s'y retrouver.

Comme mentionné plus haut j'ai développé mon application web avec Angular. J'ai utilisé Angular Matériel, une bibliothèque d'Angular créée par une team de chez Google afin de donner le « feel » ou le sentiment android aux composants des applications créées avec cette librairie.

L'interface utilisateur de l'application Locparc est conçue de manière à offrir une expérience utilisateur fluide et conviviale. Pour ce faire des composants angular ont été indispensables.

J'ai également combiné angular matériel avec un thème Bootswatch. Bootswatch est un projet open source qui fournit des thèmes créés avec Bootstrap (un framework CSS).

5.3 *Le niveau de persistance de données :*

Pour la couche de persistance de données j'ai utilisé MySQL comme système de gestion de base de données. J'ai également configuré phpMyAdmin afin d'avoir une meilleure visualisation des tables dans ma base de données et aussi afin de tester mes requêtes également.

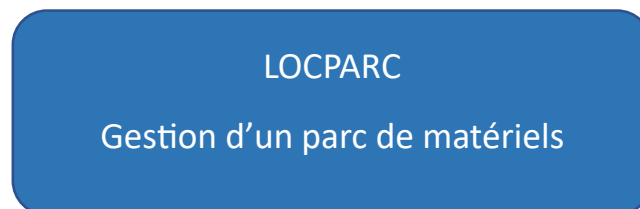
J'ai utilisé le langage SQL pour écrire des requêtes dans l'interface utilisateur phpMyAdmin, et HQL (Hibernate Query Language) dans le backend que j'ai testé avec Postman (une application faite dans le but d'aider les développeurs dans le développement de leurs APIs) et exécuté à l'aide de bibliothèques de connexion JDBC (Java Data Base Connectivity) dans le fichier pom dans le backend de Locparc.

7. Réalisations

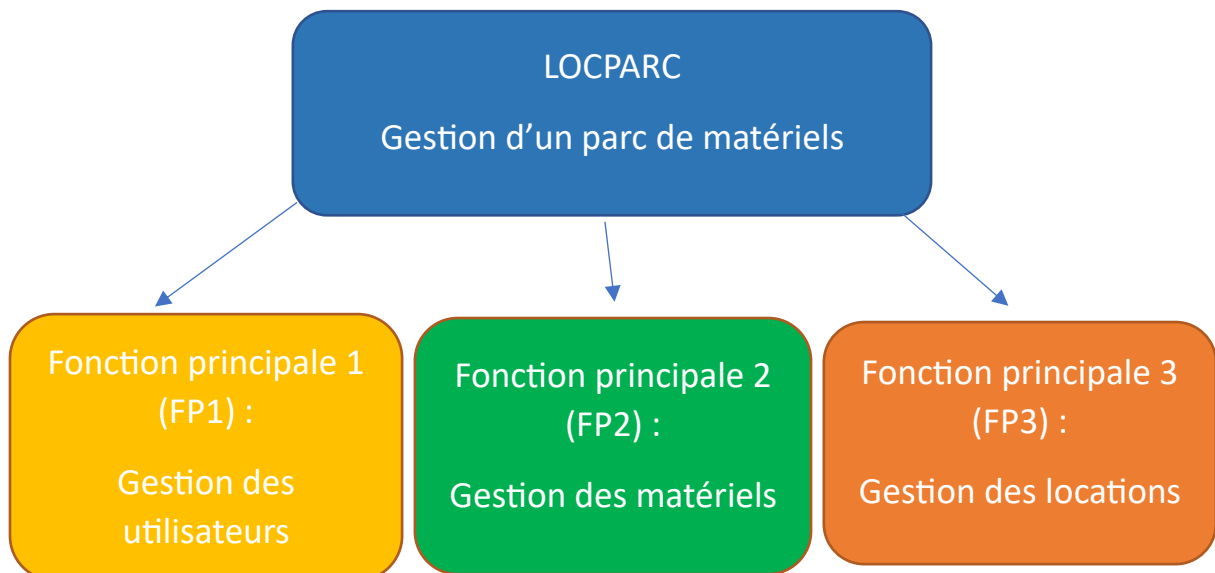
1. Conceptualisation

Méthode de conception fonctionnelle descendante :

J'ai utilisé la méthode de conception fonctionnelle descendante ou la méthode « top down » afin de démarrer mon projet avec ce que j'ai cru, plus de facilité. Mon choix était simple : commencer par la définition de la fonctionnalité principale de l'application. Dans mon cas, après avoir consulté avec mes collaborateurs, c'était la gestion d'un parc de matériels :



Ce qui a donné suite aux autres fonctionnalités de Locparc :



Auteur des figures : Gilbert Gérald, l'un des collaborateurs de Locparc

Pendant la conception de mon projet j'étais en train de lire un livre titré : Conception et programmation orienté objet, écrit par Bertrand Meyer le créateur du langage orienté objet Eiffel.

Dans son livre monsieur Meyer fait une évaluation de la méthode descendante fonctionnelle qui n'est pas d'un avis favorable : « En développant un système de manière descendante vous gagnez en facilité à court terme ce que vous perdez par une inflexibilité à long terme ».

Etant déjà un peu avancé dans la conception de Locparc, cette phrase ne m'a pas fait plaisir (c'est le moins que l'on puisse dire). Mais j'ai réalisé si j'avais continué à pratiquer la méthode descendante, je me serais renfermé dans mes fonctionnalités au bout d'un moment parce que cette méthode favorise très fortement la spécificité d'un logiciel, ce qui nuit à l'extensibilité de mon application. Et l'extensibilité étant l'une des qualités principales d'un produit, est très importante.

Même si l'application Locparc n'est pas d'une taille importante, l'un des critères imposés par le futur gestionnaire est que l'application possède une fondation capable de soutenir plus de fonctionnalités que ce qu'il y a dans le cahier des charges et la méthode top down n'est juste pas conçue pour.

J'ai du tout recommencer et cette fois ci j'ai utilisé d'autres méthodes de conception comme Merise et plus tard le langage de modélisation UML (universal modeling language).

LA DEVISE OBJET

« Ne demandez pas en premier ce que fait le système : demandez à qui il le fait ! » - Bertrand Meyer

1.1 Merise

a. Analyse des besoins :

A partir du cahier des charges et en ayant la devise objet en tête, j'ai commencé à déduire les acteurs principaux. Le premier acteur principal est bien évidemment l'utilisateur « User », j'ai ensuite déduit le deuxième : « Item » ou matériel ; le troisième est la commande « Order ». Je vais ensuite procéder par lister tous les acteurs (ou entités) restants de mon analyse : « Address », « Country », « Role » (les rôles des utilisateurs, tels que admin, user...), « Category », « SubCategory », « Manufacturer », « Model », « Licence », « Maintenance », « Repairer », « Payment », « PaymentType », « Request » et finalement « Notification » (qui servira pour des futures fonctionnalités, comme des notifications message in app, etc.).

b. Modélisation conceptuelle :

J'ai ensuite créé un diagramme entité-association en utilisant l'application web LucidChart dans laquelle j'ai créé des entités à partir de mes acteurs principaux. Un tableau a été attribué à chaque entité.

J'ai procédé ensuite par la création des associations entre mes entités. Pour ce processus j'avais besoin des avis de mes collaborateurs et des spécifications supplémentaires du client sur les fonctionnalités attendues et choix de conception, comme par exemple le choix pour le stockage des photos du matériel, vont-elles être stockées dans une base de données à part ? où sur un drive ? ou encore un espace de stockage comme Dropbox ? Nous nous sommes décidés de stocker url de l'image du matériel, comme ça l'image peut être stocké sur un serveur ou drive dédié à ça spécialement.

Une fois le consensus établi, j'ai continué la création des associations (avec une particularité que je vais expliquer). Pour ce faire j'avais utilisé la notation d'UML crow's foot (le pied de corbeau) parce que je la trouve plus lisible et parce qu'elle a un meilleur rendu dans l'application LucidChart et avec la notation j'ai également changé le sens de lecture des cardinalités ce qui peut être considéré comme une méthode complètement à part mais j'ai décidé de la regrouper avec Merise juste à cause de ma préférence pour le sens des cardinalités et la notation.

Je vais procéder par lister des relations entre mes entités :

Un User peut avoir zéro ou une Address, et une Address peut contenir zéro ou plusieurs User (dans le cas d'une colocation ou encore une entreprise par exemple).

Un User peut avoir un et un seul Role (admin, utilisateur ou loueur) et un Role pour avoir zéro ou plusieurs User.

Un User peut effectuer zéro ou plusieurs Request (demandes de location), mais un Request concerne un et un seul User.

Un User peut avoir zéro ou plusieurs Order (commande), mais un Order concerne un et un seul User et avec ça nous avons terminés avec l'entité User.

Un Item peut être dans zéro ou plusieurs Order et un Order peut avoir un ou plusieurs Item ce qu'il fait que nous avons une relation dite many to many. A chaque fois qu'une relation pareille se forme, un tableau supplémentaire entre les deux entités est requis afin d'éviter la répétition et surtout de garantir la cohérence des données. La convention pour nommer les tableaux créés à partir des relations many to many est de leur donner les noms des deux entités, ce qui dans mon cas donne Order_Items (ou une ligne de commande).

Ensuite un Item peut être de zéro ou d'un Model, un Model peut avoir zéro ou plusieurs Item. Un Model a un et un seul Manufacturer (fabricant) et un Manufacturer peut avoir un ou plusieurs Model.

Un Item peut avoir une et une seule SubCategory, une SubCategory peut contenir zéro ou plusieurs Item. Une SubCategory a forcément une Category et une Category a une ou plusieurs SubCategory.

Un Item peut avoir zéro ou plusieurs Licence et une Licence peut concerner plusieurs Item, encore une relation many to many. Une table Item_Licences est créé, qui a des relations many to one vers l'Item et vers Licence.

Un Item peut être envoyé en Maintenance zéro ou plusieurs fois, une Maintenance ne concerne qu'un Item. Une Maintenance peut être effectué par un Repairer (réparateur), un Repairer peut effectuer plusieurs Maintenances.

Passons maintenant au dernier acteur principal, l'Order (commande).

Un Order peut contenir plusieurs Items, un Item peut se trouver dans plusieurs Order. Order_Items a été créé pour ça, du coup Order_Items correspond à une ligne de commande. Un exemple éclaircira une éventuelle confusion : une commande qui contient trois équipements à été validé, ce qui correspond du côté de la base de données comme trois lignes dans la table Order_Items, où une clé primaire d'un Item correspond à une ligne de commande. Il y aura trois pour cette commande fictive et la clé primaire de l'Order va se répéter autant de fois qu'il y a d'Item dans la commande.

Un Order a un Request et un Request correspond à un Order. C'est une relation One to One.

Et dernièrement un Order peut avoir zéro ou plusieurs Payment et un Payment correspond à un Order. Un Payment a un et un seul PaymentType mais un PaymentType peut avoir zéro ou plusieurs Payment.

Il reste la table Notifications qui est prévue de servir pour un usage futur.

Avec ça j'ai terminé l'explication des relation des entités de l'application Locparc.

c. Modélisation logique :

J'ai procédé par créer des diagrammes des flux de données. Voici une version d'un des diagrammes de flux de données un peu modifié « à ma sauce » .

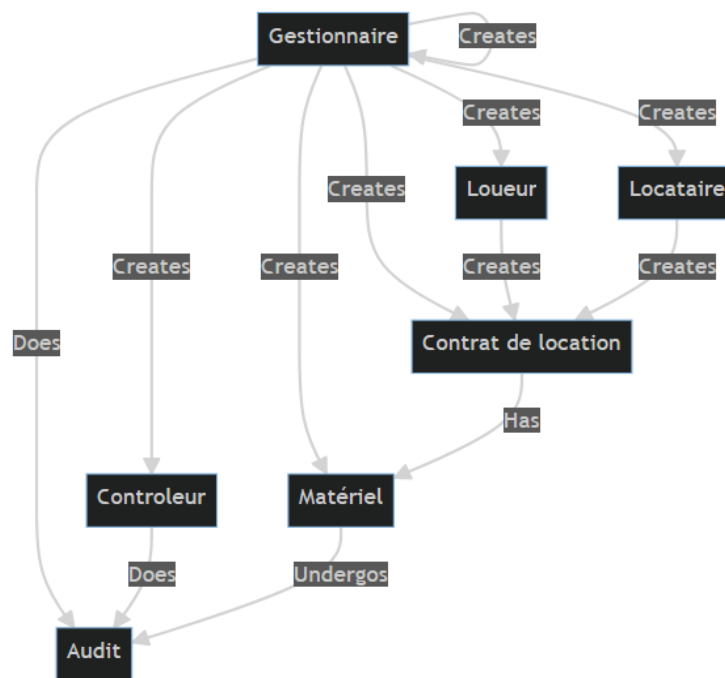


Figure 1. DFD (légèrement modifié) de Locparc

Ce diagramme (figure 1) représente l'une des premières itérations de la conception de Locparc et n'est plus à jour, mais il sera utile quand même à des fins d'explication.

