Devoir de MOCI :

Gestionnaire de stages

Gogdemir Romain
Hugel Benjamin

Sommaire

I – Introduction

Ce devoir a pour but la réalisation des diagrammes de séquence dans trois cas différents, d'un diagramme d'état et du diagramme de classes d'un gestionnaire de stages pour l'école Telecom Nancy. Suite à la réalisation d'un cahier des charges, il est judicieux de construire ces diagrammes afin de faciliter la programmation d'une application. En effet, tous les cas d'utilisation sont recensés dans les diagrammes de séquence, les diagrammes d'état permettent d'avoir une vue d'ensemble du fonctionnement de l'application et le diagramme de classes permet d'avoir une bonne organisation de la programmation. Il s'agit d'un travail préparatoire indispensable au développement d'une application ou d'un logiciel.

Les diagrammes de séquence correspondent aux trois interactions suivantes :

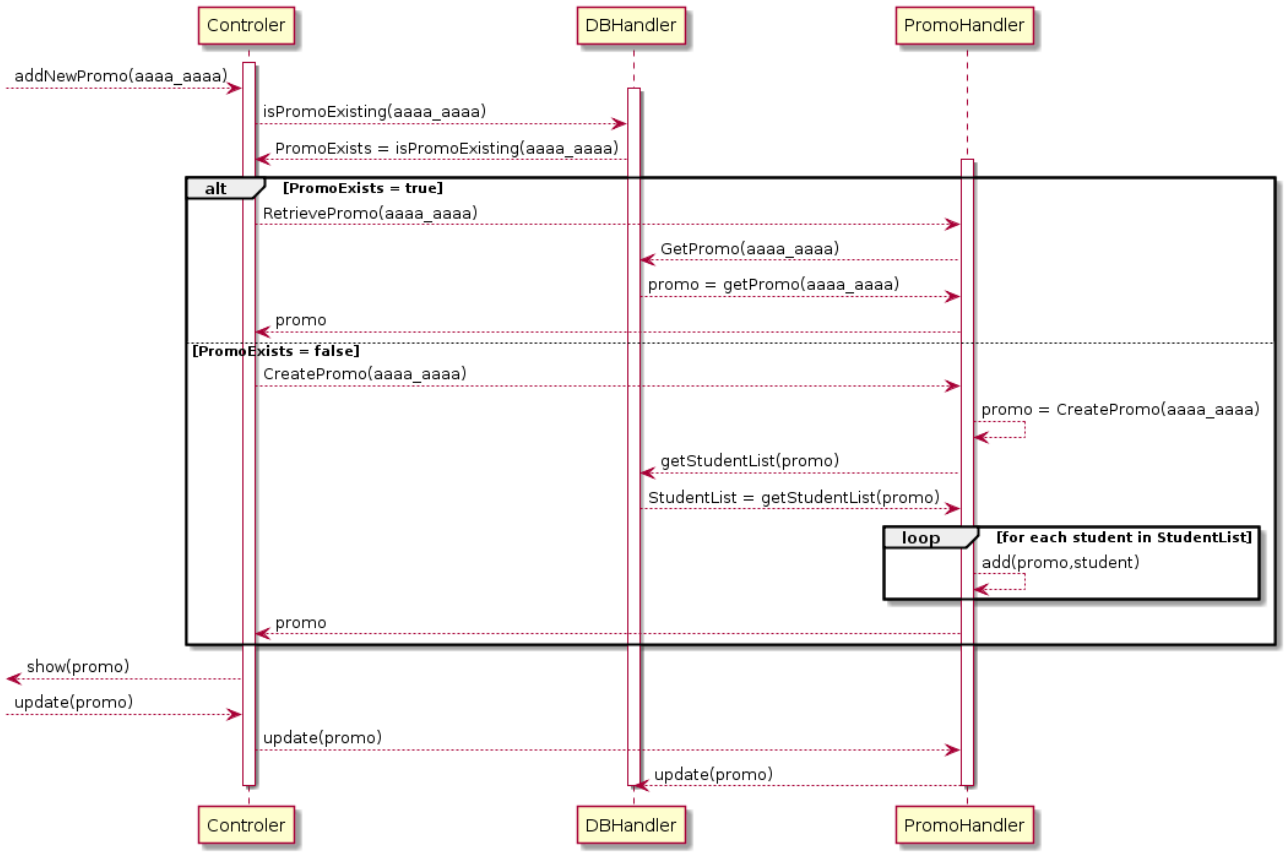
- Création d'une nouvelle promotion
- Saisie des données d'une fiche de renseignement
- Prise de décision concernant la validité ou non d'une fiche de renseignement

Le diagramme d'état correspond à la vie d'une fiche de renseignement, de sa complétion à la signature d'une convention de stage.

