Choix Technique et AGL

# Choix de la technologie

## Type d’application

### Client Lourd

**Avantages :**

* Un seul langage client et serveur
* Communication avec le serveur simplifié
* Rapidité de production
* Plateformes d’exécution rétro-compatibles
* Peut évoluer en client riche pour minimiser les problèmes de déploiement et de maintenance

**Inconvénients :**

* Déploiement plus lourd
* Maintenance et Mises à jour plus lourdes
* Sécurité et sauvegarde dépendantes du poste client

### Client Léger

**Avantages :**

* Coût plus faible dans le cadre d’un parc info
* Maintenance système, application centralisée et immédiatement disponible après chaque mise à jour
* Maintenance utilisateur
* Robustesse, serveur centralisé moins sensible aux pannes qu’un poste client
* Mobilité

**Inconvénients :**

* Langages client/serveur différents
* Partie cliente nécessitant plusieurs langages
* Formation des développeurs à l’intégralité de la pile technique de l’application
* Pas de mode hors-ligne, dépendant de la qualité du réseau
* Evolutions des navigateurs
* Performances en baisse si trop nombreux utilisateurs

### Conclusion : Client lourd

Nous avons donc décidé d’opter pour un client lourd pour les raisons suivantes :

* Le parc informatique dédié à la vente des maisons modulaires consistera en seulement en quinzaine d’ordinateurs portables ce qui réduira les inconvénients de déploiement et de maintenance
* L’application Madera MMB ne nécessite aucun travail collaboratif
* L’application Madera MMB nécessite d’être utilisée en mode hors connexion
* La rapidité de production d’un client lourd offrira une sécurité dans le respect du planning de développement
* Les AGL pour clients offre offrent des modules permettant le travail collaboratif

## Langage de programmation

Pour le choix du langage, nous avons mis en place une grille décisionnelle pour nous aider à déterminer le langage le plus adéquat selon une liste de critères

Pour développer un client lourd notre choix s’est porté entre deux langages : C# et Java

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Critères | Importance | Note C# | Note Java |  |  |  |
| Connaissance du langage | 5 | 2 | 5 |  |  |  |
| Sécurité des données | 2 | 3 | 4 |  |  |  |
| Pérennité | 2 | 5 | 4 |  | **Légende** | **Note** |
| Performance | 3 | 4 | 3 |  | Elevé | 5 |
| Portabilité | 3 | 4 | 2 |  | Haut | 4 |
| Faisabilité | 5 | 4 | 2 |  | Moyen | 3 |
| Déploiement | 3 | 3 | 3 |  | Bas | 2 |
| Cout | 4 | 1 | 3 |  | Faible | 1 |
| Maintenabilité | 3 | 4 | 2 |  |  |  |
| Stabilité | 3 | 4 | 4 |  |  |  |
| Popularité | 5 | 5 | 4 |  |  |  |
| Normalisation | 3 | 5 | 4 |  |  |  |
| Utilisation | 3 | 3 | 4 |  |  |  |

