Spécifications techniques

Réservation de salles M2L

|  |  |  |
| --- | --- | --- |
| **Charles BARDIN**  *Chef de projet* |  | **Gaël LEHCHIBI**  *Développeur* |

Historique des versions

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Changement | Auteur |
| 04/11/2015 | 0.1 | Création du document | Charles BARDIN |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table des matières

[Architecture globale 4](#_Toc477349014)

[Entités 5](#_Toc477349015)

[Services 6](#_Toc477349016)

[Adaptateurs 6](#_Toc477349017)

[Interface 6](#_Toc477349018)

[Outils utilisés 6](#_Toc477349019)

[Git 6](#_Toc477349020)

[TypeScript 6](#_Toc477349021)

[Visual Studio Code 7](#_Toc477349022)

[MongoDB 7](#_Toc477349023)

[Node Packet Manager 7](#_Toc477349024)

[Dépendances de l’application 7](#_Toc477349025)

[Dépendances pour le développement 7](#_Toc477349026)

[Serveur 7](#_Toc477349027)

[Présentation 7](#_Toc477349028)

[Dépendances pour la production 7](#_Toc477349029)

[Serveur 7](#_Toc477349030)

[Présentation 8](#_Toc477349031)

[Documents annexes 8](#_Toc477349032)

# Architecture globale

L’application de réservation de la M2L sera sous divisé en deux sous projet : le « frontend » et le « backend ».

Le backend a pour rôle l’implémentation des règles métiers. Il est également en charge de l’accès, la sauvegarde, la suppression et la modification des données persistées ainsi que de leur cohérence.

Le frontend se chargera de la présentation de ces données à l’utilisateur final, et lui offrira une interface graphique permettant d’interagir avec les règles métier offertes par le backend. Ceci correspond à la partie visible et accessible aux utilisateurs de l’application.

Pour réaliser ces deux projets, une architecture a été choisie dont les objectifs principaux sont les suivants :

* Avoir la capacité d’effectuer des tests sur la logique de l’application uniquement
* Obtenir un produit dépendant le moins possible des choix d’implémentations et de librairies faits.

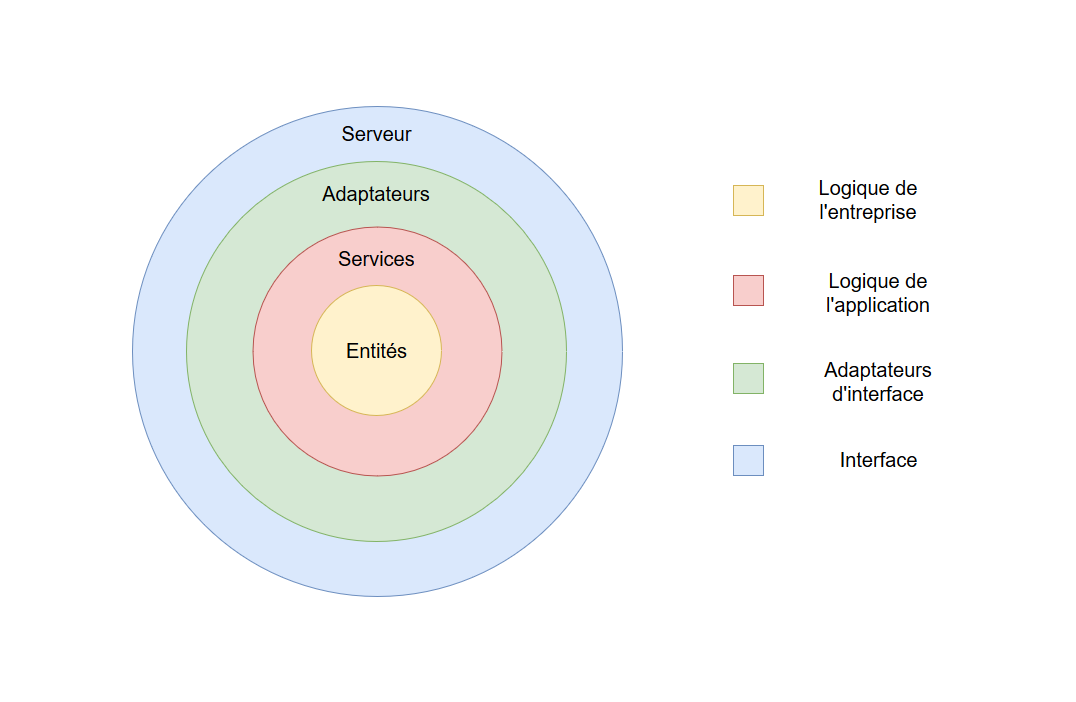
Dans cette optique, les projets sont séparés en une liste ordonnée de couches dont les rôles sont bien définis :