Ce qu'on peut déduire du diagramme est qu'un gestionnaire peut créer d'autres gestionnaires, ainsi que d'autres loueurs et d'autres locataires (les explications des loueurs et locataires se trouvent dans la partie spécifications fonctionnelles du projet).

Des gestionnaires, loueurs ainsi que des locataires peuvent créer des contrats de location, ce qui correspond actuellement à une commande. Le contrat de location contient une liste d'équipements qui vont être loués/prêtés. Ces équipements subissent une vérification par un Contrôleur. Nous avons voulu créer une entité spécialement dédiée aux contrôles des matériels afin d'éviter les éventuels futurs problèmes d'éthique où un gestionnaire loue et contrôle un équipement. Mais ça a introduit une petite couche de complexité supplémentaire qui n'était vraiment pas nécessaire, et pouvait être considéré comme une nuisance plus qu'autre chose, dans les petites structures où à cause d'un manque de personnel un gestionnaire doit être le contrôleur également.

J'ai ensuite créé des diagrammes de flux de traitement, modifiés également (parce que je n'ai pas explicitement indiqué un début et une fin). Ce diagramme n'est pas à jour non plus pour des raisons que j'explique plus bas.

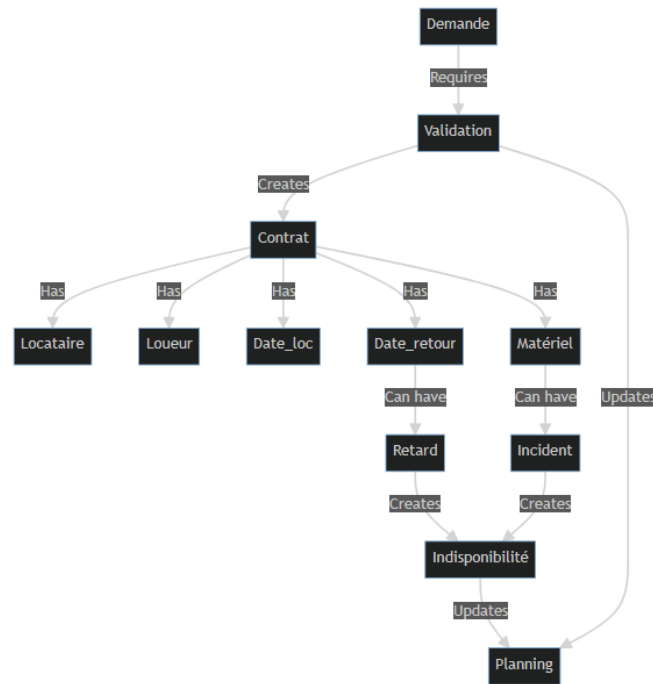


Figure 2. Diagramme de flux de traitement (modifié)

Depuis la figure 2 nous pouvons constater qu'une demande requière une validation. Peu importe si la demande est validée, un contrat (une commande actuellement) est généré. Le contrat contient un locataire, un loueur, une date de début, une date de fin, une liste de matériels, et ce qui n'est pas sur l'image : un statut, une adresse et un éventuel paiement. Les matériels peuvent ensuite subir un incident où ils se cassent ou autre chose, ce qui entrainera une indisponibilité dans le planning de Locparc. Un retard de retour des matériels peut également entrainer une modification du planning.

Ces modifications du planning se sont avérées ingérables à cause d'énormément de variables qui peuvent modifier le planning et qui peuvent affecter les réservations des autres utilisateurs. Après beaucoup de réflexion sur le sujet, et une consultation avec le client nous avons décidés de partir sur le principe qu'un équipement ne sera pas considéré comme disponible tant qu'il n'est pas rendu et vérifié par le gestionnaire. Ça nous a considérablement simplifié la gestion du planning.

d. Modélisation physique :

J'ai ensuite ajouté des attributs aux entités précédemment créées ainsi que leurs types de données.

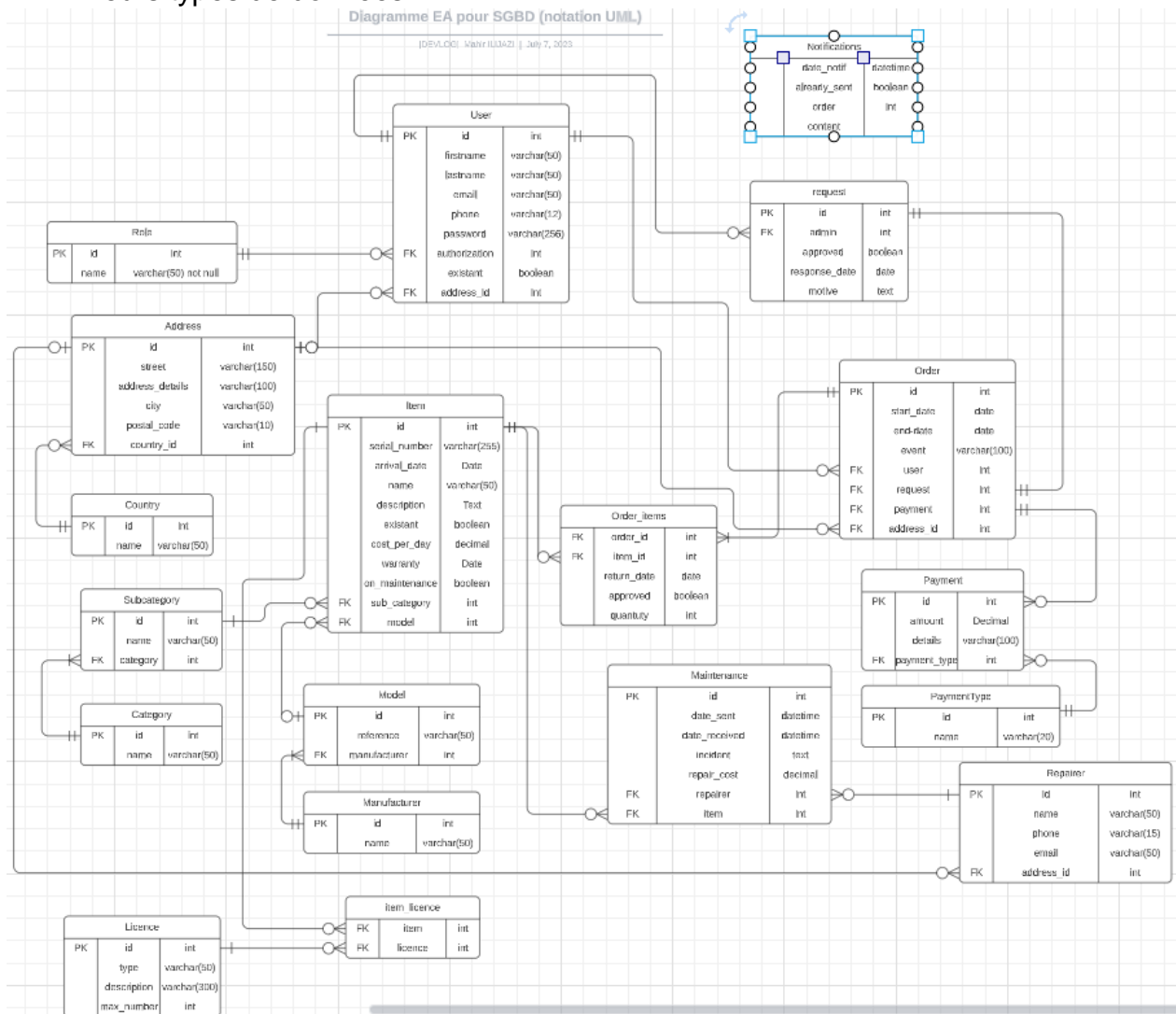


Figure 3. Diagramme Entité-Association avec une notation UML (crows foot) et la lecture inversée des cardinalités



Cette étape (modélisation physique) peut être utilisée pour la génération d'un script SQL afin de créer la base de données. Dans ce cas il faut utiliser la bonne notation des types de données des attributs afin de convenir au SGBDR souhaité. Même si j'ai créé mon SQL en utilisant Spring Data JPA j'ai quand même utilisé les types de données MySQL dans mon diagramme afin d'améliorer la lisibilité et permettre une compréhension plus facile du diagramme.

J'ai mis le mot dessiner entre guillemets car nous ne dessinons pas, ce que nous faisons c'est écrire du texte et regardons les diagrammes apparaître, avec les relations ! Et mon détail préféré : nous n'avons pas à arranger les classes pour bien voir les relations, mermaid le fait pour nous.

Cela dit voici mon diagramme de classes réalisé avec mermaid :

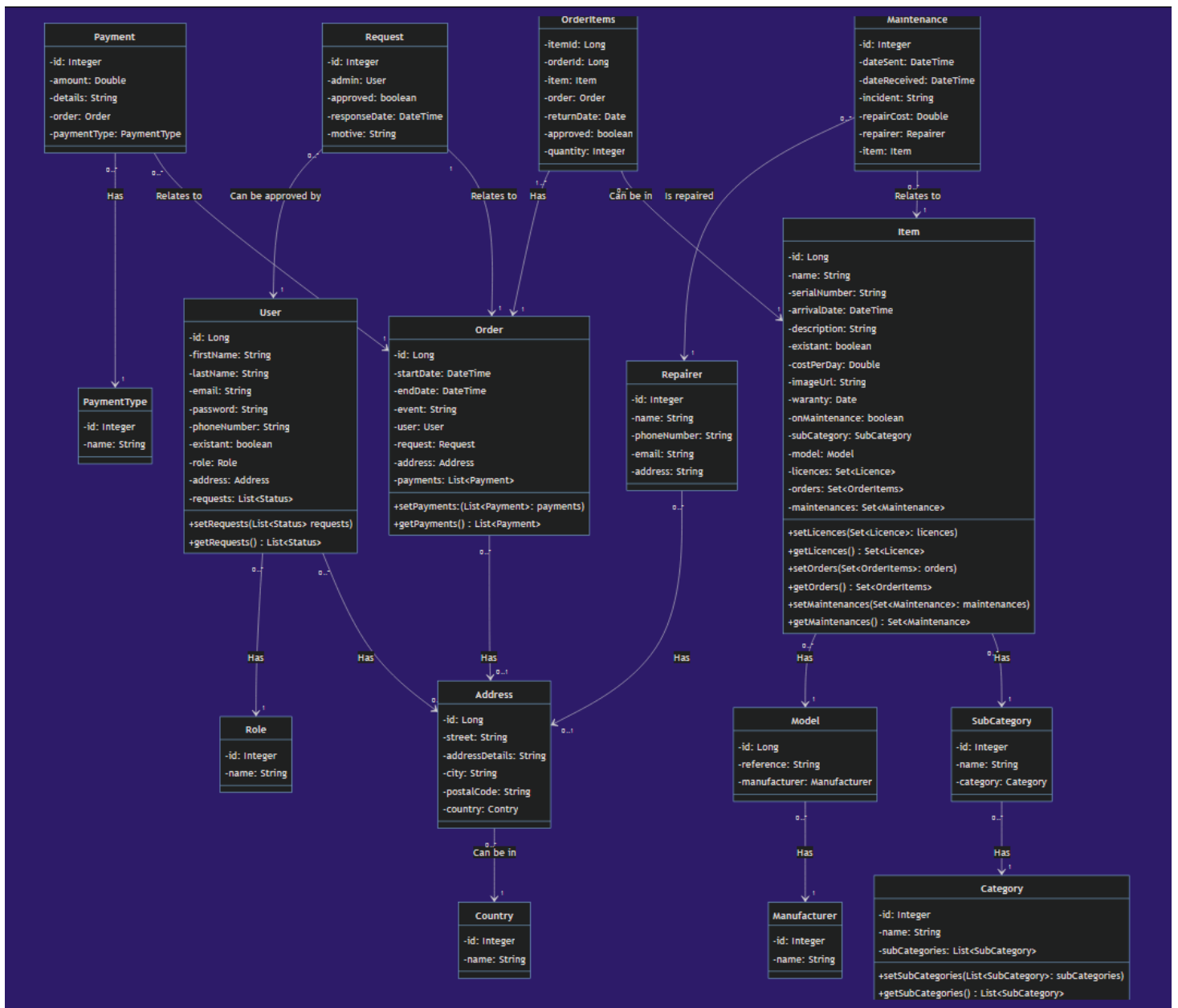


Figure 5. Diagramme de classes réalisé avec mermaid

Veuillez noter que les attributs de mes entités sont privés et uniquement accessibles via des getter et setter qui ne sont pas présent sur le diagramme parce que nous ne mettons que les getter et setter des collections (par exemple une **List**<Payment> pour une commande). Ce qui fait que les types de données des attributs ne sont plus les types adaptés à MySQL mais plutôt à Java.

