
LiVrai

Sommaire

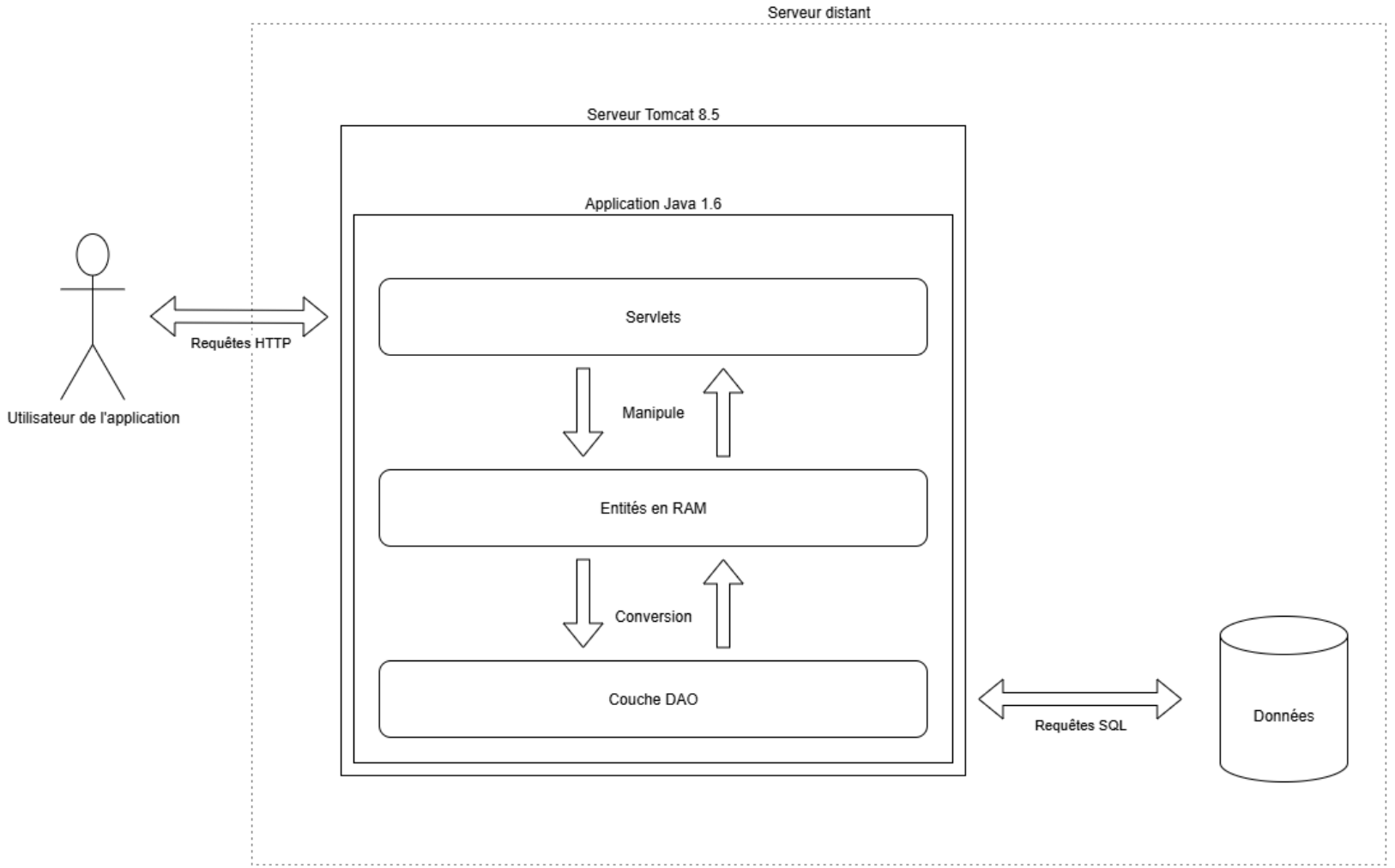
Contexte de l’application

- Livrai est une entreprise de livraison, originairement spécialisée dans les marchandises en grosse quantité pour les professionnels.
- Désormais, l’entreprise entend s’étendre à tous types de colis et ce dans toute la France
- Cela l’amène à reconsidérer son application de sorte à accommoder l’ensemble des utilisateurs français

