

CONTRÔLER SA MAISON A PARTIR D'UNE INTERFACE WEB

Université de Montpellier II

Rapport de projet informatique de l'Unité d'Enseignement
GLIN405 de la Licence informatique 2^e année effectué au cours
du second semestre.

HEDLI-BASSOUMI
FERNANDEZ-SHAKHLOUL
SIMON-ROSEAU
CHOQUET-BARTHÉLÉMY

4 juin 2013

Table des matières

1	Introduction	3
2	Contexte et technologies	3
2.1	Choix des outils de développement	3
2.2	Répartition du travail	4
3	Analyse	5
3.1	Diagramme des cas d'utilisation	5
3.2	Diagramme de séquence	8
3.3	Diagramme de classe	10
4	Réalisation	14
4.1	Modèle	14
4.2	Vue	17
4.2.1	1 ^{er} test	17
4.3	Contrôleur	22
4.3.1	Première version	22
4.3.2	La version officielle	24
5	Prévisions	26
6	Conclusion	27
7	Manuel d'utilisation	27
7.1	Entrée sur la plateforme de Domo Project	27
7.2	Agir immédiatement sur la maison	27
7.2.1	Ajouter un objet de la maison	27
7.2.2	Supprimer/Modifier un objet de la maison	27
7.3	Protocole	27
7.3.1	Créer un protocole	28
7.3.2	Supprimer un protocole	28
7.4	Pièce	28
7.4.1	Créer une nouvelle pièce	28
7.4.2	Modifier une pièce	28
7.4.3	Supprimer une pièce	28
8	Remerciements	29
9	Annexe	29

1 Introduction

Étudiants en deuxième année de licence informatique, nous avons choisi comme TER le projet domotique.

Notre projet est de concevoir une interface Web fonctionnelle qui permette de contrôler sa maison à distance. Celle-ci sera intuitive, facile d'utilisation pour le client et moderne.

Nous avons choisi ce projet pour travailler en JAVA ¹, car c'est un langage au programme ce semestre et le choisir en projet favorise donc notre apprentissage. De plus nous avons été séduit par sa modernité, son utilité ainsi que sa portabilité.

En effet, de plus en plus de maison s'équipent de matériel permettant de la contrôler. Cette technologie offre plus de sécurité, grâce par exemple, à une fonctionnalité qui ferme les volets et les ouvre chaque jour durant une absence du propriétaire, pour simuler une présence dans la maison.

2 Contexte et technologies

La domotique est un ensemble de technologies telles que la télécommunication, l'informatique et l'électronique, utilisées dans le but de centraliser le contrôle de toutes machines présentes dans une entreprise ou bien une maison. La domotique a pour but d'améliorer le confort et la sécurité de l'habitat en facilitant la gestion d'énergie, donc de diminuer le coût de ce dernier et d'optimiser l'éclairage et le chauffage. Celle-ci vise aussi à apporter une compensation des situations de handicap et de dépendance de certaines personnes.

La domotique est basée sur la mise en réseau des différents appareils électriques de la maison, contrôlés par une « intelligence » centralisée programmable par des modules embarqués ² ou bien une interface micro-informatique (écran tactile, serveur, etc.).

Diverses interfaces de contrôle sont possibles : un ordinateur, un téléphone portable ou un smartphone, une tablette tactile ou encore une télécommande.

Les interfaces permettant de contrôler le serveur sont nombreuses, il y a notamment des surfaces de contrôle possédant de nombreux boutons (télécommandes, souris, clavier), des interfaces tactiles associés à un logiciel ou une interface Web, ou encore des moyens plus humains tel que la voix recueillie par des microphones ou bien par les gestes (détecteur de mouvement par exemple pour la lumière).

Concernant le mode de transmission, plusieurs possibilités existent : les ondes radio ayant l'avantage de pouvoir être émises sur de longues distances malgré des obstacles, l'infra rouge ayant une transmissions de données à courte distance sans obstacle, le courant porteur en ligne (CPL) permettant le transfert de données numériques via le réseau électrique grâce au branchement des modules ou encore d'autres liaison filaire (bus ou Ethernet) qui ont l'avantage de porter sur l'utilisation d'un réseau filaire structuré déjà existant.

2.1 Choix des outils de développement

L'architecture du projet est basée sur le patron de conception Modèle-Vue-Contrôleur³ car il permet une bonne répartition des technologies. Tout d'abord, la vue offre au client une interface Web permettant de contrôler sa maison facilement mais aussi d'informer le client des changements de l'état de sa maison, en

1. Langage de programmation orienté objet développé par Sun ayant l'avantage d'être facilement portable

2. sorte de prise-adaptateur qui sont des passerelles domestiques

3. Le MVC est un design pattern (patron) est un modèle destiné à répondre aux besoins des applications interactives en séparant les problématiques liées aux différents composants au sein de leur architecture respective.

cours, par le biais de différents "warnings". La partie contrôleur permet de faire les vérifications relatives à la base de données, ainsi que de gérer les requêtes des clients et de les exécuter en mettant à jour ou en récupérant les données contenues dans le modèle. Quand à ce dernier, il est séparé en deux parties : les JavaBeans⁴ permettant de faciliter la gestion des données relatives aux objets et aux pièces de la maison et la BDD⁵ qui contient ces données.

Nous avons choisi la programmation en Java plutôt que le PHP⁶, pour approfondir nos connaissances du Java mais aussi pour découvrir d'autres technologies notamment les JavaBeans, le JDBC, le JSP⁷ et le JavaServlet⁸. Le Java s'applique bien au projet, mais complique la partie Contrôleur-Vue par rapport au PHP.

Le JDBC⁹ est utilisé par le contrôleur pour communiquer avec la base de données où seront stockées les informations relatives à la maison et qui sera écrite en langage SQL¹⁰. Afin de manipuler notre base de données, nous utiliserons le SGBD¹¹ MySQL.

Le JavaBeans nous sert à regrouper tous les attributs des objets encapsulés, et peut définir d'autres attributs si besoin au sein du Modèle. De plus, il est possible d'appeler des méthodes implémentées avec des notions d'héritage et de polymorphisme.

Enfin l'interface client sera codée en HTML¹² pour la conception de la page web et la mise en forme sera facilitée grâce au CSS¹³ et notamment avec le framework CSS de Twitter, Bootstrap.

2.2 Répartition du travail

Nous avons réparti le travail selon la structure du MVC. Anthonin Barthélémy et Brett Choquet étant motivés par la vue, le chef de projet Jonathan Fernandez leur a laissé s'occuper de cette partie. Ensuite, Mina Shakhloul et Thin-Hinen Hedli ont été affectés sur la partie contrôleurs. Clément Simon, Julia Bassoumi et Yann Roseau ont été affectés à la partie modèle. Cette partie comprenant à la fois les Beans Java et la BDD.

4. technologie écrit en langage JAVA permettant l'encapsulation d'objets en un seul ce qui permet de représenter une entité plus globale

5. base de données

6. Hypertext Preprocessor est un langage de programmation impératif utilisé pour produire des pages Web dynamiques

7. JavaServer Pages est une technique basé sur Java permet de créer dynamique du code de type page web

8. Interface de programmation utilisé par des classes Java qui permettent de créer dynamiquement des données au sein d'un serveur HTTP, les servlets

9. JavadataBaseConnectivity

10. Structured Query Language est un langage servant à effectuer des opérations sur des bases de données

11. Système de Gestion de base de Données

12. Hypertext Markup Language est un langage de balisage permettant de représenter le format de données conçu pour les pages web

13. Cascading Style Sheets est un langage de mise en forme pour les pages Web

3 Analyse

3.1 Diagramme des cas d'utilisation

Nous avons commencé par réaliser un diagramme des cas d'utilisation. Ce dernier est indispensable pour organiser le travail et se mettre d'accord sur la conception du projet.

Le premier diagramme n'était pas parfait, il comportait des erreurs de fonctionnalités et d'organisations (voir annexe). L'inclusion "protocole", de la tâche principale "contrôle objet", est indispensable en cas de contrôle d'objet ou de planification. La "Caractéristique de l'objet" correspond en fait à la tâche principale "Contrôle objet" abstraite et fait appel à la modification de ses fonctionnalités par la tâche "fonctionnalités" qui est ici une spécialisation.

Cas d'utilisation : Contrôler sa lumière (un objet).

Mr.X veut contrôler la lumière de sa salle de bain.

1. Contrôle de la fonctionnalité de l'objet.

Cas d'utilisation : Créer un protocole.

1. Création d'un protocole.

Cas d'utilisation : Faire une planification.

Mr.X veut créer une planification.

1. Créer un protocole

Le diagramme définitif présente l'une des tâches principale *Contrôler un objet*, comme étant abstraite et qui permet de créer des protocoles.

Ces actions impliquent l'identification du client mais aussi d'accéder et de modifier les caractéristiques de l'objet en question et par conséquent la sélection de l'objet et de la pièce associée. Accessoirement la modification et la suppression d'un objet est possible, la sélection de l'objet et de la pièce est alors impérative.

L'autre tâche principale est la gestion des règles de sécurité qui se fait par la spécification des normes de sécurité, ainsi le client pourra régler la température maximale du chauffage.

Cas d'utilisation : Manipuler les caractéristiques d'un objet.

1. Sélectionner une pièce
2. Sélectionner un objet
3. Manipuler les caractéristiques de l'objet

Cas d'utilisation : Créer un objet.

1. Manipuler les caractéristiques de l'objet
2. Créer un objet

Cas d'utilisation : Supprimer un objet.

1. Manipuler les caractéristiques de l'objet
2. Supprimer un objet

Cas d'utilisation : Consulter l'état d'un objet.

1. Manipuler les caractéristiques de l'objet
2. Consulter l'état de l'objet

Cas d'utilisation : Modifier un objet.