2. L'API avec Spring Boot

2.1 Introduction

L'utilisation de Spring boot m'a permis de développer rapidement mon API grâce aux fonctionnalités offertes par le framework. Dans cette partie je détaille les réalisations clés qui ont marqué mon projet. Je commence par l'initialisation de l'API, je présente ensuite la création d'une entité avec Spring, puis la création d'un contrôleur qui permet la communication entre le frontend et le backend. Je présente également une requête HQL personnalisé. Et pour terminer j'en parle de Spring Security grâce à laquelle je gère la sécurité de Locparc, comme l'accès à l'application, les autorisations des utilisateurs, etc.

2.2 Initialisation de l'API

Une fois la conception terminée j'ai commencé par initialiser un projet vierge sur le site de Spring Initializr start.spring.io. Sur la page de Spring Initializr nous pouvons sélectionner le build tool (un outil de construction de projet) que nous voulons, pour moi c'était Maven car il était le plus utilisé pour les projets Java et que j'avais une petite connaissance de ce build tool. Grâce à Maven

Ensuite nous avons le choix entre trois options de langage de programmation : JAVA, Kotlin et Groovy. J'ai naturellement choisi Java parce que j'ai une connaissance du langage.

Après ça nous devons faire le choix entre différentes versions de Spring Boot, j'ai choisi la version 3.0.6 qui venait à peine de sortir.

Ensuite c'est la metadata du projet comme le nom du package, description, nom du projet, le packaging en Jar (Java ARchive) ou War (Web ARchive). J'ai choisi le War parce que les fichiers war sont conçus spécifiquement pour être déployés sur le web alors que les Jar sont plus général comme pour des applications de ligne de commande ou encore les applications desktop. Puis j'ai choisi la version de Java, j'ai pris la version 17 parce que c'est la dernière version LTS (Long Term Support) qui veut dire qu'elle est maintenue dans le temps.

J'ai ensuite ajouté des dépendances :

1. Spring Web simplifie le développement d'applications web en fournissant un framework puissant pour gérer les requêtes HTTP, gérer les contrôleurs et générer des réponses.
2. Spring Security qui protège Locparc contre diverses vulnérabilités de sécurité. Par exemple, il aide à prévenir les attaques telles que le vol de session, les tentatives de connexion forcée et les attaques XSS (cross site

scripting) où l'attaquant envoie un script malveillant souvent dans la forme d'un lien cliquable afin de voler des informations.

3. Spring Data JPA un module de Spring qui simplifie l'accès aux bases de données relationnelles en fournissant des fonctionnalités de persistance des données.
4. Le MySQL driver qui permet à une application de se connecter et de communiquer avec une base de données MySQL.
5. Et Lombok qui est une bibliothèque Java qui réduit énormément le code répétitif des getter, setter et constructors.

The screenshot shows the Spring Initializr web interface. On the left, under 'Project', 'Maven' is selected. Under 'Language', 'Java' is selected. Under 'Spring Boot', '3.0.8' is selected. The 'Project Metadata' section shows: Group: 'com.mahir', Artifact: 'locparc', Name: 'locparc', Description: 'Une application de location/prêt de matériel', Package name: 'com.mahir.locparc', Packaging: 'War' (selected), and Java version: '17' (selected). On the right, the 'Dependencies' section lists: 'Lombok' (Developer Tools), 'Spring Web' (Web), 'Spring Security' (Security), 'Spring Data JPA' (SQL), and 'MySQL Driver' (SQL). Each dependency has a brief description.

Figure 6. Initialisation du projet locparc sur le site start.spring.io

a. Création d'une entité

J'ai créé ma base de données avec l'aide de JPA (Java Persistence API) qui, avec l'ORM (Object Relational Mapper) Hibernate permettent la persistance de données à une base de données relationnelle.

L'annotation `@Entity` signale au framework qu'il s'agit d'une entité qui s'appelle User qui doit être géré par le système de persistance, dans ce cas Hibernate

```
1 package com.mahir.locparc.model;
2
3 > import ...
4
5 @Entity // L'annotation qui communique à Spring Boot qu'il s'agit d'une entité
6 @Getter // Lombok, la dépendance Maven nous génère les getters
7 @Setter // ainsi que les setters
8 public class User {
9     @Id // L'entité User a besoin d'un id pour être considéré comme telle
10    @GeneratedValue(strategy = GenerationType.IDENTITY) // Ça nous permet d'avoir le champ id autoincrémenté
11    private Long id;
12
13    @Column(nullable = false, length = 50)
14    @JsonView({UserView.class})
15    private String firstName;
16
17    @Column(nullable = false, length = 50)
18    @JsonView({UserView.class})
19    private String lastName;
20
21    @Column(nullable = false, length = 50)
22    @JsonView({OrderView.class, UserView.class})
23    private String email;
24
25    @Column(length = 15)
26    @JsonView({UserView.class})
27    private String phone;
28
29    @Column(length = 256)
30    private String password;
31
32    private boolean active;
33
34    @ManyToOne
35    @JsonView({UserView.class})
36    private Address address;
37
38    @Transient
39    private String firstPassword;
40}
```

Figure 7. Entité User créé avec JPA

Les annotations `@Getter` et `@Setter` sont des annotations Lombok qui est une bibliothèque Java permettant la génération de méthodes getter et setters entre autres.

J'ai ensuite l'annotation `@Id` afin d'indiquer à Hibernate qu'il s'agit de la clé primaire de l'entité `User`. `@GeneratedValue (strategy = GenerationType.IDENTITY)` nous permet d'avoir le champ `Id` auto incrémenté en précisant la stratégie d'auto incrémentation.

`@Column(nullable = false, length = 50)` Cette annotation est utilisée pour spécifier les propriétés de la colonne associées à la propriété `firstName`. Par exemple, `nullable = false` indique que la colonne ne peut pas contenir de valeurs nulles, autrement dit que les valeurs doivent être renseignés, et `length = 50` spécifie la longueur maximale de la colonne.

L'annotation `@JsonView({UserView.class})` indique à la bibliothèque Jackson quelles parties de l'objet `User` inclure lors de la conversion vers JSON grâce aux vues `UserView.class` (à ne pas confondre avec les vues de l'architecture MVC) créés au préalable.

`@ManyToOne`, l'annotation qui indique une relation many to one qui signifie que plusieurs utilisateurs peuvent être associées à une adresse ou un rôle qui n'est pas présent dans la figure 7.

L'annotation `@Transient` sur le champ `firstPassword` nous offre la possibilité de ne pas persister une donnée dans la base de données.

Ce qui n'est pas présent sur la figure 7 également est l'annotation suivante pour la liste de requêtes (`List<Request>`) que l'utilisateur peut avoir effectué :

`@OneToMany(mappedBy = "admin", cascade = CascadeType.ALL, orphanRemoval = true)`

La relation one to many signifie qu'un utilisateur peut faire plusieurs demandes. Et les paramètres de l'annotation spécifient que :

- `mappedBy = « admin »` veut dire que la propriété `admin` de l'entité `Request` est responsable pour la relation (les relations one to many ne sont pas « importantes » dans la définition de types de relation)
- `cascade = CascadeType.ALL` indique que toutes les opérations de modification : `Create`, `Update`, `Delete` (ajout, mise à jour, suppression) sur la liste `requests` doivent être propagés sur les objets associés.
- `orphanRemoval = true` indique qu'il n'y aura pas de données orphelines où une requête n'est associée avec aucun utilisateur.

Toutes ces annotations m'ont servies à spécifier comment les données doivent être stockées dans la base de données, comment les entités sont liées entre elles et quelles propriétés doivent être incluses lors de la conversion en format JSON.

2.4 Création d'un contrôleur

```
24  mahir *  
25  @CrossOrigin  
26  @RestController  
27  @RequestMapping(value = "/api")  
28  public class UserController {  
29      13 usages  
30      private final UserDao userDao;  
31      6 usages  
32      private final PasswordEncoder passwordEncoder;  
33      2 usages  
34      private final JwtService jwtService;  
35      5 usages  
36      private final PasswordGenerator passwordGenerator;  
37  
38      mahir  
39      public UserController(UserDao userDao,  
40                          PasswordEncoder passwordEncoder,  
41                          JwtService jwtService,  
42                          PasswordGenerator passwordGenerator) {  
43          this.userDao = userDao;  
44          this.passwordEncoder = passwordEncoder;  
45          this.jwtService = jwtService;  
46          this.passwordGenerator = passwordGenerator;  
47      }  
48  
49      mahir  
50      @GetMapping("/admin/users")  
51      @JsonView({UserView.class})  
52      public List<User> getAll() { return userDao.findAll(); }  
53  
54      mahir *  
55      @GetMapping("/admin/users/{id}")  
56      public ResponseEntity<User> getUser(@PathVariable Long id) {  
57          Optional<User> optionalUser = userDao.findById(id);  
58  
59          return optionalUser.map(user ->  
60              new ResponseEntity<>(user, HttpStatus.OK)).orElseThrow(() ->  
61                  new ResourceNotFoundException("Utilisateur avec l'identifiant "  
62                      + id + " n'existe pas dans la base de données"));  
63      }  
64  }
```

Figure 8. Le contrôleur UserController Spring Boot

Afin de créer un contrôleur en Spring Boot nous devons préciser au framework qu'il s'agit d'un contrôleur REST (Representational State Transfer) avec l'annotation `@RestController`. `@RestController` est une annotation qui combine deux annotations, `@Controller` et `@RequestBody`. Il est intéressant d'utiliser cette annotation parce qu'elle communique à Spring Boot qu'il s'agit d'une classe qui sera responsable de

requêtes http et automatiquement transforme les valeurs de retour des méthodes dans la classe en Json ou XML (Json dans mon cas).

L'annotation présente dans la figure 8, `@RequestMapping`, sert à indiquer l'url de base pour toutes les méthodes de la classe. Par exemple la méthode `getAll()` retourne une liste d'utilisateurs si l'endpoint `/admin/users` est appelé, mais elle sera seulement accessible si l'utilisateur ajoute `/api` avant.

UserDao est une dépendance que j'ai injecté via le constructeur et est un Data Acces Object créé afin d'effectuer les opérations CRUD (Create, Read, Update, Delete) sur les entités User. L'utilisation d'un DAO permet également d'atteindre un couplage dit « faible » où par exemple : changer l'implémentation d'une méthode dans le DAO n'affecte pas les composants qui l'utilisent.

J'ai ensuite une autre injection de dépendance qui est le PasswordEncoder. C'est une classe qui implémente l'interface PasswordEncoder de Spring Security et est utilisé afin de hacher les mots de passe pour les préparer pour le stockage en base de données, car il ne faut jamais stocker les mots de passe en clair dans une base de données.

J'ai également une instance de JwtService qui me sert à vérifier si un utilisateur modifie bien son propre profil et pas celui des autres utilisateurs. Pour ce faire je demande un id de l'utilisateur et son token d'authentification ou le JWT. Ensuite je vérifie si le nom d'utilisateur présent dans le token correspond bien au nom d'utilisateur obtenu à partir de la méthode `getUserById(id)` dans le userDao. Si c'est bien le cas l'utilisateur peut continuer vers la page destinée à la modification de profile, sinon il recevra le statut http 403 (Forbidden) et l'accès à la page lui sera interdit.

J'ai ensuite un passwordGenerator qui sert à des fins de sécurité au moment de la création des nouveaux utilisateurs par le gestionnaire. Un mot de passe aléatoirement généré est attribué à l'utilisateur venant d'être créé et est lui envoyé sur son adresse mail personnelle. Dans cette manière seul lui sera capable de voir son mot de passe.

Je ne vais expliquer chaque route de mon contrôleur mais je vais par contre prendre un exemple pour chaque annotation de mapping.

L'annotation `@GetMapping` indique au framework qu'il s'agit d'une requête http GET. Dans l'exemple `@GetMapping("/admin/users/{id}")` présent sur la figure 8 je retourne un `ResponseEntity` qui est une classe du framework Spring et qui sert d'une réponse http. J'ai ensuite `public ResponseEntity<User> getUser(@PathVariable Long id)` la signature de ma méthode qui prend en paramètre une `@PathVariable` qui est une annotation qui représente la variable id qui se trouve entre crochet dans l'url.