Le diagramme de classe retrace les classes impliquées dans les diagrammes cités précédemment, les méthodes qu'elles utilisent ainsi que les liens qui les unissent.

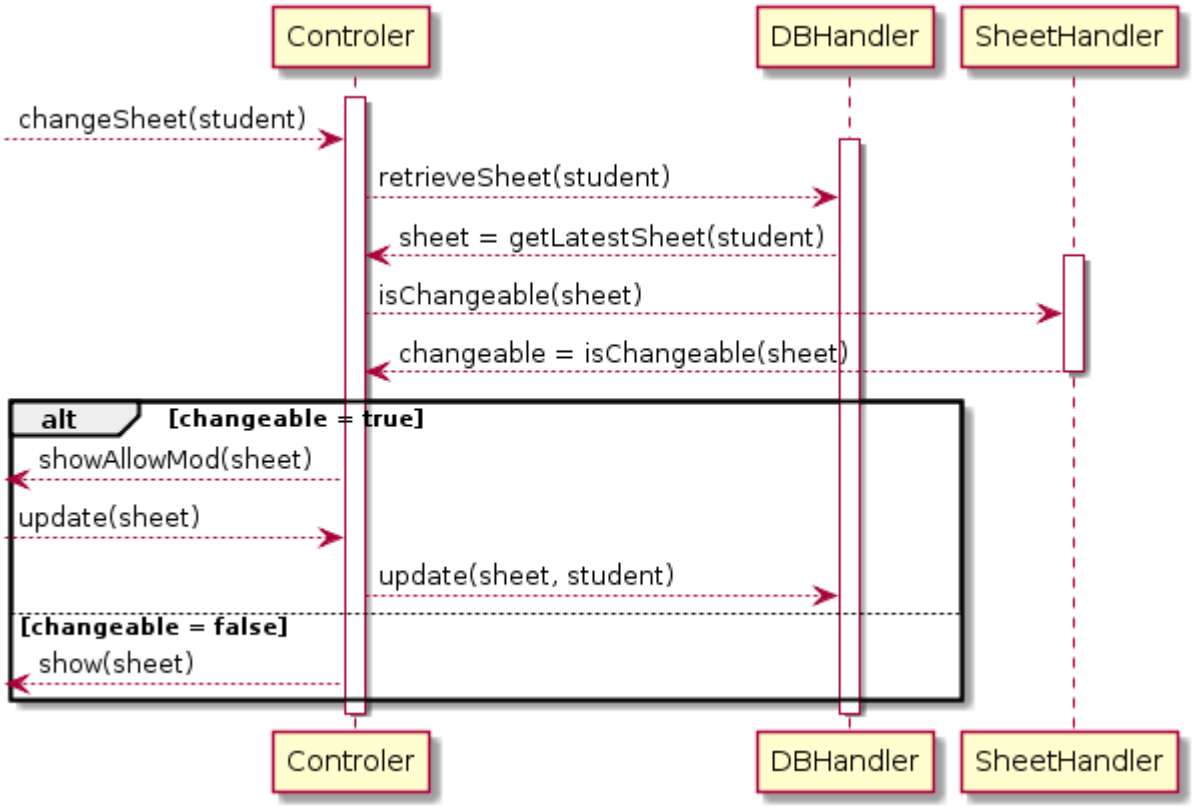
II – Diagrammes de séquence

Mise en place d'une nouvelle promotion.