1. Manipuler les caractéristiques de l'objet
2. Modifier un objet

Cas d'utilisation : Identification du client

1. S'identifier

Cas d'utilisation : Modifier/Créer un protocole

1. Consulter les protocoles
2. Manipuler les caractéristiques du protocole

Cas d'utilisation : Créer une planification

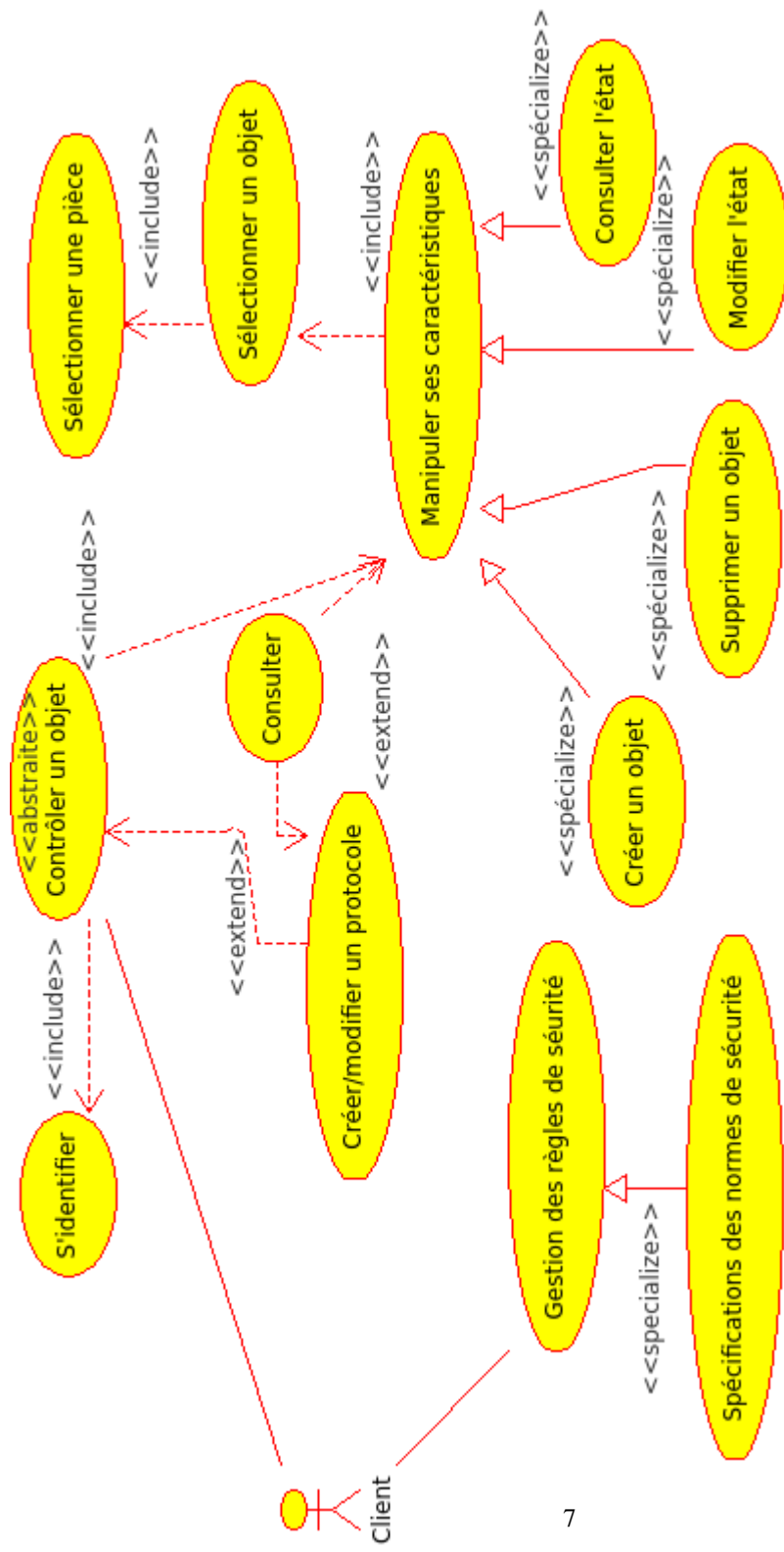
1. Manipuler les caractéristiques de l'objet

Cas d'utilisation : Créer un protocole

1. Manipuler les caractéristiques d'un objet ou plusieurs.

Cas d'utilisation : Modifier un protocole.

1. Consulter et accéder aux protocoles.
2. Manipuler les caractéristiques d'un objet ou plusieurs.



3.2 Diagramme de séquence

Le diagramme de séquence donne le déroulement des actions, permettant de visualiser le fonctionnement basique du programme. L'acteur principal, ici le client, est représenté à gauche du diagramme, et les acteurs secondaires, serveur et objet, à droite du diagramme. Le but est de décrire comment se déroulent les actions entre les acteurs ou les objets.

La dimension verticale du diagramme représente le temps, et permet de visualiser chronologiquement les actions. Les périodes d'activité des objets sont symbolisées par des rectangles, et ces objets dialoguent par le biais de messages.

Nous avons établi un diagramme de séquence qui s'est révélé solide, puisque lorsque le diagramme de classe a évolué, celui-là n'a pas changé.

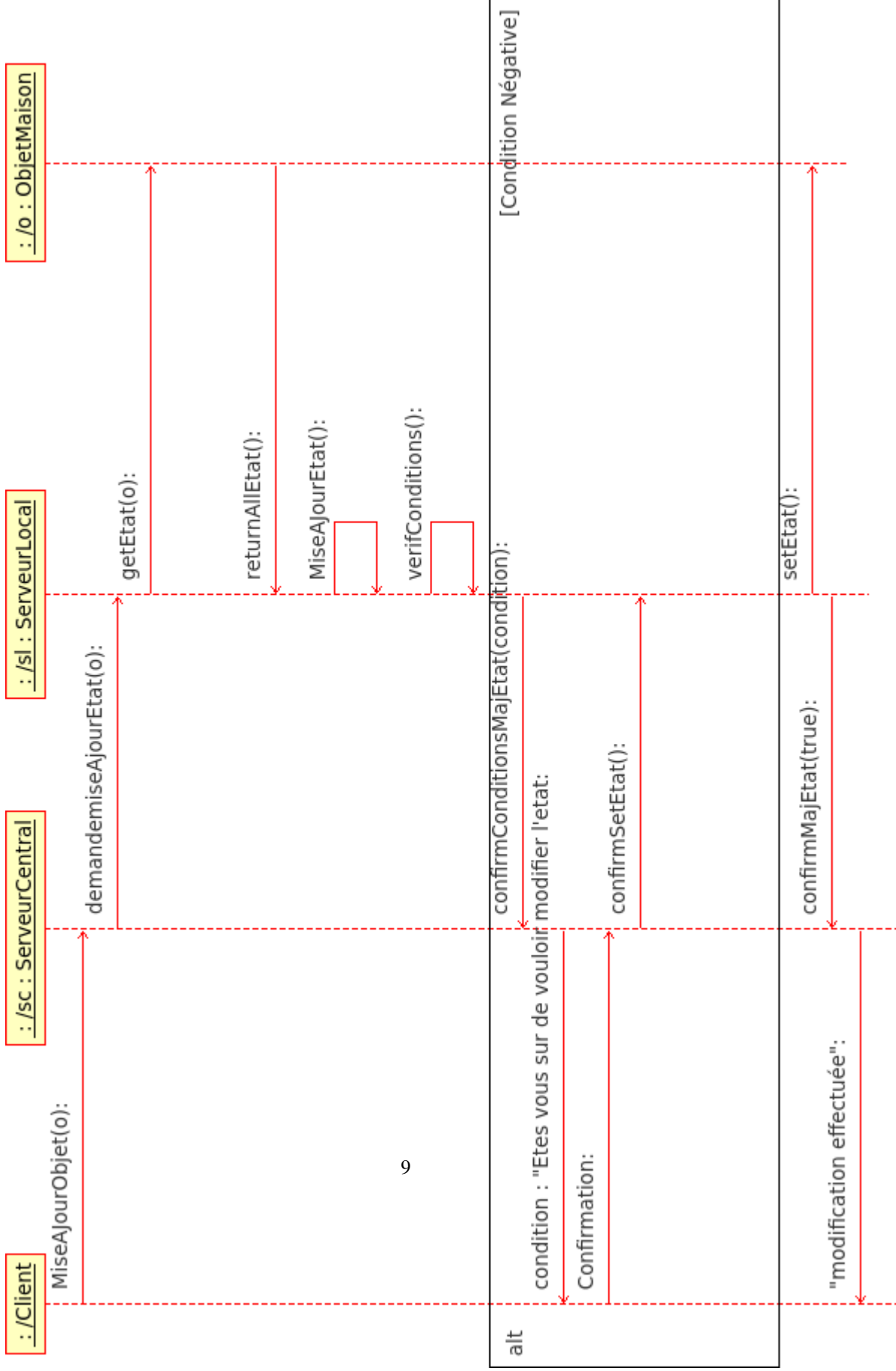
Le diagramme de séquence montre graphiquement ce qui suit :

1. Le client envoie une requête au serveur pour modifier l'état d'un objet.
2. Le serveur local reçoit la requête. Dans un premier temps ce dernier va se tenir au courant de la situation actuelle des objets de la maison pour pouvoir au mieux répondre à la demande.
3. Des vérifications de sécurité et d'état sont faites.
4. Si elles sont correctes, l'ordre est appliqué.
5. L'information est relayée au serveur, puis au client.
6. Si elles sont négatives, une alerte est envoyée au client.

Par exemple, Mr. X veut allumer la lumière du salon. A partir de l'interface web, il donne l'ordre. Le serveur reçoit la requête de demande de mise à jour de l'état de la lumière. Il demande l'état de l'objet à la lumière, qui lui est retourné. Les vérifications de sécurité sont effectuées par le serveur.

Si les vérifications confirment que la lumière était éteinte par exemple, le serveur exécute la fonction `setEtat` sur l'objet pour changer son état. Le serveur reçoit une confirmation de changement d'état de l'objet, puis renvoie au client, par l'interface web, un mot indiquant que la modification a été exécuté. Une mise à jour de l'état est effectuée.

Autrement, le serveur envoie une alerte.



3.3 Diagramme de classe

Le diagramme de classe (cf page 9) représente les relations entre les différents objets, classes et interfaces d'un système. Elle permet de modéliser un programme et ainsi de découper une tâche complexe en plusieurs tâches simples.

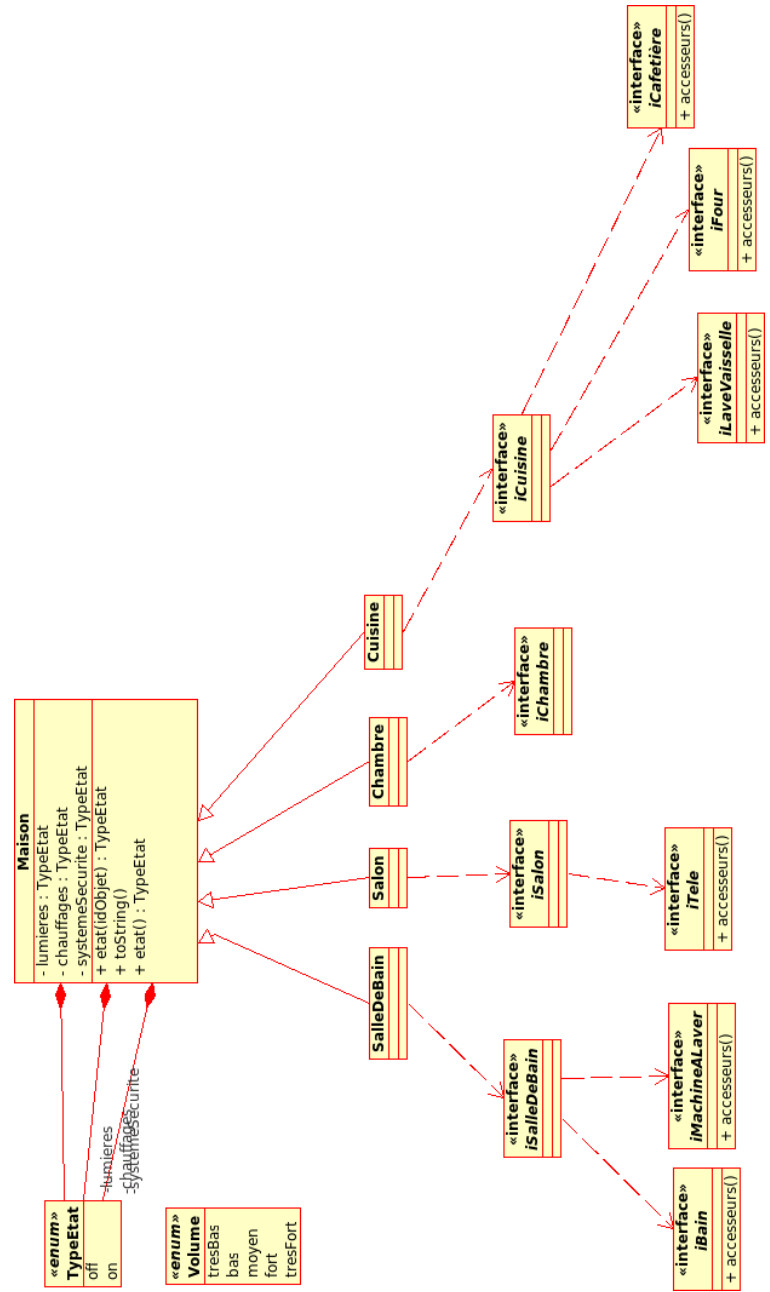
Dans un premier temps, nous avons imaginé le projet ainsi : représenter une maison grâce à une super-classe nommée *maison* qui engloberait toutes les caractéristiques communes que possèdent les différentes pièces (les classes filles) d'une maison, par exemple la lumière, pour ainsi avoir un contrôle d'ensemble. Les classes filles seraient interfacées à deux niveaux : l'interface-pièce et l'interface-technologie. Mais ce diagramme posait des problèmes pour ajouter de nouvelles technologies sans modifier le code.