1. Logique de l’entreprise
2. Logique de l’application
3. Adaptateurs d’interface
4. Interface

Les développeurs devront, afin que les objectifs soient atteints, se soumettre à une règle d’or : **une couche ne doit avoir uniquement conscience que de l’existence des couches sous-jacentes**. Par exemple, la couche logique de l’application peut avoir conscience de l’existence de la couche de logique de l’entreprise, mais pas des couches d’adaptateurs ou d’interface.

A chacune de ces couches correspond un composant ou une liste de composants dans ces projets :

1. Les entités se chargent de mettre en œuvre la logique liée à l’entreprise
2. Les services mettent en œuvre la logique de l’application en implémentant les cas d’utilisation
3. Les adaptateurs se chargent de réaliser les actions définies par les services et du formatage des messages entre l’interface et les services
4. Le serveur ou la présentation définissent l’accès extérieur à l’application ou interface



Représentation graphique de l’architecture globale des sous projets

Le développement se fera en programmation orientée objet et respectera les principes SOLID développé si dessous :

* ‘S’ pour indiquer une responsabilité unique, indiquant qu’une méthode ou une fonction ne doit avoir qu’une et une seule responsabilité.
* ‘O’ pour signifier que la classe doit être ouverte à l’extension mais fermée à la modification.
* ‘L’, pour « Liskov Substitution Principle » qui précise que qu’une instance puisse être modifiée, altérée ou remplacée sans pour autant compromettre la cohérence du programme.
* ‘I’ pour « Interface Segregation Principle » pour spécifier que l’application doit dépendre de ses dépendances, pas de ses implémentations.

## Entités

La première couche de cette modélisation est composée d’entités. Ces dernières définissent les formats des modèles de données et sont en charge d’implémenter les vérifications les plus basiques. Ces structures sont valables pour toute application développée pour la M2L, elles ne sont donc pas spécifiques à l’application et peuvent être réutilisés. Cette couche n’inclut aucune librairie extérieure.

Par exemple, on y définira un utilisateur : un objet contenant des propriétés d’un certain type (« name » de type chaîne de caractère, « mail » de type chaîne de caractère) qui doivent répondre à certains critères (« mail » doit être au format d’une adresse mail par exemple).

## Services

La couche de services, elle, définit la logique de l’application. On y trouvera une collection de services contenant des méthodes encapsulant chacun des cas utilisateurs. Les composants de cette couche, afin d’être indépendant de tout choix (librairie, interface, base de données, …), sont purement logiques et utilisent les composants de la couche d’adaptateurs pour effectuer les processus définis dans la logique. On utilisera le concept de l’inversion de dépendance afin de respecter la règle d’or.

Par exemple, un service backend offrira une méthode permettant la connexion d’un utilisateur au système. Elle vérifiera la présence des champs « mail » et « password », demandera à un adaptateur de lui fournir un utilisateur possédant cet e-mail et ce mot de passe puis vérifiera qu’un utilisateur lui a bien été fourni.

