**Organisation du fichier sources**

Table du contenu

#### **L’architecture MVC**

Model

View

Controller............................................................................................................................................

#### **L’architecture OMA**

Organism

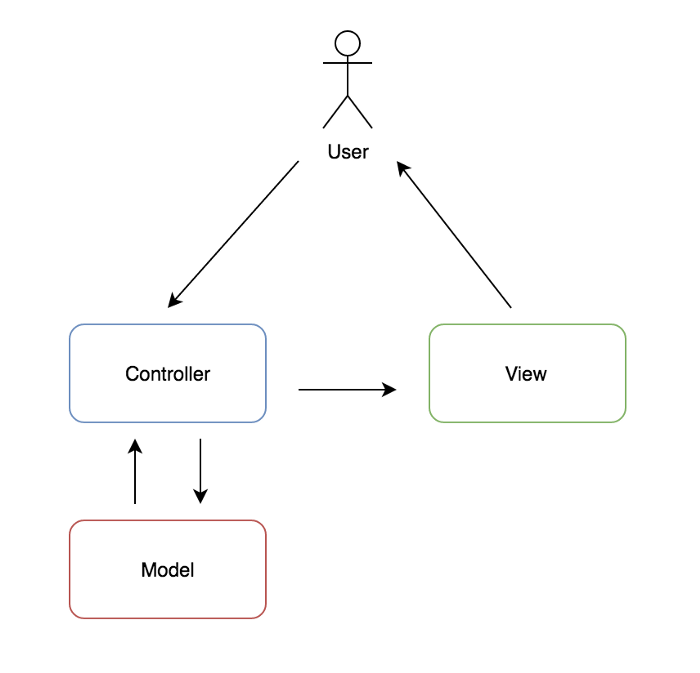
Molécule

#### Atoms

# **Architecture MVC**

# **Définition :** MVC signifie Model-View-Controller. Il s’agit d’une architecture ou d’un modèle de conception logicielle qui facilite la création d’applications énormes. Il n’appartient pas à un langage de programmation ou à un Framework spécifique, mais c’est un concept que vous pouvez utiliser pour créer tout type d’application ou de logiciel dans n’importe quel langage de programmation.

Le schéma suivant représente l’architecture MVC du site ainsi que les différents composants qui le permettent de fonctionner correctement.



*Figure 1 : Architecture MVC*

## **Model - base de données**

* Le modèle fonctionne directement avec la base de données. Il n’a pas à traiter de l’interface utilisateur ou du traitement des données. Dans un scénario réel, vous utiliserez simplement le modèle pour récupérer, insérer, mettre à jour et supprimer des données de votre base de données et définit les données et la logique applicative :
* Encapsule les données de l'application
* Exécute les commandes de mise à jour des données
* Répond aux requêtes de consultation

## **Vue – client**

- En termes simples, Vue est l’interface utilisateur sur laquelle notre client / utilisateur peut effectuer certaines actions. Il contient HTML, CSS, JS, XML ou tout autre langage de balisage que nous pouvons utiliser pour créer une belle interface utilisateur. Il contient également du code pour afficher les données qu’il reçoit de notre application, elle présente graphiquement les données du modèle :

* Se rafraîchit à la demande.
* Fournit des informations complémentaires lors de l'action de l'utilisateur.

## **3. Contrôleur - server**

## - Le contrôleur est la partie dans laquelle nous traitons les données après avoir reçu une demande de View et avant de mettre à jour quoi que ce soit dans notre base de données avec notre modèle.

* Récupère des informations à partir de la vue
* Traduit les actions de l'utilisateur en commandes de modification du modèle