Ensuite nous nous sommes orientés vers un autre diagramme. Nous avons gardé la superclasse *Maison* qui a pour attribut un identifiant et une liste d'éléments de type *Pièce* de la maison. A chaque ajout d'une nouvelle pièce, celle-ci est ajoutée à la liste. La classe *Pièce* a pour attribut les technologies "générales" comme la température et la lumière. Elle possède également une liste d'objet de type *Object*. Cette liste contient toutes les technologies contenues dans la pièce.

La classe *Pièce* est associée à une classe abstraite *textitObject* qui contiendra les technologies avec pour attribut un identifiant et l'état de la technologie. Ce diagramme fût abandonné car il nous limitait au niveau des éventuelles actions à réaliser.

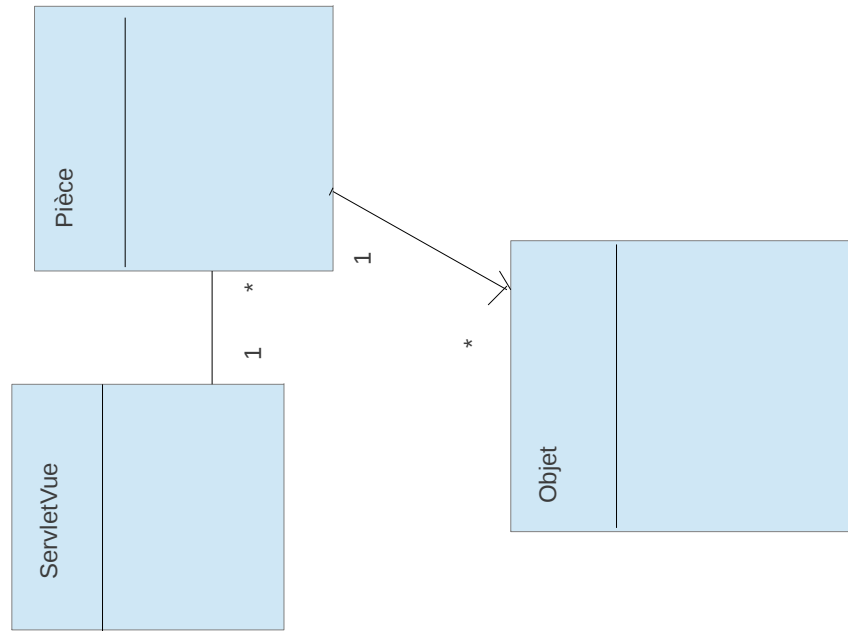
Enfin nous avons réalisé le dernier diagramme, cette fois-ci opérationnel. Ce diagramme a été trouvé tardivement, nous avons déjà commencé à coder. Mais c'est en codant que nous nous sommes rendus compte des complications que le diagramme précédent poserait.

Diagramme des Classes



Le nouveau diagramme est composé d'une part d'une classe *Pièce* contenant comme attribut un vecteur d'objets héritant des classes abstraites *Objet* tel que *ObjetBain*, *ObjetVolet* ou *ObjetTV*. Plus simplement, *Pièce* contient une liste des technologies présentes dans la pièce. La classe *Objet* est un intermédiaire pour factoriser le code en sous-classes.

D'autre part les servlets contiennent un attribut de type *ConnexionBDD* pour communiquer avec la base de données. La classe *ConnexionBDD* contient deux attributs de type *Connection* et *Statement*. Ces méthodes permettent de se connecter à la base de données.



4 Réalisation

L'architecture mise en place est composée principalement de deux parties : la partie Client, la vue et la partie Serveur, le contrôleur et le modèle, basée sur le concept du MVC.

Cette architecture datant de l'année 1979 a pour avantage de faciliter le dialogue entre les concepteurs d'une application car elle divise le travail. L'architecture est claire, ce qui en facilite la compréhension.

Au niveau de l'organisation du travail en groupe nous avons choisi d'utiliser l'*evernote* pour communiquer et se transmettre les fichiers. Bien qu'à la finalité du projet nous nous rendons compte que la mise d'un gestionnaire de projet aurait facilité grandement le développement du projet. Concernant le rapport, le langage informatique **LaTeX** a été utilisé.

Le MVC est un modèle répondant aux besoins des applications interactives que nous voulons mettre en place. Il regroupe les fonctions nécessaires en trois catégories : un modèle, une vue et un contrôleur. La vue communique avec le contrôleur. Lorsque le client modifie un paramètre, le contrôleur reçoit la modification par requête, il effectue les vérifications de sécurité en se servant du modèle, et notifie à la vue que la requête est traitée. La vue se met alors à jour via le contrôleur.

Les possibilités offertes au client sont celles qui existent dans le modèle bien que l'architecture mise en place permette l'implémentation d'un module de création de nouvelles technologies entièrement dynamisé. En ajoutant au modèle un panneau d'attributs (luminosité, canal, températures diverses,...) large, le client aurait la possibilité de créer une nouvelle technologie, en combinant des attributs. Seul ce panneau d'attributs est le facteur limitant pour la création dynamique de technologies.

4.1 Modèle

Le modèle regroupe l'ensemble des structures de données et les données elles-mêmes. Les "JavaBeans" sont les structures de données utilisées par le contrôleur lors d'une instanciation de classe. La base de données sert à stocker les données d'états. Dès qu'un objet est créé, il l'est aussi dans la base de données, avec tous ses paramètres.

Nous avons fait de nombreuses recherches pour trouver un patron de conception adapté au projet et pour comprendre comment le programmer, le MVC étant totalement inconnu pour nous tous.

Bien que l'apprentissage de la programmation objet et la conception objet soient au programme, il s'est fait parallèlement à la réalisation du projet ce qui nous a obligé à avoir un train d'avance sur la matière enseignée. De plus, la mise en place du MVC ne s'est pas avérée la solution la plus adaptée au JAVA EE.

La première version de notre code Java était constituée de plusieurs paquets :

- `com.GetEtat.beans` : contenant une classe Pièce représentant la maison, une classe pour chaque pièce (chambre, salon, etc....) et une classe main pour tester.
- `interfacesTechnos` : regroupant les interfaces pour les technologies (FOUR, LUMIÈRE,...).
- `interfacesPiece` : regroupant toutes les interfaces se trouvant dans une pièce.
- `enum` : contenant les énumérations nécessaires pour le modèle et les technologies ,OFF,ouvert,...).
- `com.GetEtat.servlets`, qui contient les liens avec le contrôleur et la base de données.

Ici, le principe était d'avoir une classe mère, la Pièce, et une classe fille pour chaque type de pièce de la maison (Salon, Cuisine,...). Chaque classe fille implémentait les interfaces des objets. En effet, chaque type d'objet (Télévision, Machine à Laver, Cafetière) était représenté par une interface qui contenait la signature des fonctions attribuées à chaque type d'objet.

Cependant le corps des fonctions et les attributs de chaque type d'objet étaient dans la classe de chaque type

de pièce auxquels ils étaient associés.

Classe Pièce

```
public class Piece{
    /*ici tout les attributs communs a toutes les pieces
    et 1 implementation des fonctions necessaires */
}

public class MaPiece extends Piece implements iMaPiece{
    /*ici les attributs de toutes les technologies se trouvant dans la pi ce MaPiece
    et les implementations necessaires, y compris celles des interfaces regroupes par
    iMaPiece */
}
```

Pour chaque interface de chaque pièce on a une implémentation de ce style.

Interface iMapièce

```
public interface iMaPiece extends iTechno1 , iTechno2 , iTechno3{
}
```

Puis chaque interface iTechno déclarait les fonctions qui nécessitaient leur propre fonctionnement. Nous avons rencontré quelques problèmes avec cette version :

- L'apprentissage : après avoir appris le code Java en cours et avec le projet, nous devions comprendre le fonctionnement et le codage des interfaces. Cela a représenté une semaine d'apprentissage et de codage.
- La pérennité : avec cette structure, nous devions écrire le code des types d'objets dans les classes des types de pièces. De ce fait, lors de l'ajout d'un nouveau type d'objet, nous devions rajouter des attributs et des fonctions dans les classes des types pièces. Nous ne respectons donc pas le principe ouvert/fermé (capacité d'être étendue, mais sans modifier le code source).

Une deuxième version a donc été étudiée, celle-ci définit de nouveaux paquetages :

- **com.GetEtat.beans**, qui contient les classes *Pièce*, *ElementsOfPiece*, *Technos* ainsi que toutes les technologies sous forme de classe.
- **enum**, qui contient tout les types de technologies possibles :
 - TYPEETAT(ON/OFF) : permet de définir si un objet est *allumé* ou *éteint*.
 - TYPEETAPORTE(OUVERT/FERME) : permet de définir si un(e) *porte/volet/fenêtre* est *ouvert(e)* ou *fermé(e)*.
 - (télévison, bain,...) : liste des type d'objets de la maison.
- **com.GetEtat.servlets**, qui contient les liens avec le contrôleur et la base de donnée.

Ici, un système de classe abstraite et d'héritage est étudiée : une classe mère *Pièce* contenant un vecteur d'éléments se trouvant dans la classe *ElementsOfPiece*, chaque élément de ce vecteur représente un objet à contrôler. La classe *ElementsOfPiece* étant elle-même abstraite. La classe mère *Pièce* est définie telle que :

Classe Pièce

```
public class Piece {
    //attributs communs toutes les pieces(id,nom,temperature Min/Max,...)
    private Vector<ElementsOfPiece> allElementsOfPiece;
}
```

Pour le contenu, voir annexe.

Les classes “Technos” et “Ouvertures” sont des classes filles de ElementsOfPiece et sont également abstraites, elle contiennent la partie commune à toutes les technologies, selon le type de cette dernière (Ouverture pour une techno ouvert/fermé et Techno pour les technologies classiques). Chaque technologie est ensuite implémenté dans une classe qui porte son nom (ex : “TechnoFour”), qui a pour mère la classe Technos et qui définit les fonctions utiles a son bon fonctionnement.

Classe TechnoFour

```
public class TechnoFour extends Technos{
    /* attributs et methodes liees au four,
       implementation des methodes declarees dans les classes mere */
}
```

Pour le contenu, voir annexe.

Ainsi la pérennité est conservée, lors de l’ajout d’un nouvel objet, il suffit juste de créer une nouvelle classe (exemple : TechnoSecheLinge) qui hérite de la classe Technos. Le code source des autres classes (abstraites ou non) restant intact. Le seul problème étant que nous avons une liste d’énumérations (voir annexe 1) pour référencer la liste des types d’objets que nous avons ; nous devons donc rajouter une énumération lors de l’ajout d’un nouveau type.

La base de données nous sert pour enregistrer les données des utilisateurs et maisons.

Dans une future version avec séparation du serveur local et central, il devra y avoir une BDD local avec les informations de la maison, et les informations concernant les utilisateurs dans la base de données centrale. Il y a une table par élément de maison. Celle-ci n’est pas optimisée. Il aurait fallut gérer ces éléments dans la même optique que les pièces.

