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 |
| 16/12/2015 | 0.2 | Modification du document | Gaël LEHCHIBI |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table des matières

[Architecture globale 4](#_Toc479354769)

[Entités 5](#_Toc479354770)

[Services 6](#_Toc479354771)

[Adaptateurs 6](#_Toc479354772)

[Interface 6](#_Toc479354773)

[Outils utilisés 6](#_Toc479354774)

[Git 6](#_Toc479354775)

[TypeScript 6](#_Toc479354776)

[Visual Studio Code 7](#_Toc479354777)

[Node Packet Manager 7](#_Toc479354778)

[Dépendances de l’application 7](#_Toc479354779)

[Dépendances pour le développement 7](#_Toc479354780)

[Serveur 7](#_Toc479354781)

[Présentation 7](#_Toc479354782)

[Dépendances pour la production 7](#_Toc479354783)

[Serveur 7](#_Toc479354784)

[Présentation 8](#_Toc479354785)

[API de l’application 8](#_Toc479354786)

[Guide du développeur 8](#_Toc479354787)

[Prérequis 8](#_Toc479354788)

[Premier lancement 8](#_Toc479354789)

[Build du projet 8](#_Toc479354790)

[Lancement d’un build 9](#_Toc479354791)

[Lancement du développement 9](#_Toc479354792)

[Documents annexes 9](#_Toc479354793)

# Architecture globale

L’application de réservation de la M2L sera sous divisé en deux sous projet : le « frontend » (client-app) et le « backend » (client-lib).

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

C:\Users\Charles\AppData\Local\Microsoft\Windows\INetCacheContent.Word\spec_technique.png

Représentation graphique de l’architecture globale des sous projets

Cette architecture est inspirée de la « clean architecture » de [Uncle Bob](https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html) et respecte les principes [SOLID](https://fr.wikipedia.org/wiki/SOLID_(informatique)).

En suivant cette logique, le projet a été décomposé en sous projets : deux communs aux deux projets présenté pour l’épreuve (m2l-core et m2l-node-logger), et quatre spécifiques à ce projet (booking-client-lib, booking-client-app, booking-server-lib et booking-server-app).

## 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 client-lib 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, …

#### 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.

#### MySQL

MySQL est adapté pour un usage gratuit à l’inverse de SQL qui est fourni par Oracle. C’est un langage de requête servant à exploiter les bases de données.

### 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.

# API de l’application

L’accès aux ressources dans le backend depuis le frontend se fait grâce à une API REST accessible via l’URL de base « /api/v1.0.0/ » :

|  |  |  |  |
| --- | --- | --- | --- |
| **Verbe HTTP** | **URL** | **Paramètres** | **Action** |
| POST | /login | credendials | Connecter l’utilisateur |
| POST | /logout | aucun | Déconnecter l’utilisateur |
| POST | /user | user | Enregistrer un utilisateur |
| PUT | /user/:userId | modifiedUser | Modifier le profil d’un utilisateur |
| GET | /room | aucun | Obtenir la liste des salles |
| GET | /room/:roomID/image | aucun | Obtenir l’image d’un salle |
| POST | /booking | booking | Enregistrer une réservation |

Les réponses de l’API REST seront sous la forme :

apiResponse : {  
 data : any,  
 faults : string[]  
}

# Guide du développeur

## Prérequis

Afin de lancer le projet, vous devez avoir installé sur votre machine [NodeJS](https://nodejs.org/en/) en version 6.10. Cela installera également Node Packet Manager (npm). Veillez à ce que npm et node soient dans vos variable d’environnement PATH.

Il vous faudra également lancer un serveur de base de données [MySQL](https://www.mysql.fr/) et une connexion à internet.

## Premier lancement

Si vous ne souhaitez que déployer le site sur la machine, ouvrez une fenêtre de commande dans le dossier PPE1 et exécutez-y « npm run release ». Vous pouvez alors accéder directement à la partie de ce document expliquant comment lancer le serveur.

Sinon, ouvrez une fenêtre de commande dans le dossier PPE1 et exécutez-y « npm run init ». Si vous souhaitez créer la base de donnée, exécutez également « npm run init:db ». Pour remplir la base de donnée de salles, exécutez également « npm run fill:db »

Veuillez noter que la commande « npm run release » ou « npm run init » peut prendre du temps à s’exécuter.

## Build du projet

Afin de générer une version complète du produit, veuillez tout d’abord vous assurer d’avoir exécuté le script « npm run init » comme indiqué dans la partie ci-dessus.

Ouvrez une fenêtre de commande dans le dossier PPE1 et exécutez-y « npm run build ». Chaque sous projet contiendra alors un build dans son dossier « dist ».

## Lancement d’un build

Avant de lancer le projet, assurez-vous d’avoir exécuté les étapes précédentes. Vous devrez également avoir initialisé la base de données. Ouvrez ensuite une fenêtre de commande dans le dossier PPE1 et exécutez la commande « npm start ». Le projet sera accessible à l’url <http://localhost:9090>.

Certaines variables d’environnement permettent de configurer le serveur :

|  |  |  |
| --- | --- | --- |
| **Nom de la variable** | **Description** | **Valeur par défaut** |
| PORT | Port sur lequel le serveur écoute | 9090 |
| ROOM\_IMAGE\_ROOT | La racine du dossier contentant les images des salles | [root]/data/user\_img |
| STATIC\_DIR | La racine du dossier contenant le frontend | [root]/PPE1/booking-client /booking-client-app/dist |
| DB\_HOST | L’hôte du serveur de base de données | localhost |
| DB\_USER | L’utilisateur de la base de données | root |
| DB\_PASS | Le mot de passe de la base de données |  |
| DB\_NAME | Le nom de la base de données | booking\_app |
| LOG\_LEVEL | Le niveau de log souhaité parmi DEBUG, LOG et INFO | INFO |

**[root] représente ici le chemin d’accès au dossier téléchargé sur Dropbox.**

## Lancement du développement

Les scripts suivants aident au développement :

* « npm run dev:core » : lance un build en watch sur m2l-core
* « npm run dev:logger » : lance un build en watch sur m2l-node-logger
* « npm run dev:clib » : lance un build en watch sur booking-client-lib
* « npm run dev:capp » : lance un build en watch sur booking-client-app ainsi qu’un serveur de développement accessible à l’url <http://localhost:9000>.
* « npm run dev:slib » : lance un build en watch sur booking-server-lib
* « npm run dev:sapp » : lance un build en watch sur booking-server-app et lance le serveur

# 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)>