## Dans notre application l’architecture se concrétise dans les fichiers suivants :

## Contrôleur -> le fichier “controllers”.

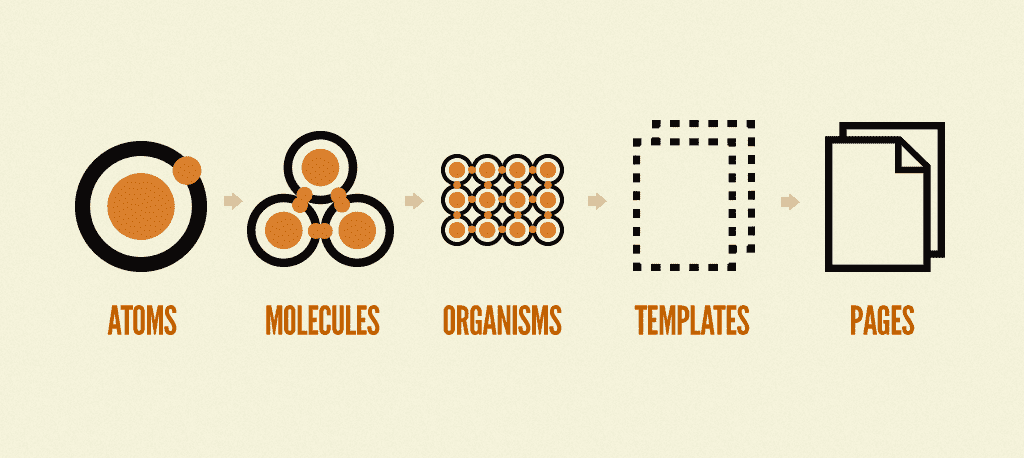
## Vue -> les fichiers “components”.

## Model -> les fichiers “models”.

# **Le pattern Atomic Design**

# **Définition :** L’idée derrière la méthodologie est de considérer les composants comme des blocs de construction. Les composants sont rassemblés, allant de composants simples à faible complexité à des composants plus complexes. Les principaux avantages derrière ce modèle architectural sont la cohérence de la conception (par exemple, les champs d’entrée se ressemblent toujours) ainsi que la réutilisabilité (en particulier parmi les composants de l’atome et de la molécule qui ne sont pas fortement avisés). La méthode à ce principe consiste à décomposer votre conception en composants jusqu’à ce que le composant ne puisse plus être décomposé.

# Le schéma suivant représente architecture appelée **« Atomic design »**



*Figure 2 : Architecture Atomic design*

# **Organism**

# Les organismes sont constitués de plusieurs molécules et/ou atomes. Les organismes deviennent de plus en plus complexes. C’est là que l’état peut commencer à montrer sa présence (par exemple, un composant <Form /> ou un <Navigation> composant).