Les contextes sur les pièces sont destinés à enregistrer les états des pièces (température courante, luminosité...). Elle est séparé en 2 tables. Une qui liste les différents états possibles avec son type (entier, flottant, string...), son unité si existante, des valeurs minimum et maximum si existantes. Une autre table étant une table de jointure entre la pièce, les contextes types, ainsi que la valeur associé au contexte.

Dans la table d’utilisateur, le mot de passe est bien évidemment crypté afin de protéger l’utilisateur.

La dernière version de notre code, mieux pensée et codée rapidement (grâce aux précédentes versions), est au final une solution améliorable mais plus stable, la pérennité du projet est assurée et elle répond mieux aux critères de la programmation objet et du JAVA EE. La structure entière du projet a été repensée en terme de code : Division des servlets, amélioration du modèle (BDD permettant d’accéder aux données de manière simple et logique) et restructuration totale de la vue.

- BDD : Suppression des tables des technos, nouvelle organisations des tables de données.
- com.GetEtat.beans : Purification des beans, mise en place d’une classe *Pièce* contenant des instances de la classe *Objet*.

Lors du changement d’organisation du programme, nous avons choisi de mettre en place pour la BDD l’amélioration prévue dans la section “Prévision”, pour avoir un code plus propre. La nouvelle BDD n’a plus une table pour chaque type de techno. Nous avons désormais six tables : itemize>

contextetype, avec l’identifiant du critère, le nom du critère, (luminosité,surface,bruit,...), une unité,une valeur min et une valeur max. Cette table est initialisée comme suit :

maison, qui crée une maison, si elle n’existe pas déjà, lui attribut un identifiant et prend un nom pour l’habitat.

objet, qui crée un objet avec un identifiant, l'identifiant de la pièce dans laquelle il se trouve, sa technologie et son nom.

Dans cette nouvelle BDD, les technologies correspondent à des entiers : ceux sont les idTypeObjet.

pièce, qui contient l'identifiant de la pièce attribué par la table, l'identifiant de la maison dans laquelle se trouve la pièce, l'entier qui détermine le "type" de la pièce, le nom de la pièce et les températures minimum et maximum de cette pièce.

piececontexte, qui correspond à la table de jointure entre la table pièce et la table contextetype en lui attribuant une valeur.

listAttributs, qui sert à regrouper tous les attributs possibles d'un objet. A présent, nous avons une table pour toutes les technologies. Tous les critères sont initialisés à NULL. Lorsqu'on ajoute une technologie, seuls les attributs qui la concernent sont modifiés.

user, la table qui enregistre les utilisateurs avec : un id pour chaque utilisateur, un login, un mot de passe et l'id de leur maison.

typeobjet, qui liste les types de technologie(télé,chauffage,...). itemize>

4.2 Vue

La vue est composée de plusieurs langages qu'il a fallu mettre en commun afin de pouvoir la rendre dynamique : toutes les pages sont générées par l'application web.

Pour répondre au sujet en traitant le modèle MVC avec la technologie Java EE, il a fallu se baser sur les librairies basées sur celle-ci.

Nous avons donc :

- Le HTML
- LE CSS
- Le JSP
- L'EL ¹⁴
- Le JSTL ¹⁵

Le duo classique HTML & CSS nous a permis de mettre un noyau en place rapidement pour pouvoir tester, implémenter, comprendre le fonctionnement des requêtes HTTP au servlet. C'est une fois cette étape d'appréhension du langage que nous avons pu nous axer sur la vue dite « dynamique » (i.e adaptée aux réponses du serveur).

Voici un bref historique de l'implémentation de cette vue.

4.2.1 1^{er} test

Il a rapidement fallu mettre en place une vue "statique" pour pouvoir tester les retours **servlets**. Cette "vue" se composait d'une simple page statique (html pur) avec un texte changeant selon la réponse reçue. Le design du site web à été implémenté grâce au **framework CSS Bootstrap de Twitter** afin d'avoir une gestion graphique plus facile et plus épurée. Son utilisation a permis de rendre plus ergonomique l'affichage client. Ce **framework** nous a pris du temps pour assimiler correctement sa puissance à nos besoins. Le site en lui-même a changé plusieurs fois de design.

Il y a tout d'abord le JSP (Java Server page), une technologie d'Oracle qui permet de créer dynamiquement du code HTML. Cette méthode permet au code Java d'être ajouté dans un contenu statique. Le JSP est

14. Expression Language

15. Java Standard Tag Library est une librairie de tags

compilé par un compilateur JSP pour devenir des servlets Java qui, à son tour, va pouvoir être compilée par le compilateur Java.

Étant donné que le modèle MVC recommande grandement la dissociation du contrôleur, du modèle et de la vue, nous avons choisi de séparer le plus possible le code Java de la vue. D'où l'utilisation du langage de script EL (Expression Language) prévu à cet effet, combiné au JSTL.

Le JSTL est un composant de la plate-forme JEE ¹⁶ qui étend la spécification JSP en ajoutant une bibliothèque de balises pour les tâches courantes, telles que les structures de contrôles (boucle et condition). L'un des problèmes étant que le JSTL n'est pas géré par Tomcat, nous avons donc dû l'importer directement dans notre projet. Il permet de rendre l'affichage plus dynamique mais ne peut fonctionner sans le EL.

Ce qui fut le plus difficile avec cette technologie a été l'apprentissage et la compréhension de celle-ci. En effet n'ayant jamais étudié le JSTL durant notre cursus scolaire, nous avons dû consacrer beaucoup de temps afin de le maîtriser, et de le synchroniser avec les autres langages (EL et HTML), mais une fois cela fait, nous avons compris l'importance et la nécessité de cette technologie dans notre projet.

Tomcat7 a été choisi comme serveur car il :

- comporte un serveur HTTP pour notre serveur web
 - implémente les spécifications des servlets et des JSP
 - est paramétrable par des fichiers XML
- permettant ainsi la gestion des communications entre le client et le serveur.

Il y a ensuite l'EL (Expression Language) qui permet de récupérer des objets Java et de les évaluer mais toujours sans effectuer de contrôles. En effet, nous pouvions évaluer des objets envoyés par le contrôleur et rendre plus dynamique l'affichage et la manipulation de la vue.

Le développement de la vue nous a posé de nombreux problèmes et notamment l'implémentation au sein du projet. A savoir que celle-ci est dépendante du contrôleur, de ce fait elle ne pouvait être réalisée tant que le contrôleur n'était pas opérationnel. En effet, la difficulté dans cette partie s'est posée sur l'« hyper-developing », le groupe avance à différentes vitesses ce qui ne permettait pas une réelle synchronisation de toutes les parties pour pouvoir au fur et à mesure améliorer nos travaux.

La vue représente l'aboutissement, la touche finale du projet : Pouvoir rendre l'application accessible à l'utilisateur de manière ergonomique. Il fallait donc pouvoir l'imbriquer avec le contrôleur sans problème, et que celui-ci puisse recevoir les bons paramètres afin que cette dernière envoie les données correspondantes. Tout au long de la mise en place de cette vue, nous nous sommes rendus compte qu'il manquait de nombreuses fonctions, méthodes ou variables nécessaires dans le contrôleur et dans le modèle. C'est pourquoi cette étape du projet a été une « super phase » de conception, débogage et programmation ponctuelle.

A l'issue de cette phase, à moins de 5 jours de la soutenance, nous avons décidé la refonte totale de la structure du projet. Ce qui nous a permis de décomposer chaque modèle de la vue en modules JSP, eux-mêmes décomposés en sous-modules pour au total arriver à 4 niveaux d'encapsulation de JSP (pour les plus grosses pages). A la manière des servlets totalement décomposés, les pages générées sont une succession d'importation de modules JSP dynamiques. La barre de navigation est elle aussi dynamique puisque l'utilisateur peut désormais cliquer sur "Maison" et consulter la liste des pièces qu'il a créé auparavant, et cliquer sur cette pièce pour modifier les états des objets à l'intérieur. La difficulté a été de décomposer l'attribut reçu ListePiece en sous objets pour les transmettre à chaque module les traitant distinctement car la technologie JSTL ne permet la transmission d'objets entre JSP-JSP. Il nous a fallu créer des variables de requêtes dans les itérations "forEach" pour les rendre accessibles lors de l'importation de modules et les écraser et les

16. Java Enterprise Edition est une plateforme développée par Sun pour faciliter le développement d'application

remplacer à chaque itération de la boucle. Pour envoyer dynamiquement ces objets aux bonnes JSP, nous avons créé des pages JSP nommées aux mêmes noms que les attributs récupérés en BDD. Par exemple, pour traiter la luminosité dans le module correspondant, la boucle `forEach` va itérer sur l'objet `Lumière` jusqu'à arriver sur cet attribut et l'envoyer à la page de traitement `./getluminosite.jsp`.

Extrait de code de la vue

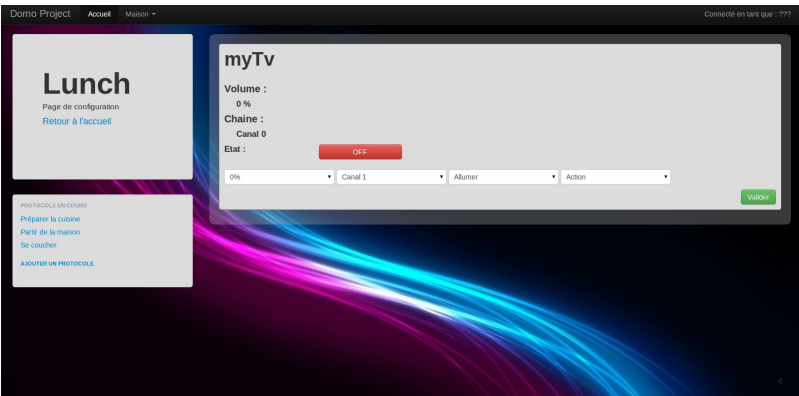
```
<form method="get" action="getEtat">

    <select name="idPiece">
        <c:forEach items="${listOfPiece}" var="piece">
            <option value="${piece.key}">
                <c:out value="${piece.value.getNomPiece()}" />
            </option>
        </c:forEach>
    </select>
    <select name="idtypeTechno">
        <c:forEach begin="1" end="6" var="techno">
            <option value="${techno}">
                <c:out value="${enumTechno[techno]}" />
            </option>
        </c:forEach>
    </select>
    <input type="text" name="nomTechno" id="nomTechno" required />
    <input class="btn" type="submit" value="Créer nouvelle pièce">
    <input type="hidden" name="createTechno" value="true">

</form>
```

Voici un exemple d'utilisation du JSTL au sein d'un code HTML permettant de faire une liste de sélections dynamiques pour paramétrer un objet. Ce code est une boucle `forEach` de la librairie Core de JSTL qui itère sur l'objet examiné. Le code fonctionne par objets implicites : le servlet envoie à cette page de configuration (d'où est tiré ce formulaire) une table de hachage contenant la liste des pièces qui contient la liste des technologies, pour pouvoir les traiter de manière dynamique dans la vue.

L'interface de login ci-dessous représente la page d'accueil qui permet de s'identifier pour accéder à son compte. Les informations saisies sont transmises par requêtes au servlet. Le servlet vérifie alors leurs existances dans la base de données. Si l'identifiant est connu le servlet instancie la maison du client et l'envoie à la vue qui affiche la page contenant les biens immobiliers du client ainsi que les pièces et leurs technologies. Sinon le servlet retourne à la vue l'interface de login.