Réalisation d’un audit

- Suite à la prise de contact avec Livrai, nous avons réalisé un audit de l’application
- Celui-ci a permis de mieux saisir les nuances et le fonctionnement en l’état de l’application
- Les features déficiantes de Livrai v1 ont été ciblées
- Des axes d’amélioration potentiels ont été trouvés

Architecture actuelle de l’application



Architecture actuelle de l’application

- Actuellement, l’application est basée sur l’utilisation de servlets et de Java EE. Il s’agit d’une architecture en couche.
- Les requêtes clients atteignent le serveur Tomcat, qui les redirige vers le servlet concerné. Dans ce servlet, les requêtes sont ensuite alimentées par des données et retournée sous la forme de page HTML crées via des templates et alimentées en données par un ensemble de requêtes prévues dans la couche DAO. Cette couche DAO communique via le driver JDBC avec un serveur MySQL.

Points forts

- Utilisation de code écrit en Java, langage fiable et sécurisé
- Clarté des fonctions et de leur utilisation
- La majorité des fonctions principales de l’applicatif sont déjà, en un sens, implémentées et fonctionnent en lien avec les volontés du client et les contraintes techniques.
- L’ensemble des requêtes SQL sont lisibles dans la couche DAO de l’application

Points faibles

- Absence de CSS moderne et look austère de l'applicatif fontend, ce qui risque de rebuter la majorité des Français habitués désormais à des standards plus esthétiques
- Pas de compatibilité mobile, ce qui est un problème de taille lorsque l'on sait qu'une grande majorité des gens privilégient désormais la navigation sur leur smartphone.
- Erreur lors d'un refus de facturation (code 405)
- Stockage des données utilisateur en clair causant une faille de sécurité importante et la possibilité pour un hackeur d'obtenir le contrôle total de l'application en cas de compromis de son intégrité.
- Fonctionnement via une session, cookies. Ce mode de fonctionnement rend l'applicatif sensible aux vols de session ou à des attaques de type Man in the Middle.
- Le stockage des cookies d'authentification dans le navigateur de l'individu demandera plus de rigueur aux utilisateurs voulant utiliser l'application dans un environnement public tel qu'une bibliothèque où plusieurs personnes partagent le même ordinateur.

Conclusion de l’audit

- Moderniser l’application
- Utilisation de technologies plus modernes (et reconnues pour leur fiabilité / performances)
  - Spring Boot / Spring Security
  - Angular
- Revoir l’affichage de sorte à le rendre plus engageant pour les futurs utilisateurs
- L’ajout de Spring et la réalisation d’une API liée à une application front-end pourrait également permettre plusieurs façons d’interagir avec l’application,
  - Back-office
  - Appels REST
  - Application mobile (Ionic ?)

Utilisation de Spring

- Spring va permettre d'avoir une architecture plus rigoureuse, basée sur l'architecture en couche. Chaque section de l'API pourra ainsi être réalisée par un ou plusieurs développeurs sans interférer avec le travail des autres.
- L'ajout de Spring va aussi permettre d'avoir une sécurité renforcée via Spring Security. L'utilisation d'une chaine de filtre personnalisée et l'ajout de l'utilisation d'un JWT va aider dans le cas où l'on aurait besoin d'avoir des informations sur l'utilisateur en amont de l'accès aux point d'accès de notre application.
- Il sera possible, par l'utilisation d'un filtre, de réaliser un RBAC (Role Based Access Control) de sorte à permettre des features uniquement accessibles par une équipe et non l'ensemble des utilisateurs.

Utilisation de Spring

- L’ajout de Lombok devrai drastiquement améliorer la lisibilité du code et son écriture.
- Utilisation du JWT et fonctionnement du back-end via une API RESTful. Le JWT permettra de réaliser un back-end stateless, ce qui aidera dans la sécurisation et la résilience à une intrusion par un individu malintentionné.
- Le passage du jeton d’authentification via les cookies et une redirection automatique de l’HTTP vers l’HTTPS va aussi permettre de renforcer l’applicatif. Le passage du JWT par les headers d’une requête et le fait que cela soit le seul élément conservé dans la partie frontend va aider à éviter de nombreuses failles de sécurités, dont une attaque de type Man In the Middle.