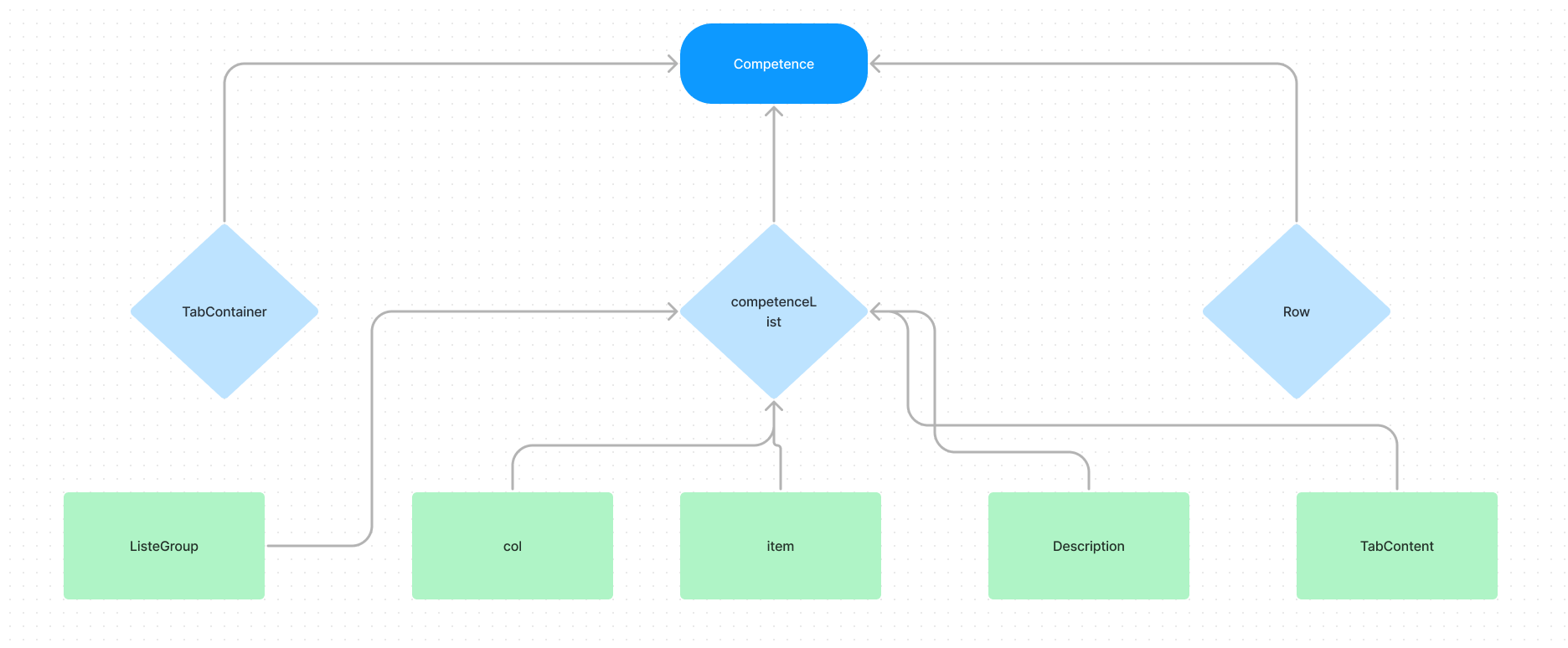
# **Liste des organismes :**

* **Competence :** un composant qui représente une competence d’un élevé sélectionnée à partir d’une liste de competence qu’elle réunit aussi deux autres composants qui sont la description et Item.

- les fonctions utilisées : **setCompetence** qui permet d’initialiser une competence basant sur son nombre.

**SelectionHandler** qui initialise une competence par la valeur sélectionnée (elle fait appel a la première méthode).

- Le schéma suivi :



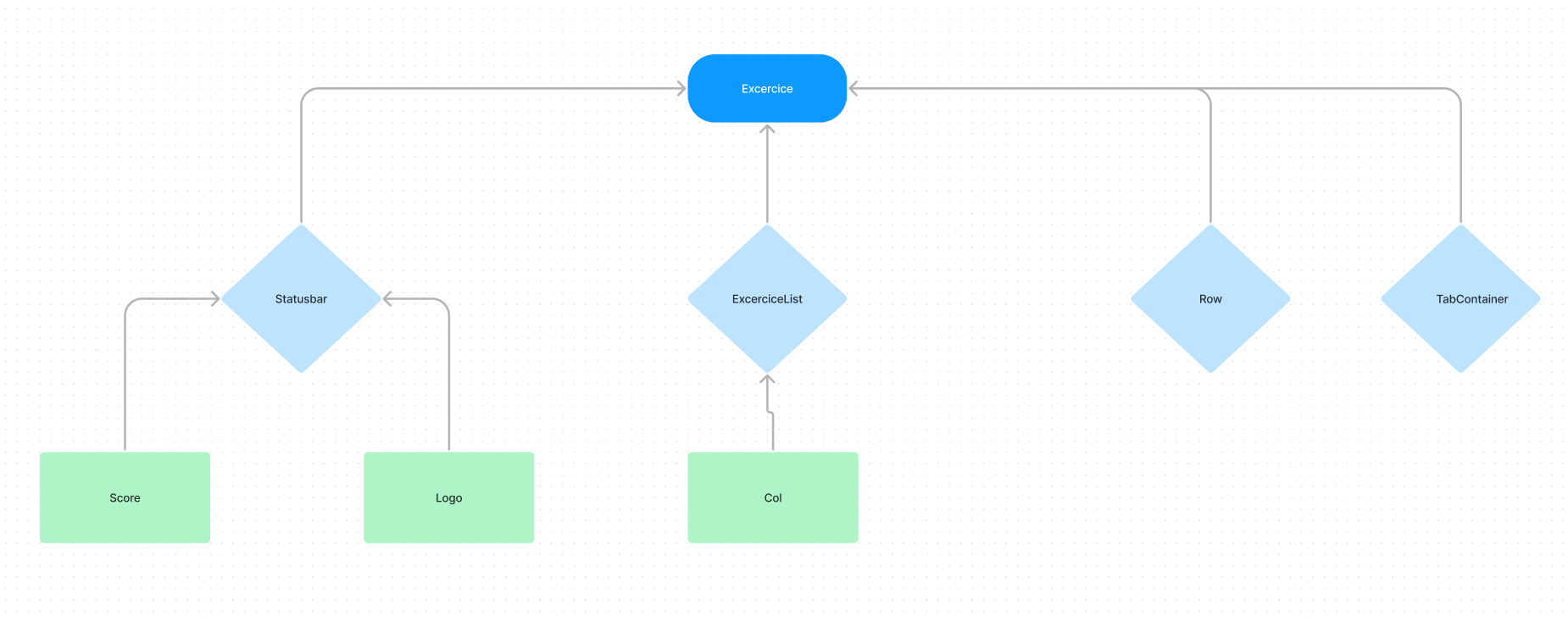
* **Excercice :** un composant constitue StatusBar et choisit à partir d’une liste d’exercices.