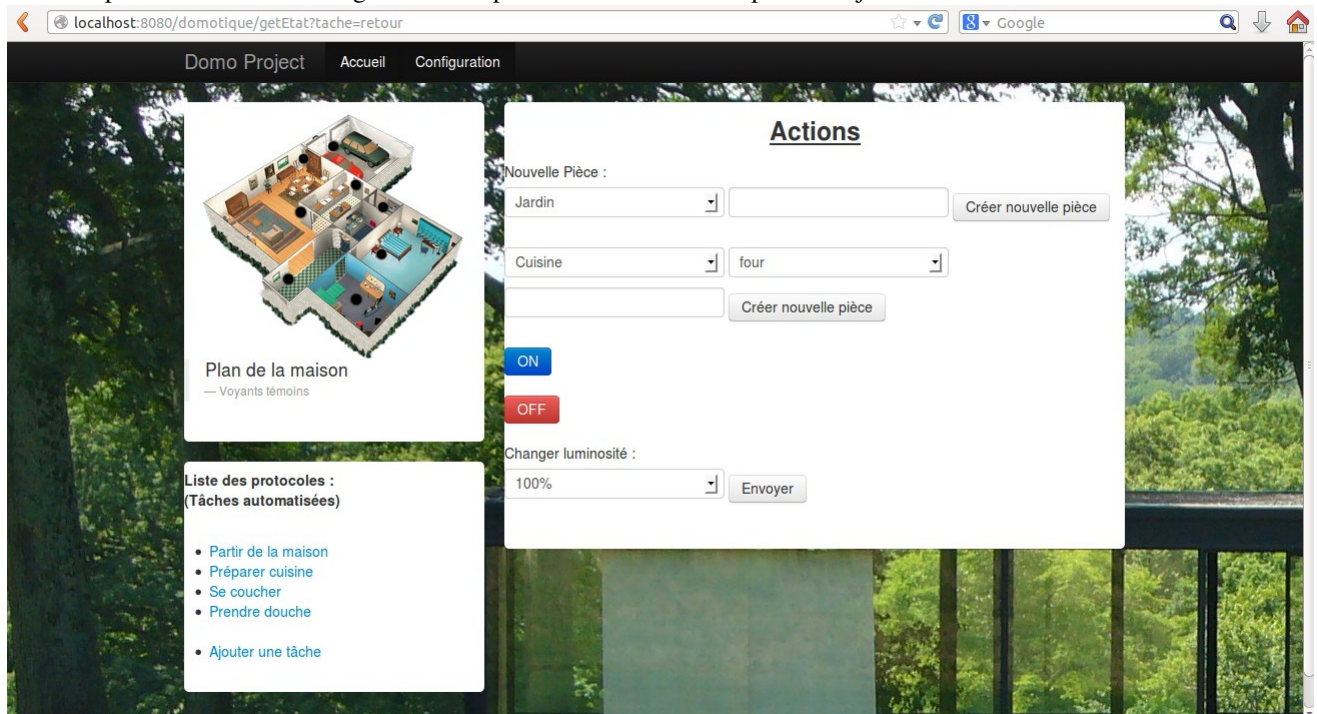
L'image ci-dessous est l'image s'affichant une fois que vos identifiants sont validés par le serveur. Ici vous pouvez voir 4 parties : La barre de navigation dynamique qui a un bouton accueil, une liste déroulante "Maison" qui affiche la liste des pièces de la maison. Une partie où vous apercevez la configuration de la maison, une liste de protocoles à effectuer et une dernière partie pour créer, modifier, supprimer des tâches, des protocoles.

La seconde partie, contenant une liste de protocoles, ceux-ci sont créés par l'utilisateur lors d'un choix de paramétrage des tâches. Lors de l'activation d'un protocole, la requête est envoyée au servlet qui va gérer le traitement des actions à effectuer, d'une part la sauvegarde du protocole dans un fichier JSON ; d'autre part la mise à jour de la BDD en fonction des actions à effectuer.

En effet, la troisième partie de l'image est une partie où l'utilisateur crée une ou plusieurs tâches, il pourra alors choisir d'effectuer la tâche immédiatement ou bien de l'ajouter à la liste de tâches (vu dans la 2ème partie), puis de créer un protocole avec cette liste, s'il le souhaite.

Lors de la création d'une pièce, une requête est envoyée au servlet afin qu'il instancie un Beans avec les paramètres envoyés par la requête ; de même il ajoute le champ correspondant dans la base de données. Lors de l'ajout d'une technologie le même procédé est suivi. En outre, les créations de pièce et de technologie sont dynamiques :

- Une pièce/technologie ne peut être créée que si elle existe déjà dans la base de données.
- On ne peut créer une technologie dans une pièce seulement si cette pièce a déjà été créée.



4.3 Contrôleur

Le contrôleur est le chef d'orchestre du programme. Il a pour rôle de contrôler les actions provenant du client qui sont sous formes de requêtes HTTP. Si l'action est de modifier des données, il demande la modification des données au modèle et notifie la vue que les données ont changé pour qu'elle se mette à jour sinon il analyse la requête du client et se contente d'appeler le modèle adéquat et de renvoyer la vue correspondant à la demande.

4.3.1 Première version

Lors de la conception du prototype du projet, nous avons privilégié la synchronisation entre le modèle et le contrôleur puisque c'est lui qui gère les actions. Il fallait donc bien comprendre les Beans pour pouvoir instancier les objets et en même temps être capable de les créer dans la BDD. Cette dernière requérant la connaissance du langage SQL pour assurer la communication avec la BDD. Nous avons donc été amené à maîtriser plusieurs langages : le JavaBeans et le SQL. Mais aussi à apprendre à se servir de technologies comme le JavaServlets et à comprendre le fonctionnement des requêtes HTTP. Nous nous sommes servi du logiciel Eclipse pour implémenter les servlets et les JavaBeans.

Par la suite nous il a fallu résoudre le problème qu'allait poser un crash de serveur. Lorsqu'on crée un objet, il est créé dans la BDD. Dans ce cas là, la BDD n'est pas détruite, mais les objets n'existent plus.

Or, pour la vue, qui communique avec les Beans, ces objets doivent exister. La fonction `initialisPiec` recrée la table de hachage en allant chercher les informations dans la base de données. Cette fonction d'initialisation est appelée à chaque fois qu'on appelle la servlet. A chaque fois qu'on appelle la servlet, on vide la table de hachage et on récupère les informations dans la base de données pour être cohérent avec celle-ci. Cette fonction est synchrone : en cas de modification, elle met à jour la table de hachage des pièces.

Méthode de la classe `getEtat` - Fonction d'initialisation des pièces

```
public boolean initialisPiec(int idMaison){
    try {
        ResultSet rs = st.executeQuery("SELECT * FROM piece where
            IDmaison=" + idMaison);
        while(rs.next()){
            int id = rs.getInt("ID");
            String nom = rs.getString("nom");
            int tempMin = rs.getInt("temperatureMin");
            int tempMax = rs.getInt("temperatureMax");
            Piece p = new Piece(id,nom,tempMin,tempMax);
            listPiec.put(id,p);
        }
    } catch (SQLException e) {
        System.out.println("ERR BDD");
    }
}
```

```

        e.printStackTrace();
        return false;
    }
    return true;
}

```

La fonction suivante récupère à partir de la base de données toutes les technologies et recrée ces technologies dans la table de hachage.

```

private Vector<struct> initialisTechno(int idMaison){
    Vector<struct> s= new Vector<struct>();
    try {
        ResultSet rs = st.executeQuery("SELECT * FROM objet, piece
        WHERE piece.ID=idPiece AND IDmaison=" + idMaison);
        while(rs.next()){
            String idObjet = rs.getString("idObjet");
            String idPiece = rs.getString("idPiece");
            String idTypeObjet = rs.getString("idTypeObjet");
            String nom = rs.getString("nomObjet");
            struct st=new struct(idObjet,idPiece,idTypeObjet,nom);
            s.addElement(st);
        }
    } catch (SQLException e) {
        System.out.println("ERR BDD 2");
        e.printStackTrace();
    }
    return s;
}

public void initi(Vector<struct> s){
    for(struct str : s){
        int idObjet = Integer.parseInt(str.idObjet);
        int idPiece = Integer.parseInt(str.idPiece);
        int idTypeObjet = Integer.parseInt(str.idTypeObjet);
        String nom = str.nom;
        switch(idTypeObjet){
            case 1 : recupAttTV(idObjet, idPiece, nom);break;
            case 2 : recupAttLumiere(idObjet, idPiece, nom);break;
            case 3 : recupAttChauffage(idObjet, idPiece, nom);break;
            case 4 : recupAttVolet(idObjet, idPiece, nom);break;
            case 5 : recupAttBain(idObjet, idPiece, nom);break;
            case 6 : recupAttFour(idObjet, idPiece, nom);break;
        }
    }
}

```

Créer des protocoles n'était pas évident à réaliser. Nous avons choisi de stocker les différentes actions à faire et les caractéristiques de ces objets dans un fichier JSON¹⁷ pour plus de facilité. Les protocoles sont créés en convertissant le fichier Json en table de hachage pour pouvoir au mieux manipuler ces données en vue d'une modification du protocole. Nous avons également fait une méthode pour enregistrer les données d'une table de hachage dans un fichier Json afin d'archiver les protocoles. La création d'un protocole se fait par l'ajout successif d'actions, de la part du client, puis ce dernier va finaliser l'ajout des actions (via un bouton), ce qui va enclencher une sauvegarde de ce protocole dans le fichier JSON.

17. JavaObject Notation est un format de données textuelles qui permet de représenter de l'information structurée

Méthode pour encoder une hashtable en fichier Json

```
public void encodeJSON(Hashtable ht){
    if(jsonFile!=null)
        encodeJSON(ht, jsonFile);
    else
        System.out.println("Le fichier json n'a pas t initialis .");
}
```

Méthode pour decoder une hashtable en fichier Json

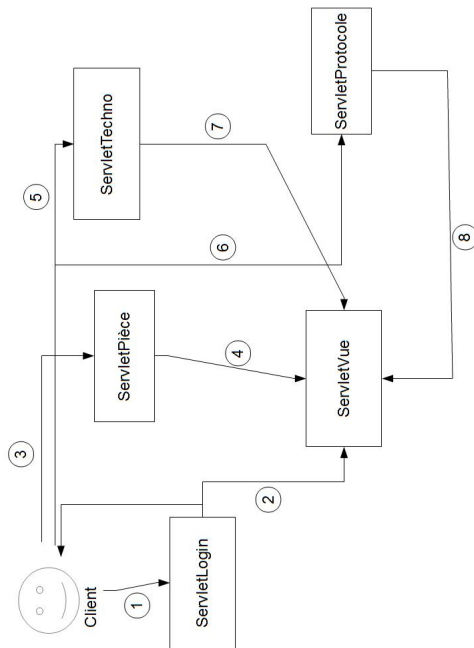
```
public Hashtable decodeJSON(){
    if(jsonFile!=null)
        return decodeJSON(jsonFile);
    else
        System.out.println("Le fichier json n'a pas t initialis .");
    return null;
}
```

Une des difficultés que nous avons eu en gérant les protocoles est le déclenchement d'un tel protocole à une certaine heure ou à une certaine date, nous n'avons pas pu gérer le déclenchement événementiel des protocoles.

4.3.2 La version officielle

La dernière version du projet, et des servlets a grandement été changée :

- com.GetEtat.servlets : Il a été choisi de mettre en place un servlet par action :
 - 1 servlet ConnexionBDD, gérant les communications BDD serveurs
 - 1 servlet ServletPiece, gérant toutes les actions sur les pièces
 - 1 servlet ServletTechno, gérant toutes les actions sur les technos
 - 1 servlet ServletProtocole, gérant les protocoles en partenariat avec le ServletTechno via un servlet de redirection.
 - 1 servlet ServletVue, récupérant toutes les actions fournies par les autres servlets permettant le mapping de la vue et ses transmission d'objets implicites pour les JSP.
 - 1 servet ServletLogin, gérant les connexions et la mise en place des sessions d'utilisateurs.



newpage L'absence de cahier des charges a d'abord posé quelques problèmes au niveau du travail à effectuer. En effet, nous avons l'impression d'avoir aucun point de départ, et avons dû nous imposer un cahier des charges afin de mener le projet à bien..