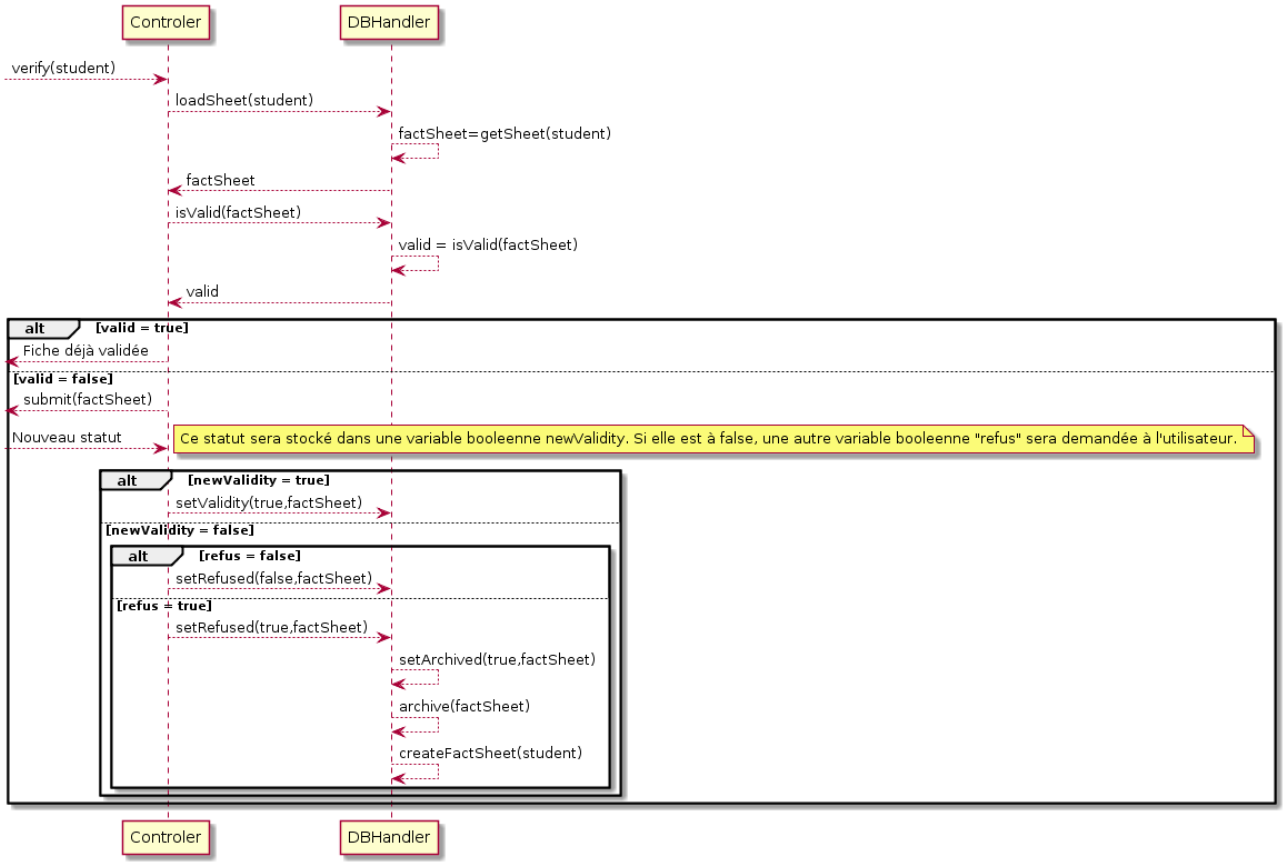
Si la promotion existe déjà, on renvoie la promotion existante. Sinon, on la créée. Une promotion contient la liste des étudiants de l'année demandée récupérée dans la base de données. Chaque étudiant se voit créer une fiche de renseignement vierge au moment de la création de la promotion, dans la fonction add(promo,student).

Saisie de données dans la fiche de renseignement.