- les fonctions suivi : **SetExcercice** qui permet d'initialiser l’excercice sélectionné

**SetImage** qui modifier la valeur responsable de la navigation entre les images de transition.

**OpenExcercice** qui Redirige vers la page des questions et affiche les questions de l'exercice sélectionné étape par étape.

- le schéma suivi :



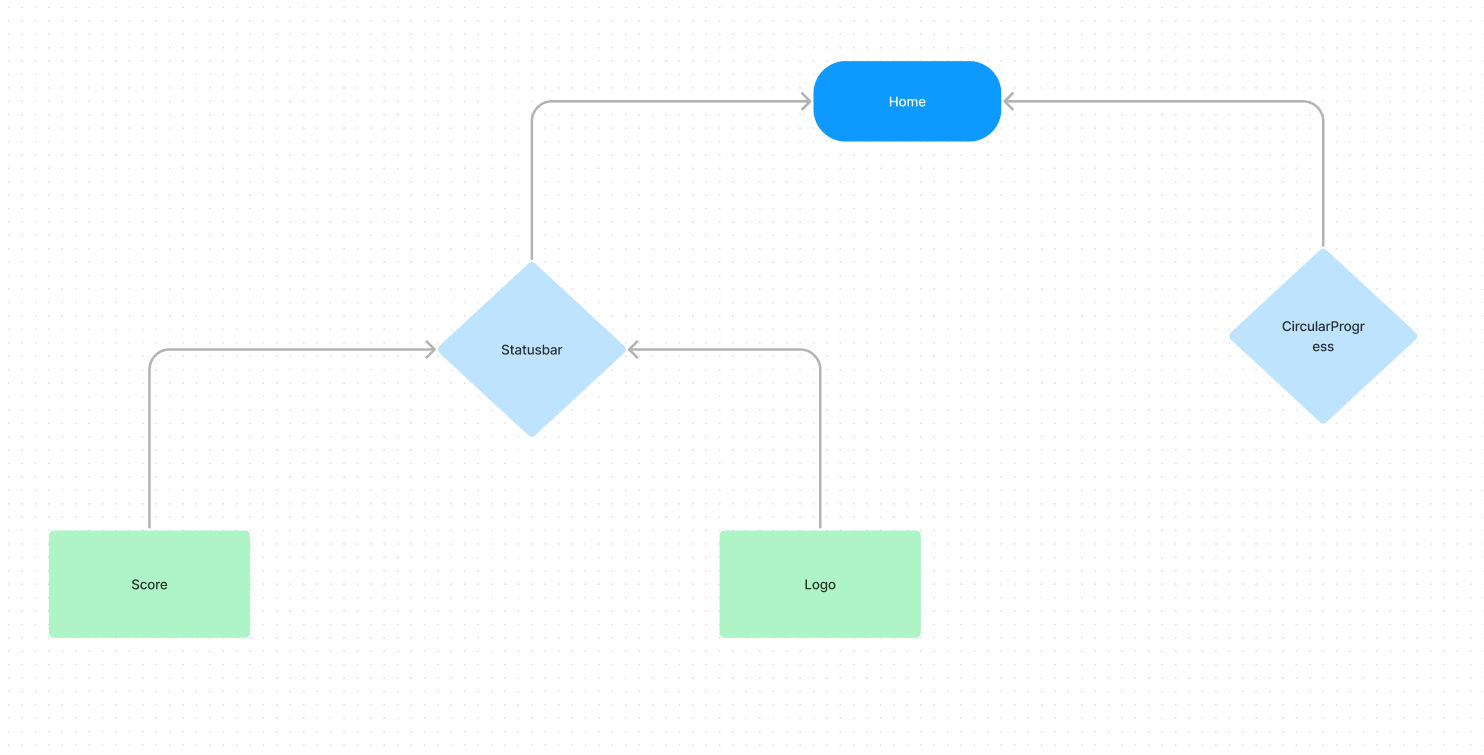
* **Home :** le composant graphique qui représente la vue de la page Home obtenue après le Login, on trouve les iles proposées à jouer.