Globalement l'organisation du travail a été assez difficile, la taille du groupe de travail (9 personnes) a posé certains problèmes au chef de projet pour répartir les tâches. La gestion du temps de travail a aussi présenté certaines difficultés du fait de la vitesse de progression de chacun. L'apprentissage des technologies fût délicat pour certaines personnes. Enfin pour le chef de projet la mise en relation des différentes parties fût particulièrement compliquée et a engendré une grosse perte de temps.

5 Prévisions

Maintenant qu'un noyau solide et fonctionnel a été implémenté, il est possible d'améliorer de nombreuses fonctionnalités que nous n'avons pas eu le temps d'implémenter.

On peut citer par exemple la gestion d'un serveur central redirigeant vers les serveurs des particuliers pour une application multi-user, une amélioration du système de protocoles avec la gestion des dates ou encore la gestion de création dynamique de technologies grâce à la nouvelle structure des modèles. On peut citer encore la gestion événementielle des warnings, ou l'implémentation en ligne pour une utilisation mobile.

6 Conclusion

Durant ce projet, nous avons appris de nouvelles technologies pour la vue (JSP, JSTL, ...) et pour le contrôleur (servlet, SQL, ...). Écrire de la documentation en JAVA a également dû être maîtrisé. De plus, l'apprentissage du Latex nous sera utile tout au long de nos études supérieures. La gestion du projet, en terme de temps, est une expérience intéressante. La pression due à la date limite a été difficile à gérer.

Nous avons dû apprendre à gérer le travail de groupe, avancer avec les autres, et le travail individuel. Nous avons dû progresser en communication, pour la bonne marche du projet.

L'auto-documentation a été essentielle dans le déroulement du projet : Internet, livre... Nous avons utilisé plusieurs moyens d'apprentissage, même si Internet reste le plus utilisé. L'absence de cahier des charges a d'abord posé quelques problèmes au niveau du travail à effectuer. En effet, nous avions l'impression d'avoir aucun point de départ, et avons dû nous imposer un cahier des charges afin de mener le projet à bien..

Globalement l'organisation du travail a été assez difficile, la taille du groupe de travail (9 personnes) a posé certains problèmes au chef de projet pour répartir les tâches. La gestion du temps de travail a aussi présenté certaines difficultés du fait de la vitesse de progression de chacun. L'apprentissage des technologies fût délicat pour certaines personnes. Enfin pour le chef de projet la mise en relation des différentes parties fût particulièrement compliquée et a engendré une grosse perte de temps.

7 Manuel d'utilisation

7.1 Entrée sur la plateforme de Domo Project

Tout d'abord, munissez vous des identifiants qui vous ont été remis lors de l'installation du matériel de domotique chez vous.

Vous devez avoir un login et un mot de passe. Saisissez votre mot de passe et login. Une fois connecté vous pouvez configurer votre maison en cliquant sur le bouton de validation du formulaire.

7.2 Agir immédiatement sur la maison

Après vous être connecté...

7.2.1 Ajouter un objet de la maison

Rendez vous sur la page de configurations. Sélectionnez la pièce dans laquelle il doit être ajouté, lui donner un type (télévision, chauffage...) ainsi qu'un nom puis valider le formulaire.

7.2.2 Supprimer/Modifier un objet de la maison

Se rendre sur la page de la pièce où se trouve l'objet en question. Cliquez sur le bouton rouge prévu à cet effet.

7.3 Protocole

Allez sur la page "Gestion des protocoles".

7.3.1 Créer un protocole

Rendez vous sur les paramètres des objets et choisir un protocole. Validez le protocole final, en se rendant sur la page de gestion des protocoles.

7.3.2 Supprimer un protocole

7.4 Pièce

7.4.1 Créer une nouvelle pièce

Dans la fenêtre au centre se trouve un bouton “Créer une pièce”, cliquez dessus. Configurez la pièce.

7.4.2 Modifier une pièce

Pour modifier une pièce, choisissez votre pièce dans le menu déroulant et cliquez dessus.

7.4.3 Supprimer une pièce

Sélectionnez la pièce par le biais du menu déroulant. Cliquez ensuite sur le bouton “Supprimer”, en bas, à droite.

8 Remerciements

Nous remercions notre tuteur, Abdelhak-Djamel Seriai, sans qui le projet n'existerait pas, pour son aide durant ces trois mois, Clémentine Nebut pour son aide en JAVA et en UML. Ils nous ont apporté lors des différents suivis l'aide et les conseils concernant les missions évoquées dans ce rapport.

Merci à la cafetière qui nous a tenu compagnie lors des longues nuits de perdition dans la documentation de JAVA EE.

9 Annexe

1. Enumération

Enumeration des technologies

```
public enum TypeTechno {  
    tele,  
    bain,  
    cafetiere,  
    four,  
    laveVaisselle,  
    machineALaver,  
    piece  
}
```

2. Classe *getEtat*, la servlet.

Classe "getEtat"

```
@SuppressWarnings("serial")
/**
 * @author Mina Shakhloul
 */
public class getEtat extends HttpServlet {

    public static final String VUE1 = "/index.html";
    public static final String VUE2 = "/return.html";
    private ConnexionDB con = null;
    private Statement st;
    private ResultSet rs;
    private Connection conn;
    Hashtable<Integer,Piece> listPiece=new Hashtable<Integer,Piece>();
    private static final File myFile = new File("/home/mina/ProjectDomo/
        domotique/src/com/GetEtat/protocole.json");
    Iterator<Piece> itListPiece;

    public void init(ServletConfig config) throws ServletException{
        super.init(config);
    }

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException {
        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException {
        /*
         * recupere les parametres, avec la recuperation de l'etat souhaite
         * idTypePiece->nous sert a savoir le type de la piece pour la bdd
         * idElemPiece-> id effectif de la piece qui est dans la bdd
         * idTypeTechno -> id du type de la techno
         * idElemTechno-> id effectif de la techno
         * Etat -> pour determiner si on fait la modif de l'etat d'un
            element
         * temp -> temp voulue
         * volume, chaine,lum pareil
         * createPiece -> pour determiner si on doit creer une piece ou pas
         * nomPiece -> nom de la piece choisit par le client
         * createTech-> pareil que createPiece
         * allumeAll->pr allumer tte les pieces..
         */
        String idTypePiece = request.getParameter("idtypePiece");
        System.out.print("idtype est: " + idTypePiece + "\n");
        String idElemPiece = request.getParameter("idPiece");
        String idTypeTechno = request.getParameter("idtypeTechno");
        System.out.print("idtypeTechno est: " + idTypeTechno + "\n");
        String idElemTechno = request.getParameter("idTechno");
        String Etat = request.getParameter("getEtat");
```

```

System.out.print("Etat est: " + Etat + "\n");
String temp = request.getParameter("temperatureVoulue");
String vol = request.getParameter("VoulumeVoulu");
String chaine = request.getParameter("chaineVoulue");
String lum = request.getParameter("luminositeVoulue");
String createPiece=request.getParameter("createPiece");
System.out.print("creer est: " + createPiece + "\n");
String createTech=request.getParameter("createTechno");
String nomPiece=request.getParameter("nomPiece");
System.out.print("nomPiece est: " + nomPiece + "\n");
String nomTechno=request.getParameter("nomTechno");
System.out.print("nomTechno est: " + nomTechno + "\n");
//String allumeAll=request.getParameter("allumeAllPiece");
String deletePiece=request.getParameter("deletePiece");
String deleteAllPiece=request.getParameter("deleteAllPiece");
String protocole=request.getParameter("creerProtocole");
String tache=request.getParameter("tache");
String nomProtocole=request.getParameter("nomProtocole");

CreateSession();

//initialisation de la maison
System.out.println("Initialisation de la maison.");
if(initialisPiece(1))
    initialisTechno(1);
System.out.println("Fin de l'initialisation.");

//creation d'une piece ou ajout d'une piece
try{
    if(!createPiece.equals(null)){
        int idtypePiece= Integer.parseInt(idTypePiece);
        createPiece(1,idtypePiece,nomPiece);
        System.out.println(listPiece.get(1).getNomPiece());
        int idP =listPiece.size();
        System.out.println(idP);
        request.setAttribute( "listOfPiece", listPiece);
        this.getServletContext().getRequestDispatcher("/test
.jsp").forward(request, response);
        System.out.println(idP);
    }
}catch (NullPointerException e) {}
try{
    if(!tache.equals(null)){
        request.setAttribute( "listOfPiece", listPiece);
        this.getServletContext().getRequestDispatcher("/
retour.jsp").forward(request, response);
    }
}catch(NullPointerException e) {}

//creation d'une techno ou ajout d'une techno
try{
    if(!createTech.equals(null)){
        int idPiece= Integer.parseInt(idElemPiece);
        int idtypeTech= Integer.parseInt(idTypeTechno);
        createTechno(idPiece,idtypeTech,nomTechno);
        System.out.println("taille du vector des technos "+

```

```

        listPiece.get(1).getAllElementsOfPiece().size();
    };
    int idt =listPiece.size();
    request.setAttribute( "listOfPiece", listPiece);
    this.getServletContext().getRequestDispatcher("/test
        .jsp").forward(request, response);
    }
} catch (NullPointerException e) {}

//changement de l'etat
try{
    if(!Etat.equals(null)) {
        int idPiece= Integer.parseInt(idElemPiece);
        int idTechno= Integer.parseInt(idElemTechno);
        int idtypeTech= Integer.parseInt(idTypeTechno);
        String tech=convertToTechno(idtypeTech);
        Piece p=listPiece.get(idPiece);
        for(ElementsOfPiece e: p.getAllElementsOfPiece()){
            if(e.getIdentifiant()==idTechno){
                if(RecupEtat(tech,idTechno)==1){
                    UpdateEtat(0,tech,idTechno);
                    if(e instanceof Technos){
                        ((Technos) e).
                            setEtat(TypeEtat
                                .OFF);
                    }else{
                        if(e instanceof
                            Ouvertures){
                            ((Ouvertures
                                ) e).
                                setEtatOuverture
                                    (
                                        TypeEtatPorte
                                            .ferme);
                        }
                    }
                }else{
                    if(RecupEtat(tech,idTechno)
                        ==0){
                        UpdateEtat(1,tech,
                            idTechno);
                        if(e instanceof
                            Technos){
                            ((Technos) e
                                ).
                                setEtat(
                                    TypeEtat
                                        .ON);
                        }else{
                            if(e
                                instanceof
                                    Ouvertures
                                ){
                                    ((
                                        Ouvertures