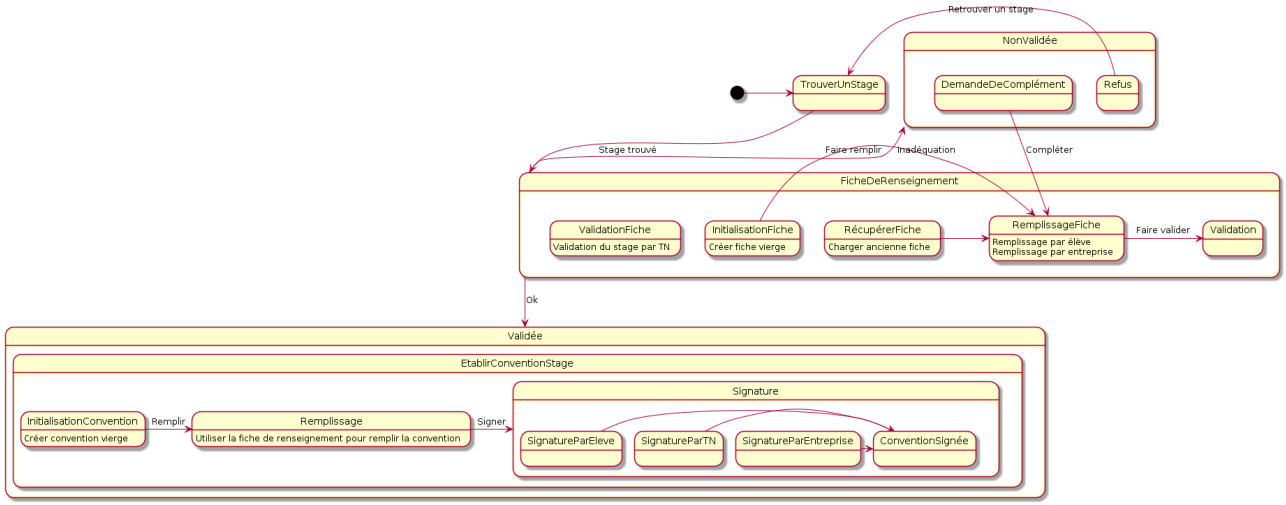
La dernière fiche de renseignement mise en place par l'élève choisi est montrée à l'utilisateur, avec droit ou non à la modification selon l'état de la fiche de renseignement : une fiche qui a abouti à une convention de stage ne doit pas pouvoir être modifiée tant que la convention n'a pas été refusée ou le stage terminé.

Prise de décision concernant une fiche de renseignement.