**Total**  **47 44**

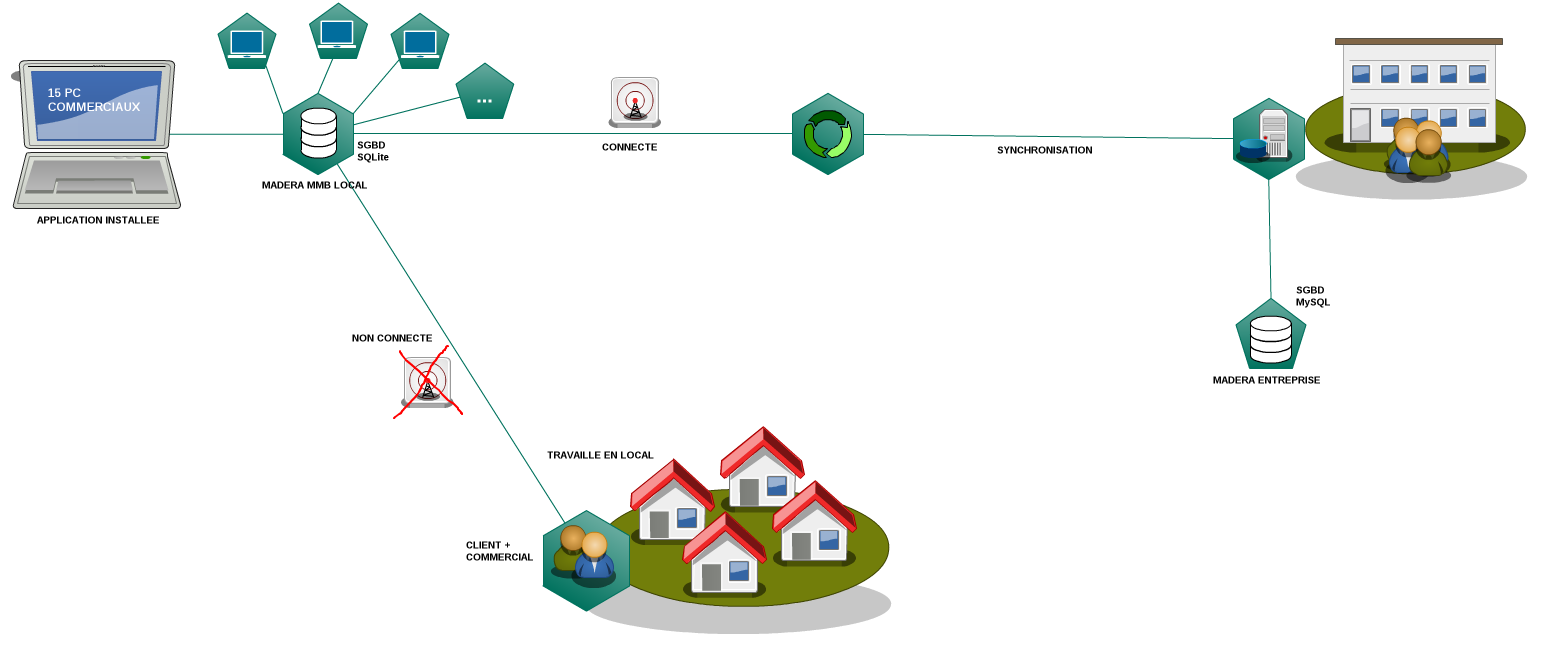
Il en sort que le C# sera le langage choisi du fait de son obtention de la note la plus élevée.

Afin de séparer la conception graphique de l’interface et des traitements de l’application, et pour sa compatibilité native avec le langage C#, nous avons donc choisi d’utiliser XAML pour la partie graphique.

Ainsi l’application sera développée en C#/XAML.

## Système de gestion de base de données relationnelle

Suite à la nécessité d’implémenter une base de données locale à l’environnement de l’application devant se synchroniser régulièrement avec la base de données principale du Bureau d’étude afin d’en récupérer le catalogue produit, le schéma suivant en a découlé :



**➢ Choix SGBD ?**

Dans notre configuration proposée, nous devons effectuer un choix sur les bases de données. A savoir, que d'après le schéma, nous envisageons de mettre en place, une base de données dites

« Embedded », c'est à dire qui sera intégré localement et utilisable sur une machine par l'utilisateur, ainsi qu'une seconde base dites « master », qui sera la base de données principal de l'entreprise

MADERA. Nous prévoyons d'effectuer des synchronisations automatisées à chaque démarrage de l'application. En mode connecté, les données stockées sur la base de données master seront synchronisé sur la base de données locale. En mode sans connexion, l'application pourra fonctionner simplement avec la base de données locale et y stocker les nouvelles données qui seront créés.

**➢ Un coup d'œil sur les différents types de SGBD No SQL ?**

On peut classer celle-ci en 4 catégories ou 4 grandes familles :

1. les bases de données orientées doc : MongoDB, CouchDB, RavenDB, MarkLogic …

2. les bases clé/valeur : Aerospiken, Redis, Riak, Voldemort ...

3. les bases en colonnes : Accumulo, Cassandra, Hbase, HyperTable ...

4. les bases orientées graphes : InfiniteGraph, Neo4j, FlockDB, Giraph …

Les bases de données orientées document sont idéalement adapté pour les applications en évolution permanente. L'ensemble des données sont décrites dans une structure de document au format JSON ou XML dans la plupart des cas. On peut interroger la BDD sur tous les composants du schéma défini. L’avantage est de pouvoir récupérer, via une seule clé, un ensemble d’informations structurées de manière hiérarchique. Pas de jointures.