Ensuite je procède par un appel à une méthode dans mon userDao (`userDao.findById(id)`) qui sert à trouver un utilisateur à partir de son identifiant et qui renvoie un `Optional`, une classe Java qui sert à indiquer si une valeur est présente ou absente. L'avantage avec la classe `Optional` est que je n'ai pas à vérifier des valeurs nullables. Si l'utilisateur est présent je l'envoie avec un statut http 200 OK, sinon

j'envoie une exception `ResourceNotFoundException` (qui prend en paramètre un message personnalisé indiquant que l'utilisateur avec l'identifiant fourni n'existe pas dans la base de données) que j'ai créé auparavant pour gérer des ressources pas trouvés.

L'annotation `@PostMapping` indique au framework qu'il s'agit d'une requête http POST. L'exemple pour l'annotation `@PostMapping` n'est pas présent sur la figure 8 à cause d'un manque de place. Je vais commencer par l'explication de l'annotation `@Transactional`. Cette annotation est utilisée dans les cas des opérations de modification (mise à jour, création ou suppression) afin de retourner au point de départ dans le cas d'un problème (panne de courant, données incorrectes, etc.).

Ensuite, comme nous pouvons voir dans la signature de la méthode :

```
public ResponseEntity<List<User>> createUsers(@RequestBody
User[] users)
```

Elle prend en paramètre un tableau d'utilisateurs qui est bind-é avec l'aide de l'annotation `@RequestBody` (ou corps de la requête). Ce tableau est ensuite traversé (s'il n'est pas vide) et pour chaque utilisateur dans le tableau : l'utilisateur est mis en tant qu'actif, un mot de passe est généré aléatoirement grâce à la méthode `generatePassayPassword(String password)`, le champ `firstPassword` de l'utilisateur est défini avec le mot de passe généré aléatoirement, le mot de passe est ensuite encodé avec le `passwordEncoder` et le rôle de l'utilisateur est défini, pour ce endpoint le rôle est celui d'un simple utilisateur (`ROLE_USER`). L'utilisateur est ensuite persisté dans la base de données et un email lui est envoyé le mot de passe qui vient d'être créé.

L'annotation `@PutMapping` indique à Spring Boot qu'il s'agit d'une requête http PUT qui est utilisé pour les requêtes de mise de jour. L'exemple d'une requête PUT n'est également pas présent sur la figure 8, du coup je vais le détailler comme les précédents.

L'endpoint est aussi doté d'une annotation `@Transactional` car il s'agit d'une opération de modification et l'url de l'endpoint est le : `@PutMapping("/edit-profile/{id}")`

La signature de la méthode est la suivante :

```
public ResponseEntity<User> editUser(
    @PathVariable Long id,
    @RequestBody User user,
    @RequestHeader("Authorization") String jwt
)
```

Elle prend trois paramètres : un id de type Long (car les longs vont plus loin que 2^{32} qui est environ 2.15 milliards de chiffres entiers, ils vont jusqu'au 2^{64} et que la convention recommande ainsi), un `@RequestBody` avec les nouvelles données pour l'utilisateur et un `@RequestHeader` qui est une annotation qui demande l'entête d'Authorization qui est le token JWT. Et elle envoie une réponse en forme d'une `ResponseEntity`.

L'id est demandé afin d'être comparé avec l'id obtenu après une recherche d'un utilisateur par son email, grâce à la méthode `findUserByEmail(String email)`. La comparaison se fait parce que nous ne voulons pas que des utilisateurs modifient les données des autres utilisateurs. Si c'est le cas je permet la modification, sinon une `ResponseEntity` est retourné indiquant le statut `http 403 Forbidden`.

La dernière annotation de mapping que je vais expliquer pour le `UserController` est l'annotation `@DeleteMapping`. L'url de l'annotation est `/admin/users/{id}` indiquant que seulement des gestionnaires (ou autrement appelés administrateurs) peuvent supprimer des utilisateurs. Ceci est rendu possible grâce à la configuration de sécurité que nous allons voir en détails plus loin.

Pour l'instant voici la signature de la méthode `deleteUser` :

```
public ResponseEntity<User> deleteUser(@PathVariable Long id)
```

Il s'agit d'une opération de modification du coup elle contient l'annotation `@Transactional`.

Afin d'effectuer une suppression d'un utilisateur nous devons d'abord s'assurer qu'il existe dans notre base de données. `UserDao` vient à notre secours avec la méthode `findById(Long id)`. Comme d'habitude, un `Optional` est envoyé qui nous permet de vérifier en toute sécurité si notre utilisateur est bel et bien présent. Si oui, il sera supprimé de la base de données et une `ResponseEntity` va être envoyé avec le statut `http 200 OK`, sinon ça sera un statut `http 400 BAD_REQUEST`. Et avec ça j'ai couvert ce qu'il faut pour créer un contrôleur rest avec Spring Boot même si toutes les endpoints n'ont pas été abordées.

2.5 Spring Security

Je vais procéder par détailler mon implémentation de Spring Security.

J'ai tout d'abord créé une classe que j'ai nommé `SecurityConfig`. Cette classe porte deux annotations : `@Configuration` et `@EnableWebSecurity`.

L'annotation `@Configuration` indique au framework qu'il s'agit d'une classe de configuration et peut être utilisé afin de définir des `@Bean` dans la classe entre autres.

L'annotation `@EnableWebSecurity` permet de sécuriser Locparc en authentifiant les utilisateurs. Cette annotation permet également la restriction d'accès aux certaines pages de l'application, et elle protège également contre des failles de sécurité communes comme les failles XSS (Cross Site Scripting) en s'assurant que les données venant de l'utilisateur sont nettoyées avant d'être utilisés.

```

21  @Configuration
22  @EnableWebSecurity
23  public class SecurityConfig {
24
25      2 usages
26      private final JwtFilter jwtFilter;
27
28      mahir
29      public SecurityConfig(JwtFilter jwtFilter) { this.jwtFilter = jwtFilter; }
30
31      mahir
32      @Bean
33      public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
34
35          http
36              .cors().configurationSource(httpServletRequest -> {
37                  CorsConfiguration corsConfiguration = new CorsConfiguration();
38                  corsConfiguration.applyPermitDefaultValues();
39                  corsConfiguration.setAllowedMethods(Arrays.asList("GET", "POST", "DELETE", "PUT"));
40                  corsConfiguration.setAllowedHeaders(
41                      Arrays.asList("X-Requested-With", "Origin", "Content-Type",
42                          "Accept", "Authorization", "Access-Control-Allow-Origin"));
43                  return corsConfiguration;
44              }) CorsConfigurer<HttpSecurity>
45              .and() HttpSecurity
46              .csrf().disable()
47              .authorizeHttpRequests() AuthorizationManagerRequestMat...
48              .requestMatchers(@"/api/login").permitAll() // anybody can have access to the login page
49              .requestMatchers(@"/api/admin/**").hasRole("ADMIN") // the url that contains admin can only be accessed by admins
50              .requestMatchers(@"/api/lender/**").hasAnyRole("ADMIN", "LENDER")
51              .requestMatchers(@"/api/items/admin/**").hasRole("ADMIN")
52              .requestMatchers(@"/api/items/**").hasAnyRole("ADMIN", "LENDER", "USER")
53              // .requestMatchers("/api/items/**").hasRole("ADMIN")
54              .requestMatchers(@"/api/**").hasAnyRole("ADMIN", "LENDER", "USER")
55              .anyRequest().authenticated()
56              .and().exceptionHandling() ExceptionHandlingConfigurer<HttpSecurity>
57              .and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS) SessionManagementConfigurer<HttpSecurity>
58              .and().addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);
59
60          return http.build();
61      }

```

Figure 9. L'implémentation de la classe SecurityConfig

J'avais une ancienne implémentation de la classe SecurityConfig où elle héritait de la classe WebSecurityConfigurerAdapter qui est malheureusement à l'heure d'aujourd'hui déprécié dès que nous voulons utiliser les versions de Spring Boot plus récentes que la 2.7.0 et 5.7.1 pour la version de Spring Security.

J'ai donc écrit une nouvelle implémentation de la classe SecurityConfig qui est présente sur la figure 9.

J'injecte la classe JwtFilter par le biais du constructeur parce que les annotations @Autowired sont également dépréciés car l'équipe de Spring Boot souhaite encourager l'injection par constructeur.

La classe JwtFilter me sert à gérer l'authentification et l'autorisation basé sur les token JWT (Json Web Token, qui est une manière d'authentification dite « stateless » ou qui ne retient pas d'état).

Ensuite j'ai un @Bean nommé filterChain où j'ai défini les règles de sécurité ainsi que les filtres de l'application qui contribuent au flux de sécurité et qui retourne un SecurityFilterChain.

J'ai également une méthode dans mon `@Bean` `cors().configurationSource` qui configure la gestion de Cross Origin Resource Sharing ou CORS. Dans le block de cette méthode je spécifie quelles provenances (origines), méthodes (GET, POST, PUT et DELETE) et entêtes (« Authorization », "X-Requested-With", etc.) j'autorise.

La ligne qui suit est `csrf().disable()` et désactive la protection CSRF (Cross Site Request Forgery) parce que j'utilise une authentification stateless avec les tokens JWT et que la protection csrf concerne la génération des tokens uniques pour chaque *session* de l'utilisateur.

La ligne `authorizeHttpRequests()` marque le début des autorisations de requêtes http :

- `.requestMatchers("/api/login").permitAll()` → permet à tout le monde d'accéder à la page de connexion
- `.requestMatchers("/api/admin/**").hasRole("ADMIN")` → permet l'accès seulement aux admins.
- `.requestMatchers("/api/lender/**").hasAnyRole("ADMIN", "LENDER")` → permet l'accès aux administrateurs ainsi que loueur/prêteur
- `.requestMatchers("/api/items/**").hasAnyRole("ADMIN", "LENDER", "USER")` → permet l'accès aux utilisateurs, admins et loueur/prêteur
- `.anyRequest().authenticated()` → indique que tout autre requête doit être effectué par un utilisateur authentifié.

La prochaine ligne intéressante est la suivante `sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)` qui indique qu'il s'agit d'une authentification stateless pour laquelle Spring Boot ne doit pas créer de sessions du côté du serveur.

Ensuite j'ai la dernière méthode `.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class)` qui ajoute le filtre JWT (jwtFilter) avant le filtre par défaut de Spring boot (UsernamePasswordAuthenticationFilter). En faisant ainsi toutes les requêtes entrantes passent d'abord par le jwtFilter et sont vérifiés pour un token JWT valide, si l'utilisateur n'a pas de token, il ne sera pas authentifié pas le filtre et sera transféré vers le filtre par défaut pour la suite du traitement qui le passera ensuite au AuthenticationManager pour être authentifié.

J'ai ensuite un `@Bean` qui retourne un AuthenticationManager dans les fins d'authentification.

Et le dernier `@Bean` PasswordEncoder qui sert à encoder les mot de passe et à les vérifier.

2.6 Requête HQL

La raison pour laquelle j'ai fait le choix d'utiliser HQL même si je ne le maîtrise pas au même niveau que le SQL, est parce que je veux rendre mon code plus maintenable et extensible. La raison principale pour laquelle je n'ai pas employé le SQL classique est que je ne voulais pas être dépendant des SGBDR. Parce qu'il se peut que du jour au lendemain, je dois changer de SGBDR pour une raison quelconque et que je dois réécrire tout le SQL parce que tous les SGBDR ne sont pas identiques. Avec le HQL je n'aurai pas ce problème.

Voici la requête :

```
@Query("FROM Item as i " +
        "WHERE lower(i.name) LIKE concat('%', :name, '%') " +
        "AND i.subCategory.name IN :subCategories " +
        "AND i.existing = true " +
        "AND i.onMaintenance = false " +
        "AND i.id NOT IN (SELECT i2.id FROM Item as i2 " +
                        "JOIN i2.orders oi " +
                        "WHERE oi.returnDate IS NULL " +
                        "AND oi.approved = true ) ")

Optional<List<Item>> findAvailableItemsByNameAndSubCategories(
    @Param("name") String name,
    @Param("subCategories") List<String> subCategories
);
```

Cette requête retourne un Optional d'une Liste de matériels, et prend en paramètre le nom du matériel, ainsi qu'une liste de string de noms des sous catégories.