Utilisation d’Angular

- Le fonctionnement d’Angular et la méthode de développement va permettre un travail en équipe plus efficace et un répartissement des taches plus facile.
- En utilisant Angular, il sera plus facile de réaliser la sécurisation de la partie front-end de l’application (guards, interceptors, proxies…).

Utilisation d’Angular

- Angular va permettre une amélioration de la structure de l’interface avec l’utilisateur.
- Il sera possible de réaliser un ou plusieurs themes dans l’application, dont un qui sera adapté aux besoins de personnes en situation d’handicap visuel (via l’utilisation de contrastes élevés).
- Dans le cas où l’on aurait besoin de réaliser une application mobile par la suite, il sera possible de transformer l’application web en une version mobile de façon simplifiée en couplant l’applicatif Angular avec Ionic.

Utilisation de Docker

- En couplant Docker à un environnement cloud, il sera possible de passer facilement d'une application hébergée en bare metal ou vers un déploiement en nuage.
- Via Kubernetes, il sera même possible d'améliorer la résilience de notre applicatif à la surcharge de requêtes, à des bogues causant un crash de notre applicatif ou de la base de données.

Utilisation de Docker

- Docker va permettre de réaliser une série d'environnement de développement et de testing pour accélérer et simplifier l'avancement de l'application.
- En utilisant Docker, il sera également possible de déploier l'applicatif dans plusieurs endroits de façon plus aisée. Plus besoin de demander à des architectes système de mettre en place toute une série de dépendances ou de machines virtuelles.

Evolutions futures

- Il sera même possible de réaliser notre propre environnement de cloud privé via un investissement dans un environnement bare metal suffisant et la mise en place d'une architecture via Openstack, Ansible et Terraform.
- L'ajout d'une interface mobile (native via Kotlin ou cross-platform avec Ionic) permettra l'ajout de fonctionnalités liées à l'environnement smartphone. Par exemple, on peut imaginer la capacité pour le service de livraison de confirmer le dépôt d'un colis via une photo prise via l'appareil photo du téléphone du livreur, le scan d'un code QR par les clients pour confirmer la r éception de leur bien, etc.