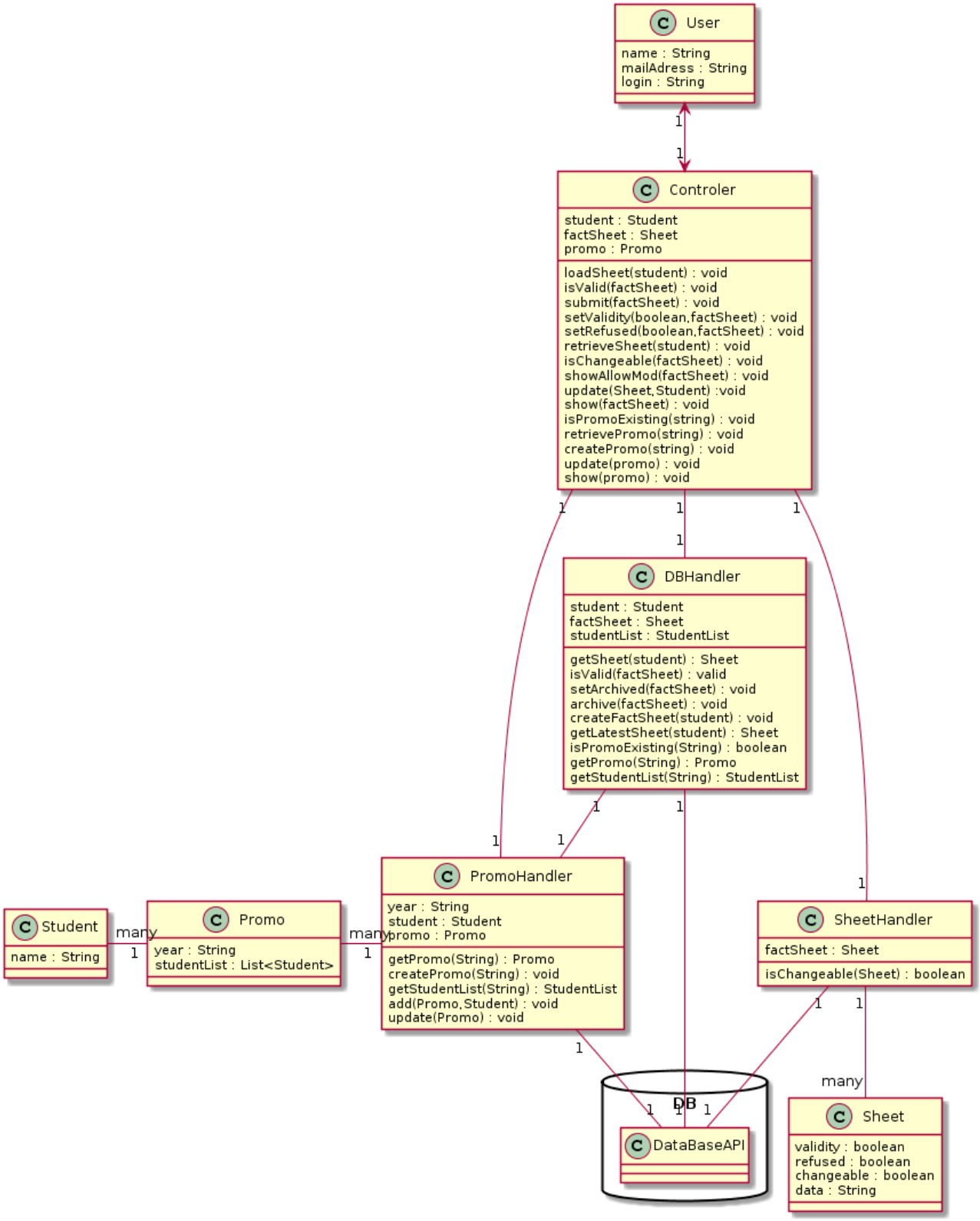


Une fiche de renseignement doit être validée. Si elle ne l'est pas encore, elle peut également être refusée ou en attente de compléments.
Lors de la prise de décision, on demande déjà à l'utilisateur si la fiche est validée ou non, puis, dans le cas où elle n'est pas validée, on demande à l'utilisateur si la fiche est refusée ou non. Si elle n'est pas refusée, elle sera en attente de compléments.

III – Diagramme d'état



IV – Diagramme de classes



Annexe – Codes PlantUML

Mise en place d'une nouvelle promotion :

```
@startuml

participant Controler
participant DBHandler
participant PromoHandler

note left : aaaa_aaaa is a variable containing the promo year.

        activate Controler
        --> Controler : addNewPromo(aaaa_aaaa)
        activate DBHandler
        Controler --> DBHandler : isPromoExisting(aaaa_aaaa)
        DBHandler --> Controler : PromoExists =
isPromoExisting(aaaa_aaaa)
        activate PromoHandler

        alt PromoExists = true
                Controler --> PromoHandler : RetrievePromo(aaaa_aaaa)
                PromoHandler --> DBHandler : GetPromo(aaaa_aaaa)
                DBHandler --> PromoHandler : promo = getPromo(aaaa_aaaa)
                PromoHandler --> Controler : promo

        else PromoExists = false
                Controler --> PromoHandler : CreatePromo(aaaa_aaaa)
                PromoHandler --> PromoHandler : promo =
CreatePromo(aaaa_aaaa)
                PromoHandler --> DBHandler : getStudentList(promo)
                DBHandler --> PromoHandler : StudentList =
getStudentList(promo)
                loop for each student in StudentList
                        PromoHandler --> PromoHandler : add(promo,student)
                end
```

```
        PromoHandler --> Controler : promo

    end

    <-- Controler : show(promo)
    --> Controler : update(promo)
    Controler --> PromoHandler : update(promo)
    PromoHandler --> DBHandler : update(promo)
    deactivate DBHandler
    deactivate PromoHandler
    deactivate Controler

@enduml
```

Saisie de données dans la fiche de renseignement :

```
@startuml

participant Controler
participant DBHandler
participant SheetHandler

        note left : student contains the id of the student.
        activate Controler
        --> Controler : changeSheet(student)
        activate DBHandler
        Controler --> DBHandler : retrieveSheet(student)
        DBHandler --> Controler : sheet = getLatestSheet(student)
        activate SheetHandler
        Controler --> SheetHandler : isChangeable(sheet)
        SheetHandler --> Controler : changeable = isChangeable(sheet)
        deactivate SheetHandler
        alt changeable = true
                <-- Controler : showAllowMod(sheet)
                --> Controler : update(sheet)
                Controler --> DBHandler : update(sheet, student)
        else changeable = false
                <-- Controler : show(sheet)
        end
        deactivate DBHandler
        deactivate Controler

@enduml
```

