****

Année Universitaire : 2016 – 2017

*Réalisé par :*

*Mohamed Sidatt Alem*

*Brahim Mohamed Baheida*

*Encadré par :*

Dr. Ethmane ould Ebbe

MEMOIRE

En vue de l’obtention

De Licence Professionnel en Informatique et développement

*d’Entreprises*

*République Islamique de Mauritanie*

*Honneur, Fraternité, Justice*

*Ministère chargée de l’Enseignement Supérieur*

*Institut Supérieur de Comptabilité d’Administration d’Entreprises*

# Dédicace

​ Nous dédions ce travail à :  
Nos mères, sources de tendresse et d’amours pour leurs soutiens tout le long de notre vie scolaire.  
Nos pères, qui nous ont toujours soutenus et qui ont fait tout possible pour nous aider.  
Nos frères et nos sœurs, que nous aimons beaucoup.  
Notre grande famille.  
Nos chers ami (e) s, et enseignants.  
Tous qui ont collaboré de près ou de loin à l’élaboration de ce travail.  
  
Que dieu leur accorde santé et prospérité.

# Remerciement

# Résumé

# Introduction

# Contexte du mémoire

## Présentation du projet :

### Description du projet :

### Objectif :

## Travail Demandé :

## Bilan :

# Analyse et conception :

## Etude prealable :

## Modelisation :

## Les principes S.O.L.I.D :

SOLID est un acronyme introduit au début des années 2000 par Robert Cecil Martin connu par Uncle (l’oncle) Bob, résume 5 principes clés à appliquer en POO pour produire du code facile à maintenir.

### S – Single Responsability Principle (SRP):

Chaque objet est en charge d'une seule *responsabilité*, laquelle doit être complètement encapsulée dans la classe.

Responsabilité = raison de changer (R. C. Martin).

Une classe ne doit être susceptible de changer que pour une seule raison.

### O – Open/Closed Principle (OCP):

Les entités logicielles (classes, modules, fonctions, etc.) doivent être ouvertes aux extensions, mais fermées aux modifications.

On utilise au maximum les interfaces, figées. Mais elles peuvent être implémentées librement et augmentées par héritages.

### L – Liskov Substitution Principle (LSP) :

Concrètement, là ou un objet de type **T** est attendu, on doit pouvoir passer un objet de type **Y** sans que le comportement dudit objet soit changé. C’est-à-dire que tout invariant inhérent à la classe **T** être respecte dans ses classes filles.

LSP permet d’assurer que notre hiérarchie suit un fonctionnement logique et cohérent.

### I – Interface Segregation Principle (ISP):

Une interface doit ne comporter que des méthodes en rapport avec l’interface elle-même, de façon à ce que les clients d’une interface ne connaissent que les méthodes en rapport avec cette interface.

### D – Dependency Inversion Principle (DIP):

Dans les architectures classiques, les composants de haut niveau dépendent des composants de bas niveau sur lesquels ils reposent. Le **DIP** établit au contraire que :

* Les modules de haut niveau ne doivent pas dépendre des modules de bas niveau. Les deux doivent dépendre de l’abstractions.
* Les abstractions ne doivent pas dépendre de détails, mais les détails doivent dépendre des abstractions.

## Solution :

La solution qu’on a proposée basée sur le platform .NET, répond aux besoins cités précédemment et offre des avantages supplémentaires :

### Interaction :

La solution offre un environnement d’échange d’information et de collaboration entre les utilisateurs, les étudiants peuvent ajouter des documents, discuter entre eux. Les professeurs peuvent ajouter des documents, envoyer des messages aux étudiants. L’administration capable d’ajouter des avis, des résultats, la liste des étudiants, activer et désactiver des comptes.

### Sécurité :

Tout au long du développement de la solution on a considéré les recommandations de la communauté de sécurité des applications web. Le site OWASP offre une liste des vulnérabilités des applications web chaque année. Cette considération va être le sujet de la partie sécurité (page xxx).

### Etre responsive :

L’utilisation du Framework Bootstrap, de la bibliothèque JQuery et d’autres nous permet de réaliser une interface d’utilisation responsive.

### Optimisation des ressources :

La programmation générique, et l’utilisation de quelque patron de conception (Design Pattern) nous servi à optimiser les ressources par la diminution de nombre des codes pour réaliser des taches similaires en plusieurs classes, et le nombre des instances dans la RAM.

### Maintenabilité :

La maintenance des logiciels est divisée en trois types :

#### Maintenance corrective :

Elle consiste à corriger les défauts de fonctionnement ou les non-conformités d'un logiciel.

#### Maintenance adaptive :

Sans changer la fonctionnalité du logiciel, elle consiste à adapter l'application afin que celle-ci continue de fonctionner sur des versions plus récentes des logiciels de base, voire à faire migrer l'application sur de nouveaux logiciels de base (un logiciel de base étant un logiciel requis pour l'exécution d'une application ; exemples : système d'exploitation, système de gestion de base de données).

#### Maintenance évolutive :

Cela consiste à faire évoluer l'application en l'enrichissant de fonctions ou de modules supplémentaires, ou en remplaçant une fonction existante par une autre, voire en proposant une approche différente.

Notre solution suivre les principes S.O.L.I.D qui vont être décrites au chapitre suivant. Elle est divisée en plusieurs couches chaque couche est dédié pour des taches spécifiques ce qui facilite l’accès en cas d’erreurs aux codes responsables et les corriger. Elle permet aussi l’adaptation et l’évolution des fonctionnalités.

# Développement

## Couche D.A.L

### Entity Framework

### Log4Net

### Repository Pattern

## Couche B.L.L

### Contraintes

### Complexité

## Couche Web

### Web APIs

### NInject

### Design

### MVC

# Sécurité

# Conclusion

# Annexe

# Biographie