- FROM Item as i → prend l'entièreté des attributs de l'entité Item et lui attribue l'alias « i ».
- Where lower(i.name) LIKE concat('%', :name, '%') → cette ligne contient la condition WHERE qui est utilisé comme un filtre de données sur la base des conditions qui suivent : lower qui transforme l'attribut name de i en lettres minuscules, LIKE qui combiné avec concat vérifie pour tout les mots contenant le nom name, peu importe de l'endroit ou le paramètre name puisse être trouvé.
- AND i.subCategory.name IN :subCategories → Le mot clé AND indique l'ajout d'une condition à la clause WHERE, le mot clé IN demande si le nom de la sous catégorie du matériel correspond à un des nom dans le paramètre subCategories fourni.
- AND i.existing = true → cette ligne indique que la condition demandé est que le matériel doit être existant dans la base de données.
- AND i.onMaintenance = false → indique que le matériel ne doit pas être en réparation
- AND i.id NOT IN → que le matériel ne doit pas se trouver dans la sous requête suivante
- (SELECT i2.id FROM Item as i2 → je veux juste l'identifiant pour cette requête

- JOIN i2.orders oi → fait une jointure sur l'attribut orders de l'entité matériel et attribue lui l'alias oi (car les données d'oi sont de type OrderItems), même si il manque le mot clé AS l'attribution de l'alias s'effectue
- WHERE oi.returnDate IS NULL → où la date de retour n'est pas encore indiqué, signifiant que le matériel est encore en location
- AND oi.approved = true → si il a été approuvé en premier lieu.

3. Front end avec Angular

3.1 L'initialisation du projet Angular

J'ai fait le choix d'utiliser Angular pour des raisons expliquées précédemment.

Mais avant de commencer à développer mon front end j'ai du tout d'abord installer Node.js qui est une plateforme open-source basée sur le moteur JavaScript V8 de Chrome, permettant l'exécution de code JavaScript côté serveur. Node.js vient avec un manager de dépendances npm (node package manager) grâce au quel j'ai pu installer le command line interface Angular (Angular CLI, figure 4, chapitre 6) avec la commande `npm install -g @angular/cli`.

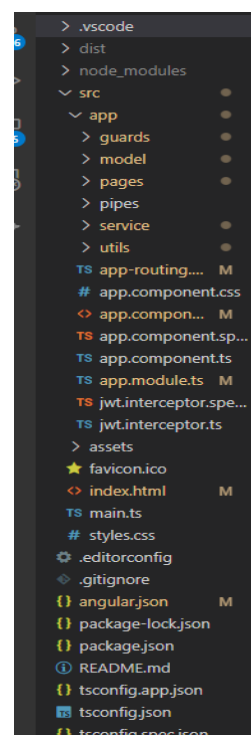
Une fois le cli installé j'ai commencé la création du front end en exécutant la commande `ng new locparc-angular` (parce que j'ai une repository sur GitHub qui porte le nom de Locparc et il s'agit de mon back end). Et avec ça j'avais déjà un début de projet avec une structure de base.

Voici à quoi peut ressembler une structure d'un projet Angular (précision : les dossiers dans le dossier app ont été créés plus tard et pas au moment de la création du projet)

Après avoir créé le projet je me suis rendu dans le répertoire du projet en tapant la commande suivante `cd locparc-angular` pour ensuite pouvoir lancer le serveur de développement sur `http://localhost:4200/` avec la commande `ng serve`.

J'ai plus tard changé la commande de lancement du serveur dans le fichier package.json de `ng serve` à `npm start`.

J'ai ensuite commencé le développement des composants que je vais expliquer dans la section suivante.



3.2 Création d'un component

Un composant (ou component) Angular est tout simplement une partie de l'application comme une page par exemple. Afin de créer un composant Angular, la procédure est très simple. Il suffit de taper la commande suivante dans le cli :

ng generate component nom-du-composant.

L'un des premiers composants que j'ai créé est le composant *équipement* (comme pour toute l'application j'ai utilisé une nomenclature anglaise). Il contient un fichier *équipement.component.ts* où se trouve la logique du composant, ensuite nous avons *équipement.component.spec.ts* qui est un fichier de test, également *équipement.component.html* ainsi qu'un fichier de style *équipement.component.css*.

```
@Component({
  selector: 'app-equipement',
  templateUrl: './équipement.component.html',
  styleUrls: ['./équipement.component.css']
})
export class EquipementComponent {}
```

Figure 11. Le composant EquipementComponent

Dans mon composant j'ai une annotation `@Component` qui est obligatoire et importé depuis `angular/core`. J'ai ensuite un objet qui possède trois propriétés. La première `selector` : étant le nom de la balise à écrire si nous voulons l'utiliser (`<app-equipement>`), la deuxième `templateUrl` désigne le nom du fichier de vue et la troisième `stylesUrls` est la liste des fichiers de styles. Le composant servira à créer des équipements.

Une fois que j'ai créé mon composant, il était temps de créer des fonctionnalités en se basant sur la maquette créée au préalable. Il me fallait un champ pour le nom du matériel, un champ pour les sous catégories et dans ce

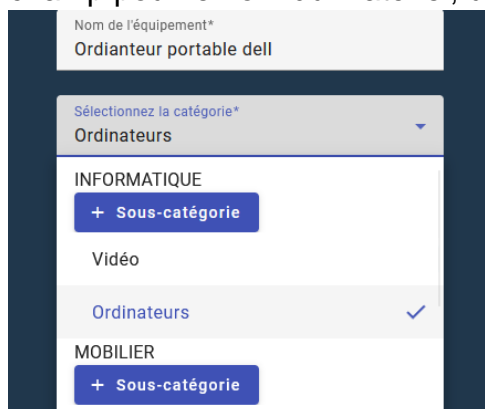


Figure 12. Champs de nom et sous catégories

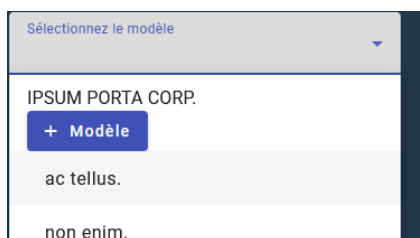


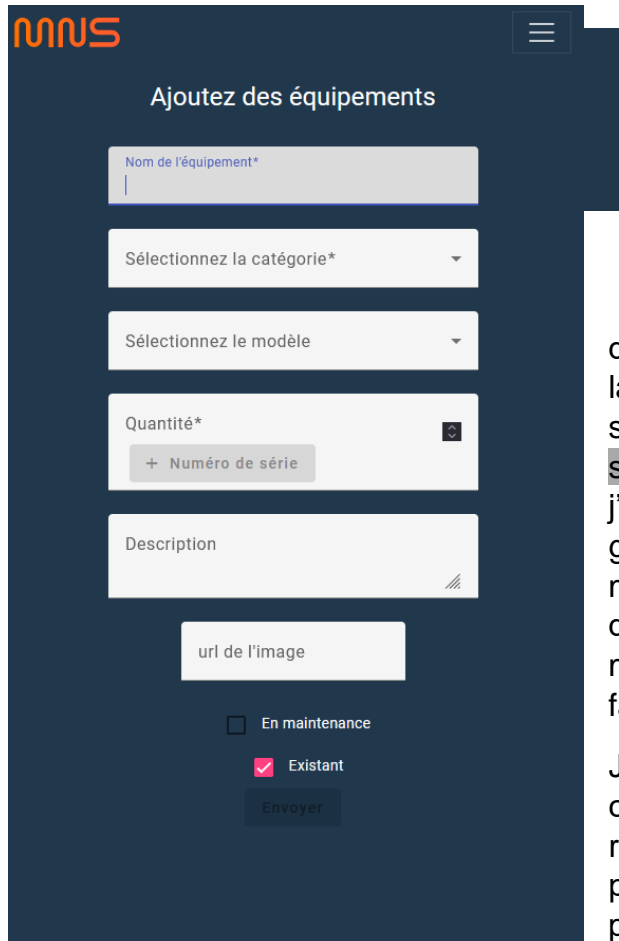
Figure 13. Champ de sélection de modèle (lorem ipsum)

champ nous pouvons ajouter des sous catégories selon besoin sans changer de page (le fait de ne pas être obligé de se déplacer vers la page dédiée aux catégories pour en ajouter une améliore l'expérience utilisateur), nous avons ensuite un champ dédié à la sélection de modèle qui nous pouvons sélectionner seulement si nous avons choisis une sous catégorie qui appartienne à la catégorie INFORMATIQUE. Nous pouvons ensuite indiquer la quantité dans le champ dédié, et en fonction de la quantité sélectionné et si l'équipement fait partie de la catégorie informatique, nous pouvons ajouter autant de numéros de série que la quantité indiquée auparavant grâce à un bouton qui devient bleu et cliquable à partir du moment où la quantité

indiquée est de 1 ou plus. Une fois le bouton cliqué le modal (créé avec Bootstrap) pour les champs de numéros de série apparaît.

Figures 14 et 15. Les champs de quantité ainsi que le modal des numéros de série

A partir de là nous pouvons ajouter l'équipement, ou lui ajouter un description un url d'une image, indiquer que l'équipement est en maintenance et dire si l'équipement existe.



The main form titled 'Ajoutez des équipements' (Add equipment) is displayed on a dark blue background. It contains several input fields: 'Nom de l'équipement*' (Equipment name*), 'Sélectionnez la catégorie*' (Select category*), 'Sélectionnez le modèle' (Select model), 'Quantité*' (Quantity*) with a plus icon, and 'Description'. Below these is a button '+ Numéro de série' (Add serial number) and a text input for 'url de l'image' (image url). At the bottom, there are two checkboxes: 'En maintenance' (checked) and 'Existant' (unchecked), followed by an 'Envoyer' (Send) button.



The modal for serial numbers is shown. It contains two text input fields labeled 'Numéro de série 1' and 'Numéro de série 2'. Below these fields are two buttons: 'Sauvgarder' (Save) and 'Fermer' (Close).

Pour pouvoir lister des catégories j'ai dû créer un service qui s'occupe de ça avec la commande `ng generate service service/category` ou plus court `ng g s service/category`. C'est un service que j'appelle à l'initialisation de mon composant grâce à la méthode `ngOnInit()` qui est une méthode Angular qui gère le cycle de vie d'un composant. J'ai également répété la même opération pour le service des fabricants.

J'avais fait erreur en construisant le composant `equipment` parce que j'ai réalisé que mon code était assez répétitif et par la suite pas très modulaire. J'ai pensé pouvoir me rattraper en allant vite mais ce n'était pas le cas et était une mauvaise

pratique à ne pas répéter. J'ai également fait erreur en ne pas factorisant mon code au fur et mesure.

Mais utiliser Angular m'a donné de la joie et un sourire sur le visage (une fois que j'ai commencé à maîtriser le framework).

3.3 Utilisation d'Angular Material

Angular Material est une bibliothèque de composants UI (user Interface) prêts à être utilisés. Cette bibliothèque contient des boutons, des `mat-form-field` qui sont

des champs d'un formulaire (que j'ai utilisé extensivement), des input, etc. Tout dans le style de Material Design.

Pour l'utiliser j'ai dû d'abord l'installer avec la commande `ng add @angular/material`. Ensuite j'étais demandé de spécifier le thème et j'ai choisi Purple&Green. Une fois choisi j'ai dû ajouter la classe `mat-app-background` dans la balise `body` de la page d'index.

Chaque composant dans Angular Material est organisé dans son propre module séparé. Pour savoir comment utiliser un composant spécifique, j'avais à me rendre simplement sur l'onglet API de ce composant.

L'un des modules que je crois avoir le plus utilisé est le *form-field*, parce qu'il me permet d'ajouter des fonctionnalités supplémentaires à des champs input (saisie). Par exemple, je peux activer un label flottant qui affiche le nom du champ au dessus lorsque l'utilisateur commence à écrire, je peux également afficher des messages d'erreur si l'utilisateur entre des informations incorrectes et etc.

```
<!-- Name field -->
<mat-form-field class="form-field">
  <mat-label>Nom de l'équipement</mat-label>
  <input matInput formControlName="itemName"/>
  <mat-error *ngIf="formGroup.get('itemName')?.hasError('required')">
    Nom de l'équipement requis
  </mat-error>
</mat-form-field>
```

Figure 17. Le code d'un composant `form-field` pour le champ de nom

Dans l'extrait de code qu'on peut voir sur la figure 17, on constate qu'il existe un champ `mat-form-field` qui correspond au module de material angular. Ensuite il y a un champ `mat-label` qui représente un label pour le input tout simplement. Un champ input suit où l'utilisateur peut entrer le nom de l'équipement et il contient un `formControlName` qui est utilisé afin de bind (lier) le champ input avec la variable `itemName` du côté du composant où un `formBuilder` « fabrique » un `formGroup` qui contient cette variable. Et le dernier champ est le `mat-error` qui affiche le message d'erreur contenu entre la balise ouvrante et fermante si l'utilisateur n'entre pas de nom d'équipement. Ceci est possible grâce à la directive `*ngIf` qui vérifie si le `formControl` contient un validateur `required` (c'est tout simplement un validateur de saisie qui contrôle si le champ est bien rempli).