Prise de décision concernant une fiche de renseignement :

```
@startuml

-->Controler : verify(student)
Controler-->DBHandler : loadSheet(student)
DBHandler-->DBHandler : factSheet=getSheet(student)
DBHandler-->Controler : factSheet
Controler-->DBHandler : isValid(factSheet)
DBHandler-->DBHandler : valid = isValid(factSheet)
DBHandler-->Controler : valid

alt valid = true
        <--Controler : Fiche déjà validée
else  valid = false
        <--Controler : submit(factSheet)
        -->Controler : Nouveau statut

        note left : Ce statut sera stocké dans une variable booleenne
newValidity. Si elle est à false, une autre variable booleenne "refus" sera
demandée à l'utilisateur.

alt newValidity = true
                Controler-->DBHandler : setValidity(true,factSheet)
        else newValidity = false
                alt refus = false
                        Controler --> DBHandler : setRefused(false,factSheet)
                                else refus = true
                        Controler --> DBHandler : setRefused(true,factSheet)
                        DBHandler --> DBHandler : setArchived(true,factSheet)
                        DBHandler --> DBHandler : archive(factSheet)
                        DBHandler --> DBHandler : createFactSheet(student)
                end
        end
end

@enduml
```

Diagramme d'état :

@startuml

[*]->TrouverUnStage
TrouverUnStage->FicheDeRenseignement : Stage trouvé
state FicheDeRenseignement {
	state InitialisationFiche
	InitialisationFiche : Créer fiche vierge
	state RécupérerFiche
	RécupérerFiche : Charger ancienne fiche
	RécupérerFiche->RemplissageFiche
	InitialisationFiche->RemplissageFiche : Faire remplir
	state RemplissageFiche
	RemplissageFiche : Remplissage par élève
	RemplissageFiche : Remplissage par entreprise
	RemplissageFiche->Validation : Faire valider
	state ValidationFiche
	ValidationFiche : Validation du stage par TN
}

FicheDeRenseignement->NonValidée : Inadéquation

FicheDeRenseignement->Validée : Ok

state NonValidée {
	state Refus
	state DemandeDeComplément
}

Refus->TrouverUnStage : Retrouver un stage
DemandeDeComplément->RemplissageFiche : Compléter

state Validée {
	state EtablirConventionStage {
		state InitialisationConvention
		InitialisationConvention : Créer convention vierge
		InitialisationConvention->Remplissage : Remplir
		state Remplissage

```
        Remplissage : Utiliser la fiche de renseignement pour remplir la
convention
        Remplissage->Signature : Signer
        state Signature {
            state SignatureParEleve
            state SignatureParTN
            state SignatureParEntreprise
            SignatureParEleve->ConventionSignée
            SignatureParTN->ConventionSignée
            SignatureParEntreprise->ConventionSignée
            state ConventionSignée
    }


@enduml
```

Diagramme de classes :

@startuml

class Controler{
student : Student
factSheet : Sheet
promo : Promo

loadSheet(student) : void
isValid(factSheet) : void
submit(factSheet) : void
setValidity(boolean,factSheet) : void
setRefused(boolean,factSheet) : void
retrieveSheet(student) : void
isChangeable(factSheet) : void
showAllowMod(factSheet) : void
update(Sheet,Student) :void
show(factSheet) : void
isPromoExisting(string) : void
retrievePromo(string) : void
createPromo(string) : void
update(promo) : void
show(promo) : void
}

class User{
name : String
mailAdress : String
login : String
}

class DBHandler{
student : Student
factSheet : Sheet
studentList : StudentList

getSheet(student) : Sheet
isValid(factSheet) : valid

setArchived(factSheet) : void
archive(factSheet) : void
createFactSheet(student) : void
getLatestSheet(student) : Sheet
isPromoExisting(String) : boolean
getPromo(String) : Promo
getStudentList(String) : StudentList
}

class SheetHandler{
factSheet : Sheet
isChangeable(Sheet) : boolean
}

class PromoHandler{
year : String
student : Student
promo : Promo

getPromo(String) : Promo
createPromo(String) : void
getStudentList(String) : StudentList
add(Promo,Student) : void
update(Promo) : void
}

class Promo{
year : String
studentList : List<Student>
}

class Student{
name : String
}

class Sheet{
validity : boolean
refused : boolean
changeable : boolean

```
data : String
}

package DB <<Database>>{
        class DataBaseAPI{

        }
}

Controler "1" -- "1" DBHandler
Controler "1" <-up-> "1" User
Controler "1" -- "1" SheetHandler
Controler "1" -- "1" PromoHandler

DBHandler "1" -- "1" PromoHandler
DBHandler "1" -- "1" DataBaseAPI

PromoHandler "1" -left- "many" Promo
PromoHandler "1" -- "1" DataBaseAPI

SheetHandler "1" -- "many" Sheet
SheetHandler "1" -- "1" DataBaseAPI

Promo "1" -left- "many" Student

@enduml
```