```



```

        }
    }
}
}
}catch (NullPointerException e) {}
//chang. chaine
try{
    if(!chaine.equals(null)){
        int idPiece= Integer.parseInt(idElemPiece);
        int idTechno= Integer.parseInt(idElemTechno);
        int idtypeTech= Integer.parseInt(idTypeTechno);
        String tech=convertToTechno(idtypeTech);
        int chaineVoulue= Integer.parseInt(chaine);
        Piece p=listPiece.get(idPiece);
        for(ElementsOfPiece e: p.getAllElementsOfPiece()){
            if(e.getIdentifiant()==idTechno){
                if(((TechnoTV) e).setChaine(
                    chaineVoulue)){
                    changeChannel(tech,
                        chaineVoulue, idTechno);
                }
            }
        }
    }
}
}catch (NullPointerException e) {}
//chang. luminosite
try{
    if(!lum.equals(null)){
        int idPiece= Integer.parseInt(idElemPiece);
        int idTechno= Integer.parseInt(idElemTechno);
        int lumVoulue= Integer.parseInt(lum);
        Piece p=listPiece.get(idPiece);
        for(ElementsOfPiece e: p.getAllElementsOfPiece()){
            if(e.getIdentifiant()==idTechno){
                if(((TechnoLumiere) e).setVariateur(
                    lumVoulue)){
                    changeLumino(idTechno,
                        lumVoulue);
                }
            }
        }
    }
}
}catch (NullPointerException e) {}
//supprimer une piece
try{
    if(!deletePiece.equals(null)){
        int idPiece= Integer.parseInt(idElemPiece);
        deletePiece(idPiece);
    }
}
}catch (NullPointerException e) {}
//supprimer toutes les pieces
try{
    if(!deleteAllPiece.equals(null)){
        deleteAllPiece();
    }
}
}catch (NullPointerException e) {}
//creer un protocole
try{
    if(!protocole.equals(null)){

```

```

        creerProtocole(1, nomProtocole);
    }
} catch (NullPointerException e) {}

    Destroy();
}

/**
 * Creation d'une connexion avec la bdd
 */
public void CreateSession() {
    try {
        con = new ConnexionDB();
        Connection conn = con.connect();
        st = conn.createStatement();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

/**
 * * Destruction de la Connection et la session une fois qu'on a termine de
 * * communiquer avec la bdd
 */
public void Destroy() {
    try {
        rs.close();
        st.close();
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (NullPointerException e) {
    }
}

}

/**
 * recupere l'etat d'un element de la bdd
 * @param elem
 * @param id
 * @return
 */
public int RecupEtat(String elem, int id) {
    int etat = 0;
    try {
        rs = st.executeQuery("SELECT etat FROM " + elem + " where ID"
            + id);
        while (rs.next()) {
            etat = rs.getInt(1);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return etat;
}

```

```

}
/**
 * MAJ de la bdd (changement de l'etat)
 * @param nouvelEtat
 * @param elem
 * @param id
 */
public void UpdateEtat(int nouvelEtat, String elem, int id) {
    try {
        st.executeUpdate("Update " + elem + " set etat= " +
            nouvelEtat
                + " where ID =" + id);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

/**
 * renvoie le nom de la techno a partir de son idtype
 * @param idt
 * @return
 */
public String convertToTechno(int idt){
    switch(idt){
        case 2 : return "elemtv";
        case 3 : return "elembain";
        case 5 : return "elemfour";
        case 7 : return "elemvolet";
        case 11 : return "elemlumiere";
        case 13 : return "elemchauffage";
        default : return "";
    }
}

/**
 * sert a recuperer id d'un element de la bdd a partir de son nom
 * @param nom de l'element
 * @param nomTable nom de la table
 * @return id de l'elt
 */
public int getId(String nom,String nomTable){
    int id=0;
    try {
        rs = st.executeQuery("SELECT id FROM " + nomTable + " where
            nom = '" + nom + "'" );
        while (rs.next()) {
            id = rs.getInt(1);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return id;
}

/**
 * definir le prochain id a affecter pour un nouveau objet
 * @param table

```

```

        * @return id
        */
    public int selectNewId(String table){
        int id=0;
        try {
            rs = st.executeQuery("select max(id) from "+table);
            while (rs.next()) {
                id = rs.getInt(1) + 1;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return id;
    }

    /**
     * Creation d'une maison dans la bdd
     * @param nom
     */
    public void createMaison(String nom){
        try {
            int idM=0;
            idM=selectNewId("maison");
            st.executeUpdate("insert into maison values (" +idM+", '"+nom+
                "')");
        }
        catch (SQLException e) { e.printStackTrace(); }
    }

    /**
     * cree une piece comme objet et comme data dans la bdd
     * @param idMaison id de la maison actuelle
     * @param idP id de la piece
     */
    public void createPiece(int idMaison, int idP, String nom){
        int id=0;
        id=selectNewId("piece");
        insertPieceIntoBdd("piece", id, idMaison, idP, nom);
        Piece p = new Piece(id, nom);
        listPiece.put(id, p);
    }

    /**
     * cree une techno comme objet et comme data en bdd
     * @param idP id de la piece dans laquelle on souhaite ajoute la techno
     * @param idTechno id de la techno, qui l'identifie
     */
    public void createTechno(int idPiece, int idtypeTechno, String nomTech) {
        int idt = 0;
        Piece p=listPiece.get(idPiece);
        System.out.println("idtypeTechno "+idtypeTechno);
        switch(idtypeTechno) {
            case 2 :
                idt=selectNewId(convertToTechno(idtypeTechno));
                TechnoTV tv=new TechnoTV(idt, nomTech);
                p.addElement(tv);
                tv.setPiece(p);
        }
    }

```

```

        insertTechnoIntoBdd("elemtv",tv.getIdentifiant(),idPiece,
            nomTech);
        break;
    case 3 :
        idt=selectNewId(convertToTechno(idtypeTechno));
        TechnoBain b=new TechnoBain(idt,nomTech);
        p.addElement(b);
        b.setPiece(p);
        insertTechnoIntoBdd("elembain",b.getIdentifiant(),idPiece,
            nomTech);
        break;
    case 5 :
        idt=selectNewId(convertToTechno(idtypeTechno));
        TechnoFour f=new TechnoFour(idt,nomTech);
        p.addElement(f);
        f.setPiece(p);
        insertTechnoIntoBdd("elemfour",f.getIdentifiant(),idPiece,
            nomTech);
        break;
    case 7 :
        idt=selectNewId(convertToTechno(idtypeTechno));
        TechnoVolets v=new TechnoVolets(idt,nomTech);
        p.addElement(v);
        v.setPiece(p);
        insertTechnoIntoBdd("elemvolet",v.getIdentifiant(),idPiece,
            nomTech);
        break;
    case 11 :
        idt=selectNewId(convertToTechno(idtypeTechno));
        TechnoLumiere lu=new TechnoLumiere(idt,nomTech);
        p.addElement(lu);
        lu.setPiece(p);

        insertTechnoIntoBdd("elemlumiere", lu.getIdentifiant(),
            idPiece,nomTech);
        break;
    case 13 :
        idt=selectNewId(convertToTechno(idtypeTechno));
        TechnoChauffage ch=new TechnoChauffage(idt,nomTech);
        p.addElement(ch);
        ch.setPiece(p);
        insertTechnoIntoBdd("elemchauffage", ch.getIdentifiant(),
            idPiece,nomTech);
    }
}

/**
 * insérer les données d'une pièce dans la bdd
 * @param table
 * @param id
 * @param idMaison
 * @param idP
 * @param nom
 */
public void insertPieceIntoBdd(String table,int id,int idMaison,int idP,
    String nom){
    try{
        st.executeUpdate("insert into "+table+" (ID,IDmaison,type,

```

```

        nom)values (" + id + ","
                    + idMaison + ","+idP+"','"+nom+"'");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

/**
 * insérer les données d'une techno dans la bdd
 * @param table de la techno a modifier
 * @param idt id de la techno
 * @param idP id de la piece a laquelle elle est rattachée
 * @param nom nom de la techno
 */
public void insertTechnoIntoBdd(String table,int idt,int idP,String nom){
    try{
        st.executeUpdate("insert into "+table+" (ID,IDpiece,nom)
            values (" + idt + ","+idP+"','"+nom+"'");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

/**
 * supprime une piece donnée
 * @param idPiece id de la piece a supprimer
 */
public void deletePiece(int idPiece){
    try{
        st.executeUpdate("Delete FROM piece WHERE ID = "+idPiece);
        listPiece.remove(idPiece);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

/**
 * suppression de toutes les pieces
 */
public void deleteAllPiece(){
    try{
        st.executeUpdate("Delete * FROM piece");
        listPiece.clear();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

/**
 * @param idMaison
 * @param nomProtocole
 * @return retourne la liste des protocoles d'une maison
 */
public Vector<String> creerProtocole(int idMaison,String nomProtocole){
    JsonConverter j = new JsonConverter(myFile);
    JHashtable jht = new JHashtable(j.decodeJSON());
    jht.protocole(idMaison,nomProtocole);
    j.encodeJSON(jht.toHashtable());
}

```

```

        return jht.listeProtocoles(idMaison);
    }

    /**
     * @param idMaison
     * @param nomProtocole
     * @param nomAction
     * @return retourne la liste des actions d'un protocole donne d'une maison
     * donnee
     */
    public Vector<String> creerAction(int idMaison,String nomProtocole,String
        nomAction){
        JsonConverter j = new JsonConverter(myFile);
        JHashtable jht = new JHashtable(j.decodeJSON());
        jht.action(idMaison,nomProtocole, nomAction);
        j.encodeJSON(jht.toHashtable());
        return jht.listeActions(idMaison, nomProtocole);
    }

    /**
     * @param nomProtocole
     * @return
     */
    public boolean executeProtocole(String nomProtocole){
        JsonConverter j = new JsonConverter(myFile);
        JHashtable jht = new JHashtable(j.decodeJSON());
        Vector<String> listAction=jht.listeActions(1, nomProtocole);
        for(String action : listAction)
        {
            JHashtable monAction = jht.action(1, nomProtocole, action);
            monAction.get("TECHNO");
            monAction.get("IDTECHNO");
            monAction.get("ACTION");
            monAction.get("VALUE");
        }
        return true;
    }

    /**
     * @param idMaison de la maison
     * @return si l'initialisation a ete correctement faite
     */
    public boolean initialisPiece(int idMaison){
        try {
            ResultSet rs = st.executeQuery("SELECT * FROM piece where
                IDmaison=" + idMaison);
            while(rs.next()){
                int id = rs.getInt("ID");
                String nom = rs.getString("nom");
                int tempMin = rs.getInt("temperatureMin");
                int tempMax = rs.getInt("temperatureMax");
                Piece p = new Piece(id,nom,tempMin,tempMax);
                listPiece.put(id,p);
            }
        } catch (SQLException e) {
            System.out.println("ERR BDD");
            e.printStackTrace();
            return false;
        }
    }

```



```

        return true;
    }
    /**
     * recuperation de toutes les technos de la piece
     * @param idPiece id de la piece
     */
    private void initialisTechno(int idMaison){
        try {
            ResultSet rs = st.executeQuery("SELECT * FROM elemtv, piece
            WHERE piece.ID=IDpiece AND IDmaison=" + idMaison);
            while (rs.next()) {
                int id = rs.getInt("ID");
                int idPiece = rs.getInt("IDpiece");
                String nom = rs.getString("nom");
                int chaine = rs.getInt("chaine");
                int volume = rs.getInt("volume");
                TypeEtat etat=(rs.getInt("etat")==1 ? TypeEtat.ON :
                TypeEtat.OFF);
                Piece p = listPiece.get(idPiece);
                new TechnoTV(id,nom,p,chaine,volume,etat);
            }

            rs = st.executeQuery("SELECT * FROM elembain, piece WHERE
            piece.ID=IDpiece AND IDmaison=" + idMaison);
            while (rs.next()) {
                int id = rs.getInt("ID");
                int idPiece = rs.getInt("IDpiece");
                String nom = rs.getString("nom");
                int temperature = rs.getInt("temperature");
                Piece p = listPiece.get(idPiece);
                new TechnoBain(id,p,TypeEtat.OFF,temperature,nom);
            }

            rs = st.executeQuery("SELECT * FROM elemfour, piece WHERE
            piece.ID=IDpiece AND IDmaison=" + idMaison);
            while (rs.next()) {
                int id = rs.getInt("ID");
                int idPiece = rs.getInt("IDpiece");
                String nom = rs.getString("nom");
                int temperature = rs.getInt("temperature");
                TypeEtat etat=(rs.getInt("etat")==1 ? TypeEtat.ON :
                TypeEtat.OFF);
                Piece p = listPiece.get(idPiece);
                new TechnoFour(id,p,temperature,etat,nom);
            }

            rs = st.executeQuery("SELECT * FROM elemlumiere, piece WHERE
            piece.ID=IDpiece AND IDmaison=" + idMaison);
            while (rs.next()) {
                int id = rs.getInt("ID");
                int idPiece = rs.getInt("IDpiece");
                String nom = rs.getString("nom");
                int variateur = rs.getInt("variateur");
                TypeEtat etat=(rs.getInt("etat")==1 ? TypeEtat.ON :
                TypeEtat.OFF);
                Piece p = listPiece.get(idPiece);
                new TechnoLumiere(id,p,variateur,etat,nom);
            }
        }
    }