Evolutions futures

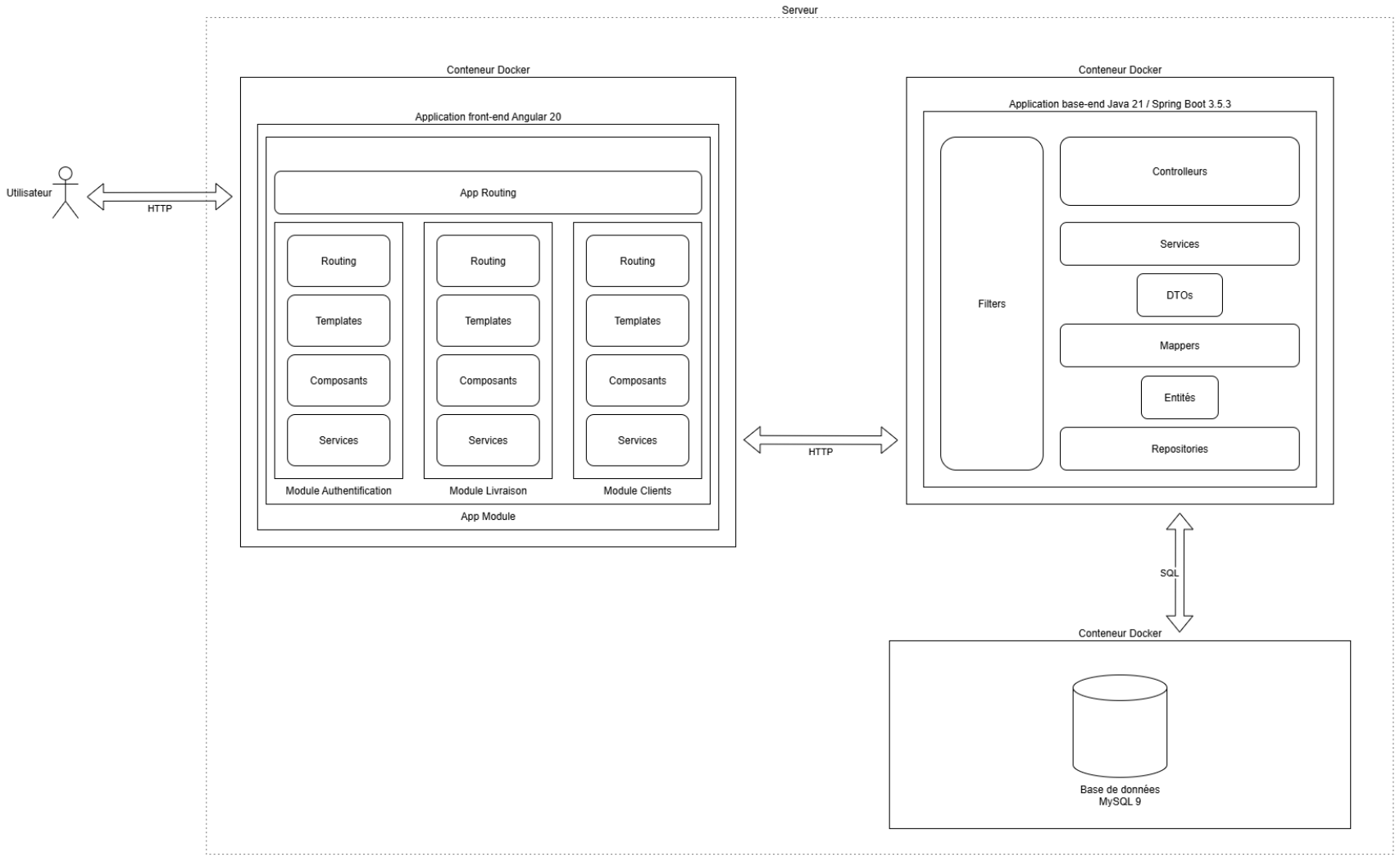
- Comme discuté précédemment, il peut être possible d'ajouter la prise en charge de Kubernetes de sorte à permettre un déploiement plus aisé encore de notre applicatif. Celle-ci deviendra également plus résistante à la « casse ».
- Dans un environnement cloud, la sécurité de notre application et l'application de certificats de types TLS sera plus aisée. Il sera possible de conserver les données sur le territoire français en passant par exemple par Azure Cloud France.

Nouvelles fonctionnalités

Nouvelles fonctionnalités

- Les fonctionnalités globales de l'applicatifs sont les mêmes, mais la répartition de ces dernières est désormais dépendantes de plusieurs rôles dépendant du service auquel le compte utilisateur sera lié.
- Ainsi, le service commercial aura la capacité de gérer les utilisateurs de son cercle et des autres cercles. Il lui sera possible de consulter les factures, de confirmer / infirmer une commande.
- Le service de livraison, quant à lui, pourra consulter les commandes de sorte à pouvoir répartir les livreurs sur le territoire et assurer un suivi plus rapide des transports.
- Les clients classiques, eux, auront la possibilité de commander des fournitures. Cette fois-ci, il s'agira non pas de professionnels uniquement mais de l'ensemble des Français qui pourront se connecter à la plateforme.

Nouvelle architecture



Nouvelle architecture

- L’ensemble sera une architecture de type client-serveur, avec un applicatif de type API pour le backend et une application SPA pour la partie frontend.
- La partie API sera une architecture en couche, comme précédemment. Cette fois-ci, le nombre de couches sera cependant plus important de sorte à permettre une meilleure maintenabilité et scalabilité de l’applicatif.
- La partie frontend sera une SPA Angular, donc une architecture basée sur les composants. En utilisant plusieurs modules, il sera possible de mettre en place du lazy loading, de sorte à améliorer la rapidité de l’applicatif et d’éviter le stockage sur l’ordinateur client de données non nécessaire à son usage du jour.
- La dockerisation de l’ensemble des applicatifs permettra le déploiement des différentes couches de notre application full-stack sur n’importe quel environnement compatible avec Docker.