8. Présentation du jeu d'essai

Un programme n'étant pas une chose palpable et n'ayant pas d'odorat (à ne pas confondre avec l'adage d'un code « qui sent mauvais ») peut prouver sa qualité seulement à travers des tests. Pour cette raison il est primordial d'effectuer des tests d'une application.

Je vais vous présenter les tests unitaires de mon application. Les tests unitaires sont les tests qui vérifient l'exactitude d'une unité atomique (indivisible) de l'application, comme une méthode par exemple.

Pour le jeu d'essai de Locparc j'ai utilisé Spring Boot Test qui est fourni par le framework. Il inclut plusieurs framework de tests comme Junit5 et AssertJ que j'ai choisi d'utiliser grâce aux nombreuses méthodes de tests qu'il possède.

```
<dependency>
  <groupId>org.assertj</groupId>
  <artifactId>assertj-core</artifactId>
  <version>3.23.1</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.hamcrest</groupId>
  <artifactId>hamcrest</artifactId>
  <version>2.2</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter</artifactId>
  <version>5.9.2</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.mockito</groupId>
  <artifactId>mockito-core</artifactId>
  <version>4.8.1</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.mockito</groupId>
  <artifactId>mockito-junit-jupiter</artifactId>
  <version>4.8.1</version>
  <scope>compile</scope>
</dependency>
```

Figure 1. Le fichier pom.xml de spring boot test

Afin de tester la couche de l'application qui s'occupe de la communication avec la base de données (grâce aux DAO), il a fallu utiliser une base de données à part afin de ne pas persister des données de test dans la base de données qui va être utilisée en production, c'est où le SGBDR H2 m'a été d'une grande utilité. H2 est un système de gestion de base de données dit « in memory » ou en mémoire, qui signifie que les données persistent tant que l'application tourne, et cela fait de H2 le candidat parfait pour le SGBDR utilisé lors des tests.

J'ai fait le choix de vous présenter un test d'une requête qui se trouve dans l'interface ItemDao.

```

3 usages new *
@Query("FROM Item as i " +
    "WHERE lower(i.name) LIKE concat('%', :name, '%') " +
    "AND i.existing = true " +
    "AND i.onMaintenance = false " +
    "AND i.id NOT IN (SELECT i2.id FROM Item as i2 " +
        "JOIN i2.orders oi " +
        "WHERE oi.returnDate IS NULL " +
        "AND oi.approved = true ) ")
List<Item> findAvailableItemsByName(@Param("name") String name);

```

Figure 2. La requête à tester

J'ai commencé par me rendre dans le package dédié aux tests créé par Spring boot. Une fois dedans j'ai créé une classe de test que j'ai nommé `ItemDaoTest` (afin de signifier qu'il s'agit d'une classe de test). J'ai annoté cette classe avec l'annotation `@DataJpaTest` qui est basé sur une approche appelée « slice based testing » qui veut dire qu'il ne faut charger que la partie de l'application qui nous est nécessaire (dans mon cas Spring Data Jpa) et pas l'entièreté de l'application qui peut prendre plus de temps et utiliser plus de ressources.

J'ai ensuite injecté mon repository `ItemDao` que j'ai appelé `underTest` en suivant la convention.

Après ça j'ai ajouté une méthode `setUp()` avec l'annotation `@BeforeEach` indiquant au framework qu'il s'agit d'une méthode à exécuter avant chaque test. J'ai persisté quatre `Item` afin de créer un jeu de données suffisant pour effectuer mes tests, j'ai jugé la création des `OrderItems` (orders dans la requête sur la figure 2) comme superflue parce qu'elle nécessite la création et persistance d'une entité `Order`, et une persistance d'un `Item` également afin de pouvoir être créés, ce qui rajoute une complexité inutile et nuit à la lisibilité du code.

Ensuite, j'ai nommé la méthode de test en suivant la convention : lui donnant un nom d'une fonctionnalité qu'elle doit vérifier et la précédant du mot « should » afin de suivre la convention « should style ». J'ai également attribué une annotation `@Test` afin d'indiquer à spring boot qu'il s'agit d'un test.

Dans le test on peut constater trois commentaires : `Given`, `When`, `Then` qui sont des équivalents pour les données en entrées, données attendues et données en sortie.

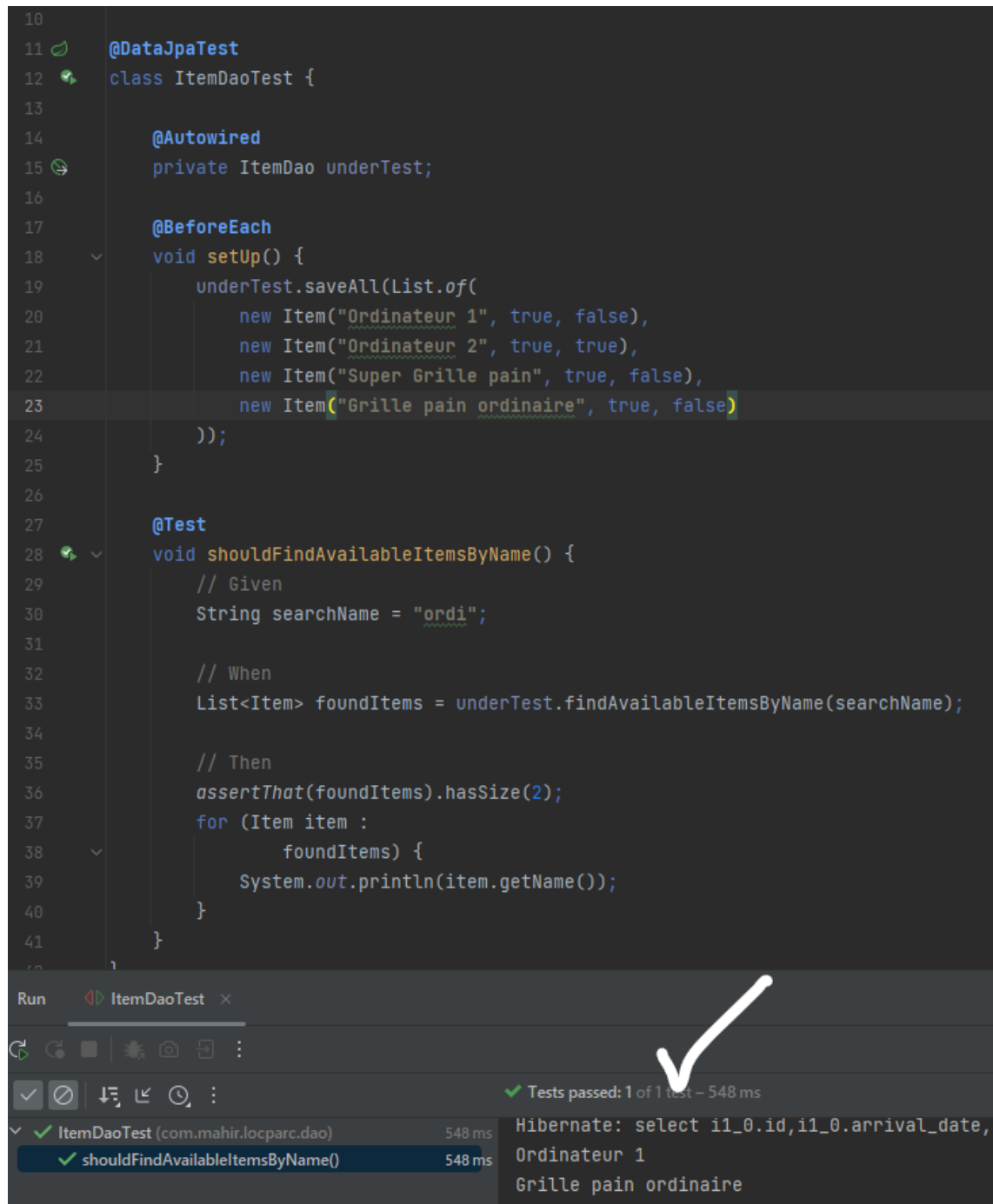
Sous le commentaire `Given` j'ai les données en entrées, un `String searchName` qui sert de paramètre à la requête que j'étais en train de tester.

Sous le commentaire `When` j'ai les données attendues ou ce que la requête renvoie.

Et dernièrement sous le commentaire Then j'ai une assertion qui signifie que foundItems est supposé avoir une longueur de 2.

La vérification de données attendues avec les données en sortie s'appelle des assertions. AssertJ nous fournit une classe Assertions et cette classe contient des méthodes d'assertion qui lancent des exceptions quand une condition n'est pas remplie et ainsi font échouer les tests.

Comme on peut constater le test a passé et ma requête fonctionne.



```
10
11  @DataJpaTest
12  class ItemDaoTest {
13
14      @Autowired
15      private ItemDao underTest;
16
17      @BeforeEach
18      void setUp() {
19          underTest.saveAll(List.of(
20              new Item("Ordinateur 1", true, false),
21              new Item("Ordinateur 2", true, true),
22              new Item("Super Grille pain", true, false),
23              new Item("Grille pain ordinaire", true, false)
24          ));
25      }
26
27      @Test
28      void shouldFindAvailableItemsByName() {
29          // Given
30          String searchName = "ordi";
31
32          // When
33          List<Item> foundItems = underTest.findAvailableItemsByName(searchName);
34
35          // Then
36          assertThat(foundItems).hasSize(2);
37          for (Item item :
38              foundItems) {
39              System.out.println(item.getName());
40          }
41      }
42  }
```

Run ItemDaoTest

Tests passed: 1 of 1 test - 548 ms

Test Name	Duration	Output
ItemDaoTest (com.mahir.locparc.dao)	548 ms	Hibernate: select i1_0.id,i1_0.arrival_date,
shouldFindAvailableItemsByName()	548 ms	Ordinateur 1 Grille pain ordinaire

Figure 3. La classe de test ItemDaoTest et le test passé

9. Description de la veille sur les vulnérabilités de sécurité

Durant le développement de mon projet Locparc, la veille sur les vulnérabilités de sécurité prenait une place importante dans la hiérarchie de développement de l'application. La veille effectuée influençait également mes choix pour les versions des framework et des packages que j'ai fait le choix d'utiliser.

J'ai choisi SpringBoot pour le back-end grâce à son autoconfiguration et sa gestion de dépendances. Mais avant de choisir la version de SpringBoot je me suis rendu sur le site <https://devhub.checkmarx.com/advisories/> où j'ai tapé les mots clés « spring boot » et je n'ai rien trouvé. Cela me semblait étrange. Je savais que Spring Boot était sécurisé mais je ne pensais pas qu'il l'était à ce point. C'est à ce moment là où j'ai écrit « springboot » en attachant les mots spring et boot, et effectivement il y avait des résultats, qui n'étaient pas nombreux, mais présent quand même.

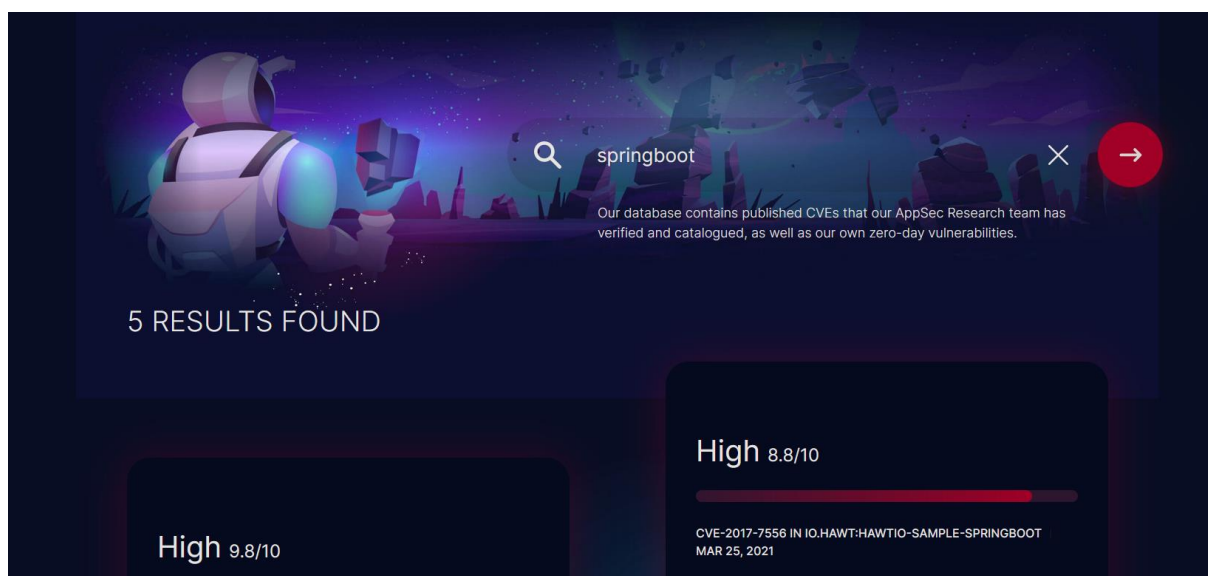


Figure 1. Les vulnérabilités trouvées sur le site <https://devhub.checkmarx.com/>

Parmi ces cinq vulnérabilités, il y avait trois qui étaient sur Apache Camel, un logiciel que je ne connaissais pas auparavant, qui permet la communication entre différentes applications.

