Projet Interface Homme Machine

Where Tact

**Dans le cadre du cours de 5ieme Année de Nicolas ANDRE à l’ESIEA**

CLAEYS Emmanuelle DECHELETTE Nicolas HEWINSANDHELOS

Table des matières

[But du projet 3](#_Toc401834141)

[Rappel du sujet 3](#_Toc401834142)

[Architecture générale 4](#_Toc401834143)

[CRUD sur les objets 5](#_Toc401834144)

[Créer des objets 5](#_Toc401834145)

[Action sur les objets 5](#_Toc401834146)

[Déployabilité et ergonomie 5](#_Toc401834147)

[Les questions. 6](#_Toc401834148)

[Architecture du code 7](#_Toc401834149)

[Implémentation 8](#_Toc401834150)

[Réflexion après projets. 10](#_Toc401834151)

[Installation de l'environnement. 10](#_Toc401834152)

[Apprentissage de nouveaux outils 10](#_Toc401834153)

[Travail en groupe 10](#_Toc401834154)

[Temps impartis 10](#_Toc401834155)

[Conclusion 11](#_Toc401834156)

# But du projet

## Rappel du sujet

**Objectif :** Concevoir et développer l’IHM d’une application Web de gestion de contact.

**Projet Interface graphique :** Développer une petite application web avec un Framework Imposé « Spring MVC » permettant de générer une interface graphique. Le travail sera concentré sur l’interface Homme-Machine. Il faut créer une application qui permet de créer un contact, créer une adresse, de faire des recherches, mises à jour et suppressions.

**Pour le CV :** Application web en Java avec Spring MVC de gestion de contacts, déployée sur le Cloud, dont le code se trouve sur Git Hub.

Insérer des contacts (données qui paraissent réelles) au démarrage pour l’effet démo !

Model :

* **Contact**
* **Adresse**

Services : Créer une ou deux class service (adresse et contact)

* **Création de contact**
* **Créer une adresse**
* **Différencier Adresse de Livraison Et de Facturation**
* **Possibilité d’associer un contact à une adresse**
* **Modifier Contact**
* **Modifier adresse** (qu’est-ce qui se passe quand on supprime une adresse qu’un contact possède) ;
* **Suppression** contact / adresse ;
* **Recherche** (attributs : Nom, Age etc.) – Polymorphisme avec plusieurs arguments possible ;
* **ETC.**

Pas de base de données ! On simule la base de mémoire avec des Maps, des Lists etc.

Déployer sur un Appache TomCat

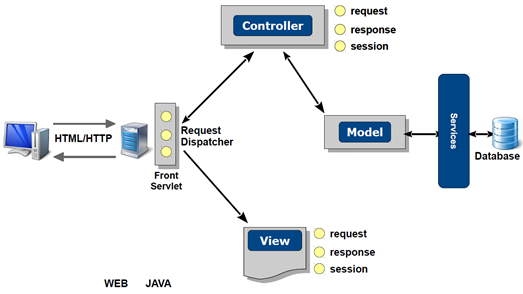
Faire des recherches de CSS sur Internet pour une meilleur Ergonomie.

Ergonomie simple et intuituve

Plateforme qui permet de faire tourner du Tomcat.

## Architecture générale

Le projet doit suivre l’architecture MVC suivante :



*Architecture MVC – Cour IHM ESIEA N.André 2014*

# CRUD sur les objets

## Créer des objets

Pour commencer, nous avons défini les objets suivants :

Objet Contact

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| name | nom | prénom | adresse mail | date de naissance | actif? |
| type | string | string | string | date | boolean |
| regex | X | X | format adresse e-mail | format de date jj/mm/dddd | format boolean |

Objet Adresse

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| name | numéro | rue | code postal | ville |
| type | string | string | string | string |
| regex | X | X |  | **X** |

Pour la gestion de stockage en mémoire nous allons utliser java.util.map, une interface entre les key et les value.

## Action sur les objets

Pour la création de l’objet, nous avons vérifié via les regex les entrées de l’utilisateur pour enregistrer une donnée. Les données sont consultables, modifiable et offre la possibilité d’être détruite par l’utilisateur, mais protège le code via une architecture MCV.

En accord avec les obligations du sujet, l’utilisateur peut avoir plusieurs adresses : son adresse principale qui est l’adresse de facturation et une adresse secondaire qui peut être une adresse différente pour la livraison.