## Adaptateurs

Les adaptateurs d’interface sont en charge de transformer les données du format utilisé par les interfaces vers le format utilisé par les services et les entités. Certains sont utilisés par les services pour effectuer les actions qu’ils y définissent, d’autres permettent d’effectuer la liaison entre l’interface choisie et les cas utilisateurs définis dans les services.

Ces adaptateurs faisant la liaison entre les choix d’interface et de librairies faits, ils sont par nature dépendants de ces choix.

## Interface

Les interfaces correspondent à tous les détails d’implémentation : la base de données, la librairie d’interface web, la librairie utilisée par le server, l’api REST, …

Ils se trouvent dans le cercle le plus éloigné, ce qui implique qu’aucune autre couche n’a connaissance de son existence, et que des changements dans cette partie créeront un risque limité de régression dans le système.

# Outils utilisés

## Git

Git est un logiciel de gestion de versions décentralisé. Il permet à l’équipe de suivre l’évolution des versions et du code source du projet.

## TypeScript

TypeScript est un sur ensemble de JavaScript libre et open source développé par Microsoft. Il améliore et sécurise la production de code JavaScript en permettant un typage statique des variables et fonctions, création de classes et d’interfaces, … Le code TypeScript est transcompilé en JavaScript avant d’être interprété par un navigateur web ou un moteur JavaScript.

Cette technologie a été choisie car le typage statique, les classes et les interfaces sont d’une grande aide dans l’implémentation de l’architecture logicielle choisie.

## Visual Studio Code

Visual Studio Code est un IDE développé par Microsoft pour Windows, Linux et OS X. Il est personnalisable grâce à l’ajout de différents modules, et est optimisé pour coder en TypeScript (auto complétion, vérification des types, …)

## Node Packet Manager

Node Packet Manager (NPM) est le gestionnaire de paquets officiel de Node.js. Il permet de télécharger et gérer les versions de module JavaScript développé par la communauté.

# Dépendances de l’application

## Dépendances pour le développement

### Serveur

#### Nodemon

Nodemon observe les fichiers sources d’une application Node.js et redémarre automatiquement lorsque ces derniers subissent une modification. Il facilite ainsi le développement en permettant aux développeurs de ne pas manuellement relancer le serveur.

### Présentation

#### Webpack

Webpack permet de regrouper toutes les ressources statiques du frontend. Il fonctionne à l’aide de différents chargeurs permettant de gérer un type de ressources : JavaScript, images, sons, html, …

#### SASS

SASS est un langage de génération de feuilles de styles initialement développé par Hampton Catlin et Nathalie Weizenbaum. Il permet d’ajouter de la logique, de déclarer des variables, de factoriser du code, … et sera transcompilé en CSS pour être interprété par le navigateur web.

#### Babel

Babel permet d’utiliser la syntaxe des dernières spécifications du JavaScript (ECMAScript) pour le développement. Ces spécifications n’étant pas implémentées dans tous les navigateurs web, Babel transforme ces récentes syntaxes en une syntaxe compréhensible par ces derniers.

## Dépendances pour la production

### Serveur

#### Node.js

Node.js est une plateforme logicielle libre et événementielle en JavaScript utilisant le moteur JavaScript V8 et implémentant les spécifications CommonJS.

C’est un environnement d'assez bas niveau permettant d'exécuter du JavaScript non plus dans le navigateur web mais sur le serveur.

#### Express.js

Express.js est une librairie Node.js pour la création d’applications web. Il est de base relativement minimaliste mais voit ses fonctionnalités de base étendue par des plugins.

### Présentation

#### VueJS

VueJS est un framework Open Source pour construire des interfaces utilisateurs. Il est principalement utilisé avec JavaScript afin de bénéficier d’une meilleure optimisation.

# Documents annexes

* <https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html>
* <http://blog.cleancoder.com/uncle-bob/2016/01/04/ALittleArchitecture.html>
* <https://en.wikipedia.org/wiki/SOLID_(informatique)>