La seule vulnérabilité sur SpringBoot lui-même était causé par un plug-in que je n'utilise pas et cette vulnérabilité datait de juin 17 2021.

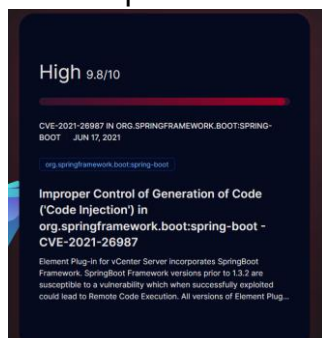


Figure 2. Element plug-in « code injection » vulnérabilité

J'ai également fait des recherches sur Angular, le framework front-end que j'ai fait le choix d'utiliser. Contrairement à SpringBoot, Angular avait plus de vulnérabilités.

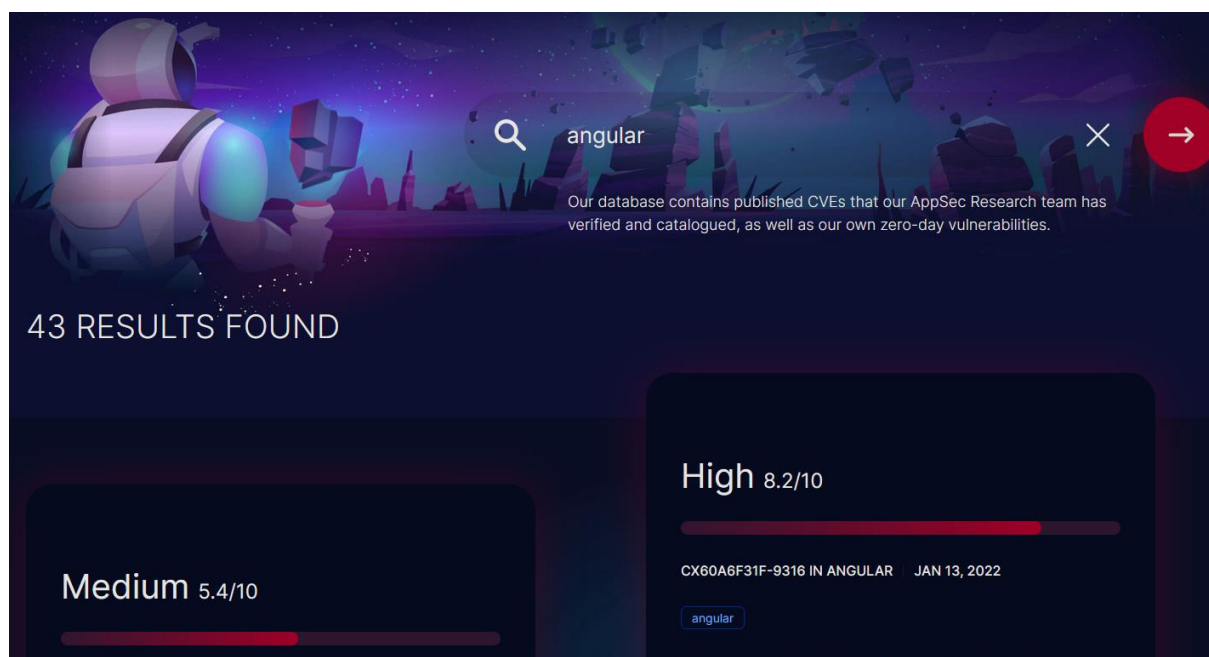


Figure 3. Résultat de recherche en entrant le mot clé « angular »

43 pour être précis. Mais après une étude de toutes les vulnérabilités plus ou moins récentes, il n'y avait aucune qui concernait mon application Locparc.

J'ai aussi fait des recherches sur les vulnérabilités de SpringBoot et Angular en entrant le mot clé du framework suivant de « vulnerabilities » et en utilisant des moteurs de recherche Google et DuckDuckGo car les deux moteurs ne donnent pas les mêmes résultats. Les sites retournés étaient les suivant →

Pour Google :

1. <https://spring.io> Le site officiel de Spring
2. <https://security.snyk.io> qui n'avait pas de résultats récents
3. <https://cvedetails.com> qui est une source de données sur les vulnérabilités réputé
4. <https://kaspersky.com> qui liste une vulnérabilité sur la dépendance « Spring4Shell »

Les résultats similaires pour DuckDuckGo :

1. <https://spring.io>
2. <https://stackoverflow.com>
3. <https://redhat.com>

...

J'ai les mêmes étapes pour angular et voici la liste de sites :

1. <https://angular.io> le site officiel
2. <https://stackoverflow.com>

3. <https://security.snyk.io>
4. <https://securityboulevard.com>

Je n'ai pas trouvé de faille de sécurité me concernant sur aucun de ces sites. Mais en consultant mon fichier pom.xml, l'IDE que j'utilise (Integrated Development Environment), IntelliJ, m'avait averti de deux potentielles failles de sécurité pour une de mes dépendances.

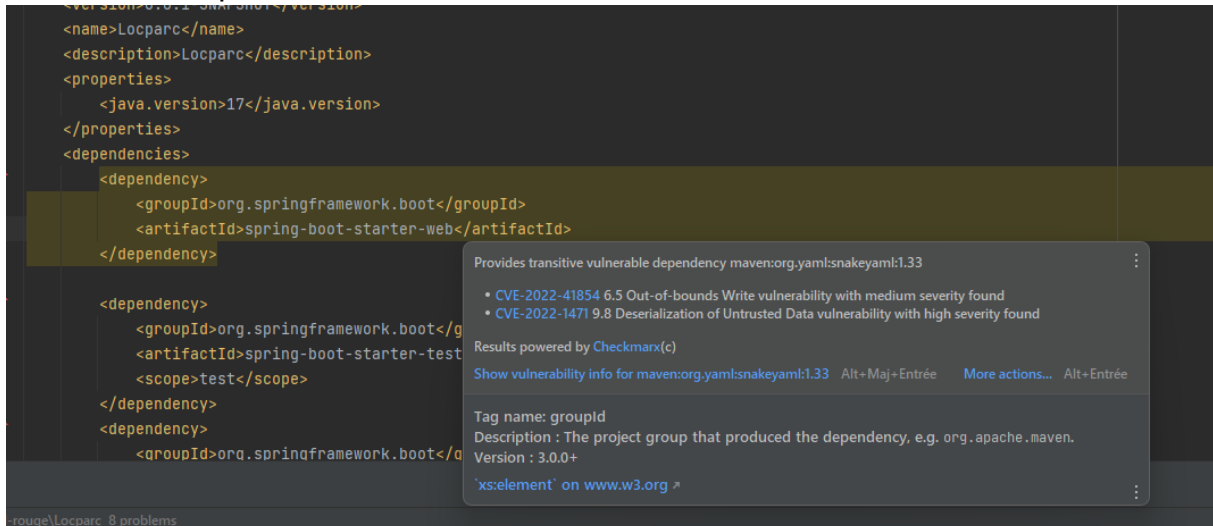


Figure 4. Avertissement de l'IDE IntelliJ sur des potentielles failles de sécurité

Après avoir fait des recherches j'étais content de savoir que je n'étais toujours pas concerné.

J'ai me suis aussi régulièrement rendu sur le site l'agence nationale de la sécurité des systèmes d'information (ANSSI) afin de mieux connaître les risques actuels de cybersécurité et également des bonnes pratiques de développement qui nous permettent d'éviter au maximum ses risques.

10. Description d'une situation de travail ayant nécessité une recherche

J'ai décidé de déployer mon projet avec docker parce que j'en ai tellement entendu parler qu'il fallait absolument que je teste. Mais pour ce faire il a fallu apprendre qu'est ce que docker et comment l'utiliser.

Docker est un service qui permet de déployer des applications dans des conteneurs. Il permet de faire bien plus mais c'est principalement pour le déploiement qu'il m'a intéressé.

Avant d'apprendre à utiliser docker j'ai dû me connecter sur le serveur virtuel de chez OVH qui m'était attribué par monsieur Quirin. Ensuite je lui ai installé Debian version 10, un système d'exploitation open source. Et une fois terminé j'ai installé openSSH un outil de connexion à distance pour pouvoir me connecter avec le logiciel Putty sur le port 22. Après m'avoir connecté avec Putty j'ai enfin installé docker. Jusqu'à maintenant je n'avais rien de nouveau à apprendre.

J'ai ensuite tapé « docker deployment » dans le moteur de recherche et cliqué sur le premier résultat : <https://docs.docker.com/language/java/deploy/>. Une fois sur la page j'ai cliqué sur « Get started » dans le menu déroulant à gauche.

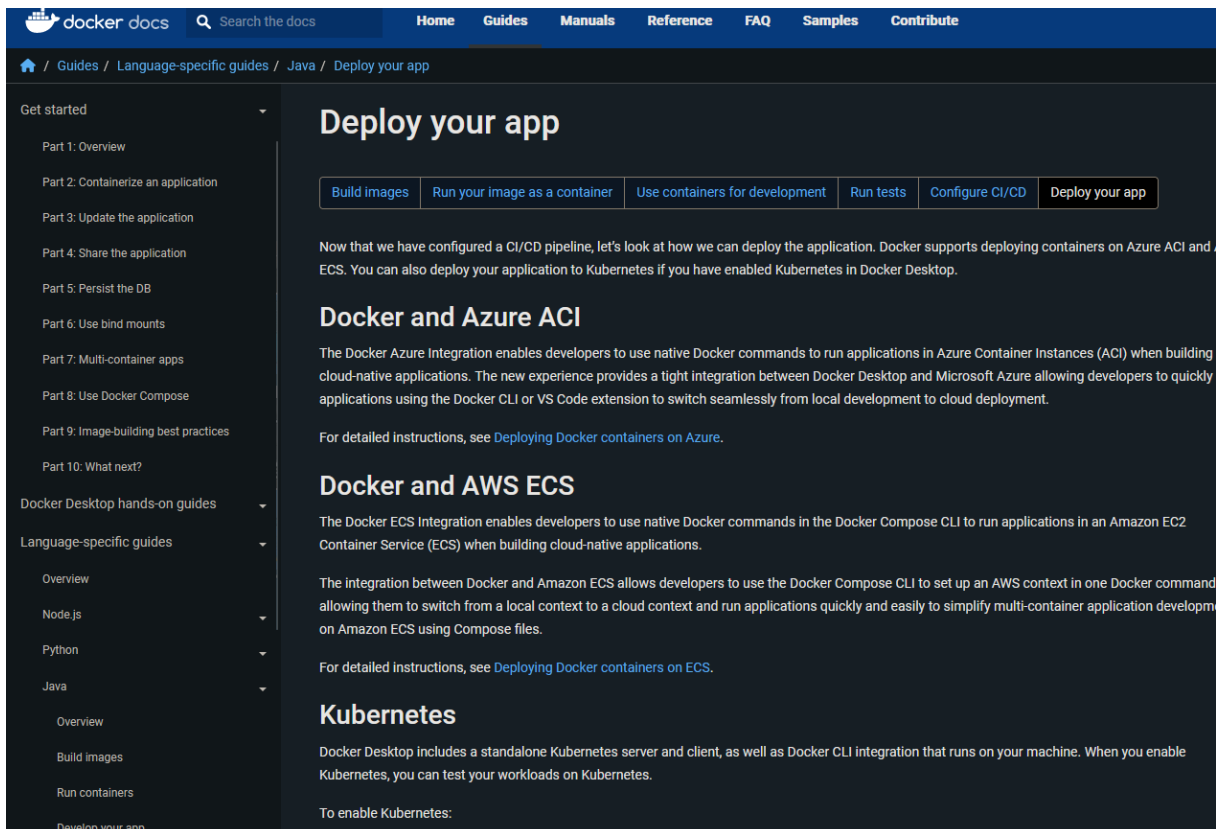


Figure 1. Page expliquant le déploiement avec Docker

Après avoir appris qu'est ce qu'une image et un conteneur docker, je me suis rendu sur l'étape suivante où j'avais déjà un Dockerfile (un fichier contenant du script qui construit l'image Docker, ou autrement dit crée un conteneur) à créer. Une fois terminé je suis passé à la prochaine étape et pour ne pas risquer d'être trop ennuyant je vais résumer l'expérience. J'ai appris comment créer un Dockerfile, comment démarrer, consulter, arrêter et supprimer un container. J'ai ensuite appris ce que sont les volumes et comment les utiliser. Comment installer mysql et finalement comment faire un docker-compose afin de déployer plusieurs services d'un seul coup. A la dernière page j'ai cliqué sur la vidéo « getting started » d'une durée de 3h. Après l'avoir regardé j'étais prêt de chercher ailleurs.