## Déployabilité et ergonomie

L’application doit être déployable sur Apache Tomcap pour faire XXXXX et Google app Engine pour XXX.

L’ergonomie doit être intuitive. Nous avons fait un schéma de la page et nous avons voulu utiliser des termes simples et courts dans un webdesign inspiré de Windows 8

L’application devait présenté des notifications d’action (exemple “contact ajouté” ainsi que des notifications d’erreur.

### Les questions.

Pour la conception de l’architecture du code nous avons dû nous poser les questions suivantes :

*Est-ce qu’une adresse de livraison peut être une adresse de facturation ?*

* Oui

*Laisse-t-on la possibilité de changer les champs utilisateurs à partir d'une adresse?*

* Non

*Laisse-t-on la possibilité d'ajouter une adresse en base à partir de la page d'un utilisateur?*

* Non, un utilisateur est identifié via un ID unique générer automatiquement, mais il est déterminé par son adresse Email. Pour l’adresse on regroupe les champs d’information et on vérifie qu’il n’y a pas de doublons sur l’ensemble.

*Est-ce qu’on utilise le principe de pages par composants pour mettre les 2 formulaires sur une même page ?*

* Oui, car on pourrait créer ainsi plus rapidement la base de données, on entre un contact, on valide on reste sur la page avec les infos contacts grisés, on remplit les adresses les unes après les autres avec le bouton "ajouter une autre adresse". pour ajouter un autre contact, on appuie sur le bouton "ajouter un autre contact".

*Y a-t-il une authentification? Est-ce que deux utilisateurs manipulent les mêmes données?*

* Oui, dans les délais impartis nous n’avions pas le temps de faire une interface administrateur.

*Adapte-t-on l’application web au format tablette mobile?*

* Non. Nous avons essayé d’inclure la bibliothèque css bootscap mais nous avons dû arrêter cette démarche suite à un problème de temps et des bugs provoqués (il fallait redéfinir les balises jsp)

*Est-ce qu’une adresse est forcément liée à un contact ?*

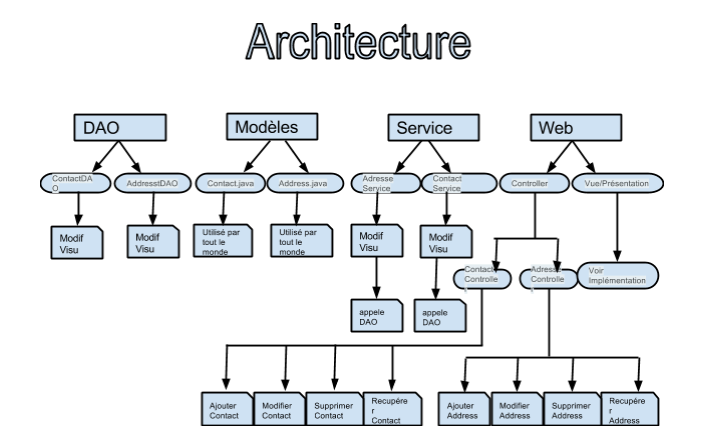
* Non, car comme le sujet le précise, une adresse et un contact sont deux éléments qui peuvent être indépendants.

*Est-ce qu’on la détruit si on supprime l’utilisateur ?*

* Si on relie une adresse à un utilisateur, l’adresse disparait comme tous les attributs du contact.

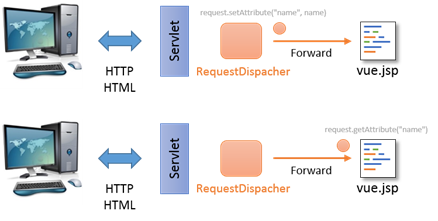
# Architecture du code

Notre architecture respecte le principe du  design pattern MVC. On répond de manière structurée à une requête HTTP en distribuant les responsabilités entre le modèle (module métier), la vue (responsable de l’affichage) et le contrôleur (fais communiquer la vue avec le modèle). L’utilisateur va agir avec le contrôleur.



# Implémentation

Côté Vue, la vue est commune à toutes les pages pour permettre de respecter le principe de dépendance des couches en suivant  ce schéma :



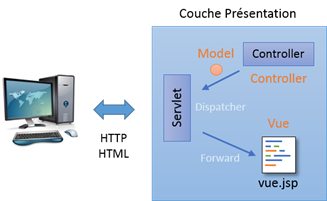
Nous avons à notre disposition un header et un footer pour la gestion visuel de nos pages.