Les bases clé/valeur offrent un bon niveau de performance lorsqu'il s'agit de faire des recherches sur des jeux de données relativement simple. On ne peut interroger la BDD que sur sa clé. Bien adapté aux applications nécessitant un accès rapide aux données.

Les bases en colonnes offrent un haut niveau de performance lorsqu'il s'agit de traiter des données à grande échelle. Le nombre de colonnes est dynamique. Idéalement adapté au stockage de listes

(Messages, billets, commentaires ...)

Les bases orientées graphes offrent une solution qui privilégie les relations entre les valeurs et stocke les données en utilisant une structure des graphes en nœuds, en relations et en propriétés. Ce type de base de données est surtout utilisé pour les mises en œuvre à l'échelle du Web, du Big Data et à caractère analytique. Pour simplifier, elles permettent de gérer des relations multiples entre des objets.

**➢ SQLite – les avantages ?**

Licence : OpenSource et gratuit

Serveur : Pas de serveur

SQL : moteur SQL simplifié

Disponibilité : Linux, MacOSX, Windows, Unix, BSD

Environnement : Multiplate-forme

Installation & Configuration : Très facile à installer - 0 configuration requise ou presque

Type : Système de gestion de base de données relationnelle

Migration : Possibilité de passé à MySQL par la suite

Synchronisation : Plus facile lorsqu'il y a 2 BDD SQL

Fonctionnalités : Majeure partie des fonctionnalités classiques.

Version : La dernière version SQLite3 offre plus de performance (auto incrément, meilleur stockage, support clés étrangères)

Mais aussi : Il est portable, performant surtout en lecture et très simple. C'est à dire, qu'il y a aucune manipulation à faire. La base est stockée dans un seul fichier. Elle est incluse directement dans l'application.

**➢ SQLite – les inconvénients ?**

Fonctionnalités : Ils manquent certaines fonctionnalités importantes telles qu’ALTER TABLE,

GRANT, REVOKE, …

Limitations : Pas de reconnaissance de transactions imbriquées ni les « triggers ». Pas de réplication.

Sécurité : Minimum. Pas de notion de rôles, de hiérarchisation de groupes.

**Conclusion :**

Nous avons donc en définitive choisi **SQLite** pour la gestion de la base de données locale à l’application et sera donc la base de donnée « esclave » de la partie client.

La partie serveur, aura la charge d’héberger la base de donnée du bureau d’études.

Celle-ci, contenant un plus gros volume de données, devra être gérée sous **MySQL** . Cela permettra d’utiliser les fonctionnalités de rôles et de hiérarchisation des groupes pour administrer

# Atelier de génie logiciel

* Pourquoi un AGL ?

Pour optimiser la productivité dans le développement de l’application en C# XAML, il nous a paru nécessaire de travailler avec un outil permettant de documenter automatiquement le programme, et de maintenir en permanence à jour cette documentation, et, ce tout au long de sa conception.

L’utilisation d’un AGL facilitera la collaboration entre les différents programmeurs ainsi que la maintenance ultérieure de la solution en partageant les mêmes méthodes.

* Pourquoi Visual Studio ?

Plateforme de développement logiciel édite par Microsoft, Visual Studio est un ensemble complet d'outils de développement permettant de générer, entre autres, des applications bureautiques et de partager plusieurs technologies telles que C# et XAML, nous concernant.

Il intègre de nombreuses fonctionnalités telle que le travail collaboratif et permet une production rapide des applications bureautique.

Visual Studio se trouve donc être le candidat parfait pour devenir notre AGL dans le développement de l’application Madera.

* Pour quoi 2013 ?

Visual Studio 2013 est connu à ce jour pour être l’une des versions les plus stables de Visual Studio. Elle offre la possibilité de développer des applications multiplateformes. Fonctionnalité que ne propose pas la version précédente.

Egalement, Visual Studio 2015 est une version réputée pour ne pas être stable et offrant des fonctionnalités dont notre équipe n’a pas l’utilité.

**Conclusion :**

Notre équipe utilisera donc l’AGL Visual Studio 2013 dans le développement de l’application MaderaMMB.