J'ai commencé par chercher pour des exemples de Dockerfile en tapant les mots clés « Dockerfile anatomy » ce qui m'a emmené sur la page <https://dev.to/mbaris/anatomy-of-a-dockerfile-4b4p> où j'ai beaucoup appris sur les fichiers Dockerfile. J'ai aussi entendu parler d'une repository GitHub appelé « awesome-compose » qui contient des exemples de fichiers docker-compose qui j'ai étudié également.

1. Déploiement base de données

Les nouvelles informations combinées avec les cours mon permis de commencer mon déploiement doucement mais surement.

Je savais que je ne connaissais pas encore assez sur des networks docker pour en créer un network fonctionnel. J'ai donc opté pour créer un docker compose avec une base de données mysql qui créera tout seul un network docker au quel je peux ensuite simplement connecter mon backend. Vu le peu de temps qu'il me restait pour rendre mon projet je ne voulais pas risquer d'être trop lent et n'avoir rien à la fin mais j'ai prévu d'apprendre docker d'avantage.

```
1  version: '3'
2  services:
3    mysql:
4      image: mysql:5
5      restart: always
6      environment:
7        MYSQL_ROOT_PASSWORD: [REDACTED]
8        MYSQL_DATABASE: locparc
9      volumes:
10       - mysql_data:/var/lib/mysql
11
12   phpmyadmin:
13     image: phpmyadmin/phpmyadmin
14     container_name: phpmyadmin
15     restart: always
16     environment:
17       PMA_HOST: mysql
18       PMA_PORT: 3306
19     ports:
20       - "8080:80"
21
22   volumes:
23     mysql_data: [REDACTED]
```

Voici à quoi ressemble mon docker-compose, Il est **composé** de deux services :

- Le service mysql →
image : mysql:5 avec la version 5 de mysql plus précisément, parce que c'est le dialecte choisi pour hibernate dans le fichier application properties,
restart : always est là pour dire à docker de redémarrer le conteneur automatiquement s'il est arrêté de manière abrupte.
environment : définit les variables d'environnement, le mot de passe de mysql pour root et le nom de la base de données.
volumes : précise l'endroit où mysql va stocker les données (le fichier défini est là

juste pour une précision parce que c'est l'endroit de stockage par défaut pour mysql).

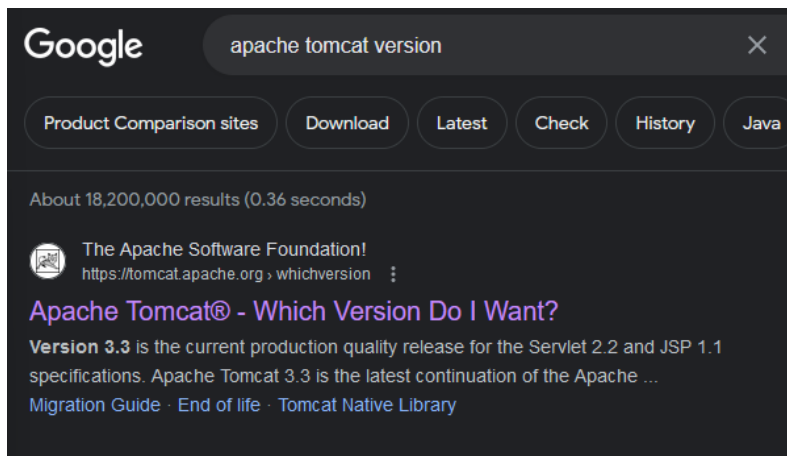
- Le service phpmyadmin → pareil comme pour le service mysql pour les trois premières valeurs, ensuite je précise le host pour dire à phpmyadmin sur quelle base de données se connecter (PMA_HOST), le port sur lequel se connecter à mysql PMA_PORT et ensuite ports: mappe le port 8080 au port 80 du conteneur.
- Le volumes → indique le nom du volume qui est le mysql_data.

Une fois que j'ai tout défini je me suis connecté au serveur avec l'outil winSCP qui nous permet de transférer les fichiers entre notre système et celui du serveur. J'ai ensuite transféré le fichier docker_compose.yml dans le répertoire /home/debain.

J'ai ensuite tapé la commande suivante : `sudo docker-compose -f -d docker_compose.yml up` ce qui a lancé le processus d'installation et de configuration de conteneurs docker (en arrière plan parce que j'ai mis la commande -d pour detached). Et avec ça j'avais ma base de données et phpmyadmin préparés.

2. Déploiement Backend

La prochaine étape était le backend avec spring boot. J'ai fait des recherches d'avantage et j'ai dû même appeler mon formateur à l'aide car l'installation ne fonctionnait pas. Il s'agissait d'un problème d'incompatibilité de tomcat 8.5 et Java 17 qu'il nous a fallu 2 heures pour le résoudre, en augmentant la version de tomcat de 8.5 à 10. La recherche « apache tomcat version » sur google nous a pointé dans la bonne direction.



Nous avons ensuite cliqué sur le premier résultat puis nous avons procédé à voir que nous n'avions pas la bonne version de tomcat.

Les versions à partir de la 10 sont compatibles avec Java 17.

Figure 3. Résultat de la recherche google

J'ai procédé par créer un Dockerfile pour le backend de Locparc :

```
1 FROM tomcat:10-jdk17
2 COPY target/Locparc-0.0.1-SNAPSHOT.war /usr/local/tomcat/webapps/ROOT.war
3 ENTRYPOINT ["/bin/bash", "/usr/local/tomcat/bin/catalina.sh", "run"]
```

Le FROM tomcat :10-jdk17 signifie que l'image de base sera l'image officielle de Tomcat prise de DockerHub. La ligne suivante copie le fichier .war dans le dossier target créé par Maven au moment du build dans le dossier

/usr/local/tomcat/webapps/ et le renomme ROOT.war voulons dire qu'il sera considéré comme l'application par défaut de tomcat. Et la dernière ligne spécifie les commandes à exécuter au moment du démarrage du conteneur, le fichier catalina.sh (où se trouve la configuration de tomcat) à exécuter avec le bash et avec la commande run.

Mais avant de lancer le Dockerfile il faut d'abord transférer les fichiers de locparc vers le serveur. J'ai utilisé git pour ça (que j'ai dû d'abord installer bien sûr) : j'ai téléchargé la repository de mon profile github. Une fois que c'était fait j'ai créé un fichier settings.xml qui se trouve dans /home/debian/.m2/ afin que maven puisse prendre les profiles nécessaires, avec les mots de passes pour le service des mails et le secret jwt ainsi que les informations pour la base de données.

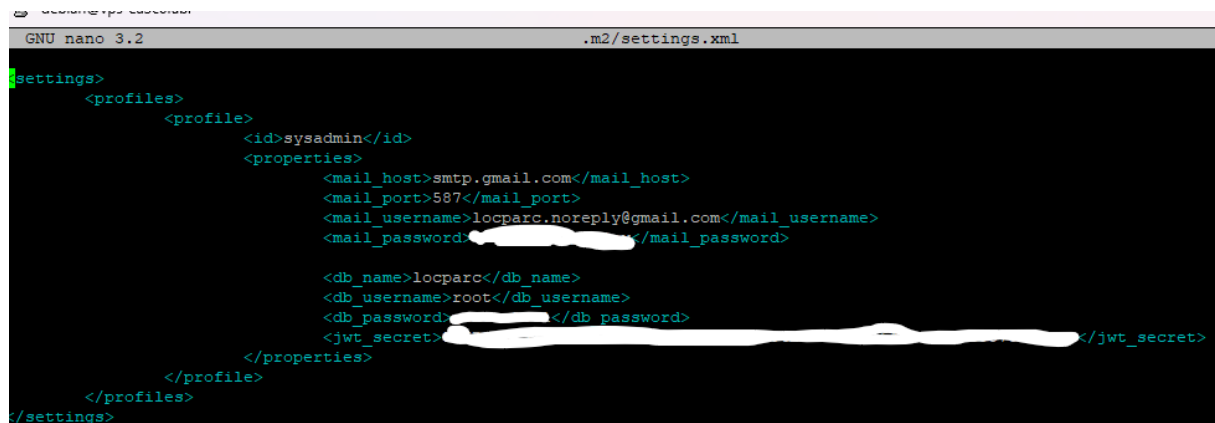


Figure 4. Fichier settings.xml

J'ai ensuite lancé Maven avec la commande : `bash mvnw package -P prod,sysadmin --settings /home/debian/.m2/settings.xml`

Qui est assez claire. Elle dit a maven de chercher les paramètres pour les profiles (-P) prod, sysadmin dans le fichier settings.xml. Une fois fini le dossier target a été créé avec le fichier .war ce qu'il signifie que nous pouvons lancer le Dockerfile et construire l'image docker. J'ai donc lancé le Dockerfile : `docker build --no-cache -t locparc .` je l'ai instruit à créer l'image sans cache et lui donne le nom (-t) de locparc dans le dossier actuel (.).

Une fois l'image créé j'ai pu créer le conteneur avec la commande suivante : `docker run -d --net debian_default --name=locparc -p 8181:8080 locparc`

Elle lance le conteneur en mode détaché sur le network debian_default créé précédemment par docker compose, lui donne le nom de locparc, mappe le port 8181 (le port du hôte) vers 8080 (le port du conteneur) et précise que c'est l'image locparc.

3. Déploiement Frontend

Je suis ensuite passé vers le déploiement d'Angular. Pour ce faire j'ai du créer un fichier de configuration pour nginx que j'ai trouvé dans les cours monsieur

Bansept et le nommer `nginx-custom.conf`, ensuite un `Dockerfile` et j'ai ajouté les deux fichiers à la racine de `locparc`.

```
nginx-custom.conf
1  server {
2      listen 80;
3      location / {
4          root /usr/share/nginx/html;
5          index index.html index.htm;
6          try_files $uri $uri/ /index.html;
7      }
8  }
```

Figure 5. Fichier de configuration du serveur nginx

```
src > Dockerfile > ...
1  FROM node:18.10-alpine AS build
2  WORKDIR /app
3  COPY package.json package-lock.json ./
4  RUN npm install
5  COPY . .
6  RUN npm run build
7
8  FROM nginx:1.17.1-alpine
9  COPY --from=build /app/dist/locparc-angular /usr/share/nginx/html
10 COPY ./nginx-custom.conf /etc/nginx/conf.d/default.conf
```

Figure 6. Dockerfile pour locparc-angular

Le fichier de configuration écoute les requêtes sur le port 80, déclare les sources du site, dans le cas d'une terminaison d'url par « / » cherche les fichiers index et finalement l'instruction `try_files` redirige toutes les urls vers la page `index.html`.

Le Dockerfile prend une image node pour alpine (de petite taille) afin de compiler le code source et lui donne l'alias `build` ensuite se déplace dans le dossier `app`. Ensuite on copie le `package.json`, puis on installe les dépendances et on copie les sources dans le conteneur et on lance la commande `build`.

Dans la deuxième partie le Dockerfile prend une image de base pour nginx avec le tag alpine pour une image plus petite, ensuite on fait une copie de l'application et la mettons dans le répertoire par défaut de nginx, et dernièrement on copie le fichier de configuration créé juste avant et on le met dans `/etc/nginx/conf.d/` en lui donnant le nom « `default.conf` ».

Et il reste seulement de construire l'image avec la même commande que j'ai utilisé pour le backend : `docker build --no-cache -t locparc-angular .`

Une fois terminé on lance le conteneur :

```
docker run -d --name=locparc-angular -p 80:8080 locparc-angular
```

Avec cette commande le frontend est déployé.