En naviguant dans notre application, nous avons les pages suivantes :

* Accueil
* Formulaires
* Index
* Page Objets

Coté controller

Le controller agit avec le servlet de la façon suivante :



Nous devons donc

* Créer des connecteurs qui lient les différents boutons aux différentes pages. Nous utilisons donc les éléments suivant:
  + Accueil
  + Formulaire contact
  + Formulaire address
  + Index contact
  + Index adresse
  + Contact by ID
  + Address by ID
* Créer la fonction qui va ajouter des données en base
  + Utilisation des regex
* Créer la fonction qui va lire les données en base
  + Visualiser un seule objet par identifiant
  + Visualiser tous les objets
  + Visualiser les objets répondant à certaines caractéristiques
  + Vérification de la non existence d’un objet en base
    - Contact par adresse e-mail
    - Adresses par l’ensemble des champs
* Créer la fonction qui vas modifier les données en base
  + Utilisation des regex
* Créer la fonction qui vas supprimer les données en base
* Créer la fonction qui va modifier les lien entre les données en base pour chaque création, modification et destruction.
* Implémenter les vérification de champs en xml afin d’effectué les vérifications avant l’envoi des données au serveur

# Réflexion après projets.

## Installation de l'environnement.

Le premier problème que nous avons rencontré était l’installation de Maven. Nous avons mis pratiquement deux semaines à pouvoir faire fonctionner maven sur toutes les machines. Installer maven  car il permettait d'automatiser certaines tâches :

- La compilation, tests unitaires et déploiement des applications qui composent le projet

- De gérer des dépendances vis-à-vis des bibliothèques nécessaires au projet

- De générer des documentations concernant le projet

Il fallait gérer différentes version du JDK et paramétrer correctement le Path ce qui nous à pris de longues heures. Cela montre bien que l'environnement de travail peut créer une réelle barrière au déroulement d’un projet

## Apprentissage de nouveaux outils

Au cours du projet nous avons appris Spring MVC, TomCat, Angular et  l’utilisation  de Google App Engin. Nous avons aussi regardé des framworks qui nous permettait de concevoir de façon plus optimisée notre application (Bootstrap par exemple). Si on prend l’exemple de Bootstrap nous n’avions pas assez de temps pour apprendre et mettre en application ce framwork. Lors d’un projet il faut prendre en compte les différents Framwork et autres éléments pour concevoir notre code de façon à pouvoir les intégrer. En fin de projet il devient de plus en plus difficile d’intégrer de nouvelles idées car elles demandent parfois de reprendre complètement notre code.

De plus l’apprentissage des nouveaux outils demande du temps qu’il ne faut pas négliger lors du planning avant-projet.

## Travail en groupe

Nous avons utilisé GitHub pour faire notre développement. GitHub est un bon outil Open Source mais gère mal les merges. Nous devons reprendre manuellement le code et corriger les doublons de versions. Le controller, les modèles et la vue communique entre eux et il est difficile de se répartir le travail au même moment sans touché aux classes respectives. Le travail simultané est donc perturbé par les différents Push.

## Temps impartis

Cette année contrairement aux années précédentes l’administration a imposé de nombreux examens sur table et les enseignants ont gardé les projets qu’ils imposaient tous les ans. La promo a eu le même problème à savoir gérer 5 projets à rendre pour la fin du mois d’octobre et des examens à la même date. Dans l’urgence nous avons dû faire rapidement nos projets et cela c’est ressentis sur la qualité même si nous avons fait au mieux. Un autre inconvénient des projets en urgence est de dispatché les responsabilités à ceux qui ont déjà de l’expérience dans les tâches alors que pédagogiquement les étudiants devraient pouvoir se former au mieux dans des domaines qu’ils ne maitrisent pas.

# Conclusion

Malgré ces difficultés l’apprentissage et la mise en pratique de l’architecture MVC nous a intéressée. Elle montre comment fonctionnent les projets de développement industriel et nous permet de nous projeté pour nos prochains stages. Apprendre des nouveaux éléments comme Angular nous rappelle qu’il faut se tenir toujours au courant des nouveaux outils de développement pour faire une programmation optimale.