```

```

        rs = st.executeQuery("SELECT * FROM elemchauffage, piece
        WHERE piece.ID=IDpiece AND IDmaison=" + idMaison);
        while (rs.next()) {
            int id = rs.getInt("ID");
            int idPiece = rs.getInt("IDpiece");
            String nom = rs.getString("nom");
            int temperature = rs.getInt("temperature");
            TypeEtat etat=(rs.getInt("etat")==1 ? TypeEtat.ON :
                TypeEtat.OFF);
            Piece p = listPiece.get(idPiece);
            new TechnoChauffage(id,p,temperature,etat,nom);
        }

    } catch (SQLException e) {
        System.out.println("ERR BDD 2");
        e.printStackTrace();
    }
}

/**
 * get le volume actuel d'un objet (tele ou autre)
 * @param elem element duquel on obtient le volume
 * @param id id de l'element
 * @return
 */
public int actualVol(String elem, int id) {
    int volume = 0;
    try {
        rs = st.executeQuery("SELECT volume FROM " + elem + " where
        ID="
            + id);
        while (rs.next()) {
            volume = rs.getInt(1);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return volume;
}

/**
 * changer le volume de n'importe quel objet
 * @param NewVol volume souhaite
 * @param elem element dont on veut changer le volume
 * @param id id de l'objet
 */
public void changeVol(int NewVol, String elem, int id) {
    try {
        st.executeUpdate("Update " + elem + " set volume=" + NewVol
            + " where ID =" + id);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

```

/**
 * get temp min
 */
public int tempMin() {
    int temp = 0;
    try {
        rs = st.executeQuery("SELECT min FROM contextetype where ID
                             = 1");
        while (rs.next()) {
            temp = rs.getInt(1);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return temp;
}

/**
 *get temp max
 */
public int tempMax() {
    int temp = 0;
    try {
        rs = st.executeQuery("SELECT max FROM contextetype where ID
                             = 1");
        while (rs.next()) {
            temp = rs.getInt(1);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return temp;
}

/**
 * modifier la temp
 * @param NewTemp
 * @param id
 * @param idp
 */
public void changeTemp(int NewTemp, int id,int idp) {
    try {
        st.executeUpdate("Update elemchauffage set temperature=" +
                          NewTemp
                          + " where ID =" + id);
        st.executeUpdate("insert into piececontexte values (" +idp+"
                          ,1,"+NewTemp+"");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

/**
 * changer la chaine actuelle de la tele ou autre
 * @param elem

```

```

    * @param chaineVoulue
    * @param id
    */
public void changeChannel(String elem, int chaineVoulue, int id) {
    try {
        st.executeUpdate("Update " + elem + " set chaine= " +
            chaineVoulue
            + " where ID =" + id);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

/**
 * changer le degre de luminosite d'une piece
 * @param id
 * @param lumVoulue
 */
public void changeLumino(int id, int lumVoulue) {
    try {
        st.executeUpdate("Update elem lumiere set variation=" +
            lumVoulue
            + " where ID =" + id);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}
}

```

Classe JHashtable

```

3 package com.GetEtat.servlets;

import java.util.Hashtable;
import java.util.Iterator;
import java.util.Vector;

public class JHashtable {
    private Hashtable ht;

    public JHashtable(){}
    public JHashtable(Hashtable ht){this.ht=ht;}
    public JHashtable(JHashtable jht){ht=jht.toHashtable();}

    public Hashtable toHashtable(){ return ht; }
    public Object get(Object key){ return ht.get(key); }
    public JHashtable getAndPut(Object key){
        if(ht.get(key) == null)
            ht.put(key, new Hashtable());
        return new JHashtable((Hashtable) ht.get(key));
    }
    public Object put(Object key, Object value){ return ht.put(key, value); }
    public Object remove(Object key){ return ht.remove(key); }

    public JHashtable maisons(){
        return new JHashtable(ht).getAndPut("Maisons");
    }

    public JHashtable maison(int idMaison){
        return maisons().getAndPut(String.valueOf(idMaison));
    }

    public JHashtable protocoles(int idMaison){
        return maison(idMaison).getAndPut("Protocoles");
    }

    public JHashtable protocole(int idMaison, String nameProtocole){
        return protocoles(idMaison).getAndPut(nameProtocole);
    }

    public JHashtable action(int idMaison, String nameProtocole, String
        nameAction){
        return protocole(idMaison, nameProtocole).getAndPut(nameAction);
    }

    public Vector<String> listeProtocoles(int idMaison){
        Vector<String> v = new Vector<String>();
        if(maisons().get(idMaison)!=null){
            Iterator it;
            it=protocoles(idMaison).toHashtable().keySet().iterator();
            while(it.hasNext())
            {
                Object key=it.next();
                v.add(key.toString());
            }
        }
    }
}

```

```

        return v;
    }
    public Vector<String> listeActions(int idMaison, String nomProtocole){
        Vector<String> v = new Vector<String>();
        if(maisons().get(idMaison)!=null && protocoles(idMaison).get(
            nomProtocole)!=null){
            Iterator it;
            it=protocole(idMaison, nomProtocole).toHashtable().keySet().
                iterator();
            while(it.hasNext())
            {
                Object key=it.next();
                v.add(key.toString());
            }
        }
        return v;
    }
}

```

4. Classe textitPièce

Classe Pièce

```
package com.GetEtat.beanstestnewversion;

import java.util.*;

/**
 * @author Yann
 *
 */
public class Piece {

    private int idPiece;
    private String nomPiece;
    protected int tempMin;
    protected int tempMax;
    private Vector<ElementsOfPiece> allElementsOfPiece;

    public Piece(int id){
        this(id, "Ma Piece "+id);
    }

    /**
     *
     * @param id identifiant de la piece
     */
    public Piece(int id, String nomPiece){
        this(id,nomPiece,10,25);
    }

    public Piece(int id, String nomPiece, int tempMin, int tempMax){
        idPiece=id;
        this.nomPiece = nomPiece;
        this.tempMin=tempMin;
        this.tempMax=tempMax;
        allElementsOfPiece=new Vector<ElementsOfPiece>();
    }

    /**
     *
     * @return Retourne l'identifiant de la p i c e
     */
    public int getIdPiece(){
        return idPiece;
    }

    public String getNomPiece(){
        return nomPiece;
    }

    public void setNomPiece(String newNomPiece){
        this.nomPiece = newNomPiece;
    }

    /**
     *
     */
}
```

```

    * @param tempMin : temperature minimum    modifier
    */
    public boolean setTempMin(int tempMin){
        if(tempMin <= tempMax)
            this.tempMin=tempMin;
        else
            return false;
        return true;
    }

    /**
     *
     * @param tempMax : temperature maximum    modifier
     */
    public boolean setTempMax(int tempMax){
        if(tempMax >= tempMin)
            this.tempMax=tempMax;
        else
            return false;
        return true;
    }

    /**
     * @return la temperature min de la piece
     */
    public int getTempMin() {
        return tempMin;
    }

    /**
     * @return la temperature max de la piece
     */
    public int getTempMax() {
        return tempMax;
    }

    /**
     * @return la liste d'elements de la piece
     */
    public Vector<ElementsOfPiece> getAllElementsOfPiece() {
        return allElementsOfPiece;
    }

    /**
     *
     * @param t element quelconque de la piece
     */
    public void addElement(ElementsOfPiece e){
        if(!e.equals(null) && !allElementsOfPiece.contains(e)){
            allElementsOfPiece.add(e);
            e.setPiece(this);
        }
    }
    protected void supprElement(ElementsOfPiece e){
        allElementsOfPiece.remove(e);
    }

    /**

```



```

        * Detruit tous les objets de la piece
        */
    public void finalize(){
        for(int i = 0; i < allElementsOfPiece.size(); i++){
            allElementsOfPiece.get(i).finalize();
        }
    }
}

```

5. Classe textitTechnofour

Classe TechnoFour

```

package com.GetEtat.beanstestnewversion;

import enums.TypeEtat;

/**
 *
 * @author Clément
 */
public class TechnoFour extends Technos{
    private int temperature;

    /**
     * constructeur avec l'identifiant du four <br>
     * fera appel au constructeur prenant en parametre un temperature de 0 par
     * default
     * @param id : identifiant du four
     */
    public TechnoFour(int id){
        this(id, "Four"+id);    // temperature a 0 par d faut
    }

    public TechnoFour(int id, String nomFour){
        this(id, 0, nomFour);    // temperature a 0 par d faut
    }

    /**
     *
     * constructeur avec l'identifiant et la temperature du four <br>
     * fera appel au constructeur prenant en parametre une piece nulle par
     * default
     * @param id : identifiant du four
     * @param temperature : temperature du four
     */
    public TechnoFour(int id, int temperature, String nomFour){
        this(id, null ,temperature, nomFour);
    }

    /**
     * constructeur avec l'identifiant , la temperature du four et la piece
     * concernee <br>
     * fera appel au constructeur prenant en parametre l'etat du four, eteint
     * par default.
     * @param id : identifiant du four
     * @param piece : piece dans laquelle se situe le four

```

```

        * @param temperature : temperature du four
        */
    public TechnoFour(int id, Piece piece, int temperature, String nomFour){
        this(id, piece, temperature, TypeEtat.OFF, nomFour); // teind par
            d faut
    }
    /**
     *
     * constructeur avec l'identifiant , la temperature et l'etat du four et la
     piece concernee <br>
     * @param id : identifiant du four
     * @param piece : piece dans laquelle se situe le four
     * @param temperature : temperature du four
     * @param etat : etat du four
     */
    public TechnoFour(int id, Piece piece, int temperature, TypeEtat etat,
        String nomFour){
        super(id, piece, etat, nomFour);
        this.temperature = temperature;
    }

    /**
     * @param temperature : temperature voulue
     * @return Retourne si le changement a t fait ou pas.
     */
    public boolean setTemperature(int temperature) {
        if(temperature > 0 && temperature <280 )
            this.temperature = temperature;
        else
            return false;
        return true;
    }

    /**
     *
     * @return retourne la temperature actuelle du four
     */
    public int getTemperature() {
        return temperature;
    }
}

```

Table *contextetype*

```

INSERT INTO 'contextetype' ('ID', 'nom', 'unite', 'min', 'max') VALUES
(1, 'temperature', ' C ', -50, 50),
(2, 'luminosit ', NULL, NULL, NULL),
(3, 'surface', 'm ', 1, NULL),
(4, 'hauteursousplafond', 'm', 1, NULL),
(5, 'bruit', 'db', 0, 140),
(6, 'humidit ', '%', 0, 100),
(7, 'detecteurmouvement', NULL, 0, 1);

```