- les fonctions utilisées : **setExcercice** qui permet de sélectionner l’excercice.

**SetIle** qui permet de sélectionner l’ile.

**SetDonneExcercice** qui vérifier que l’excercice est bien charge.

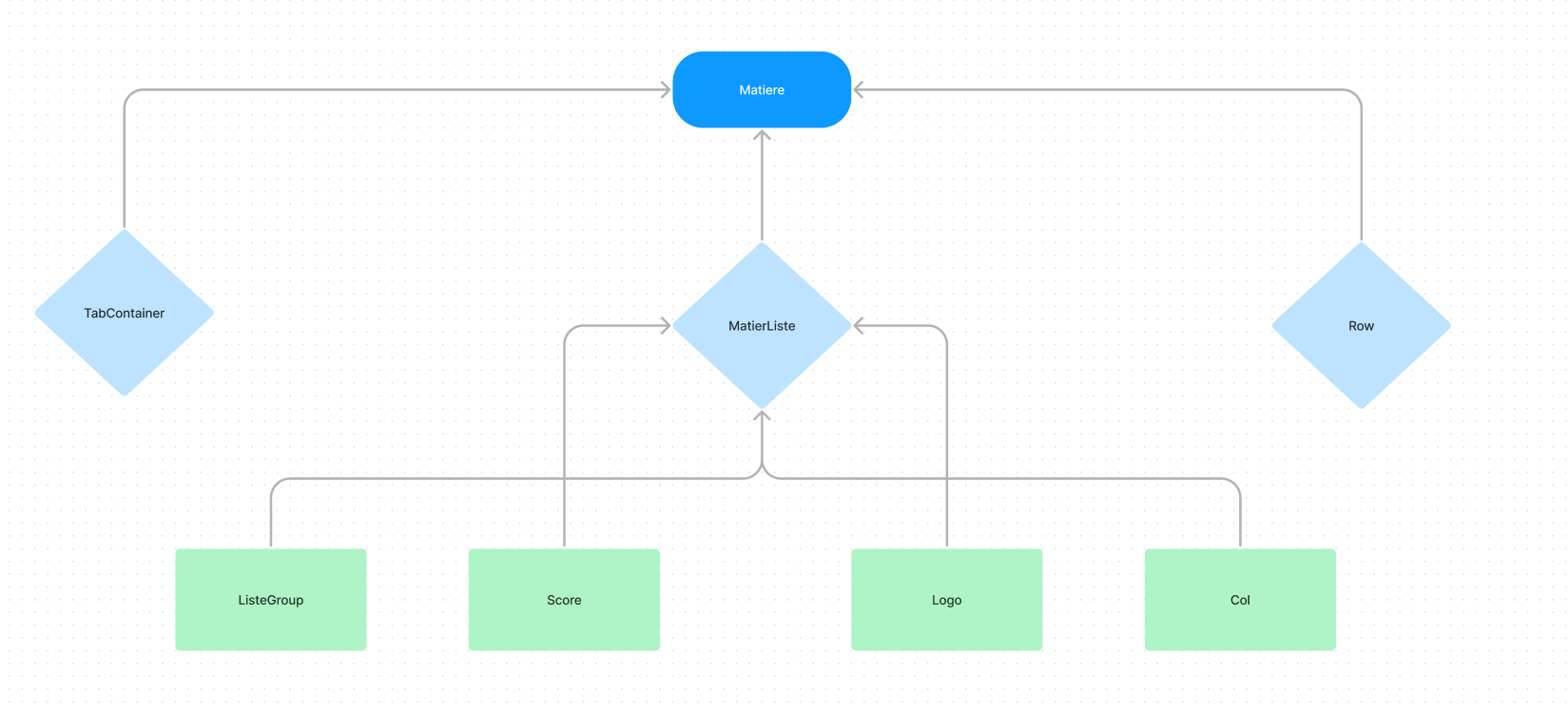
- le schéma suivi :



* **Matiere :** un composant qui permet de montrer la vue d’une matiere sélectionnée

- Les fonctions utilisées : **setMatiere** qui permet de sélectionner la matiere en basant sur l’identifiant “id” de l’objet.

- Le schéma suivi :



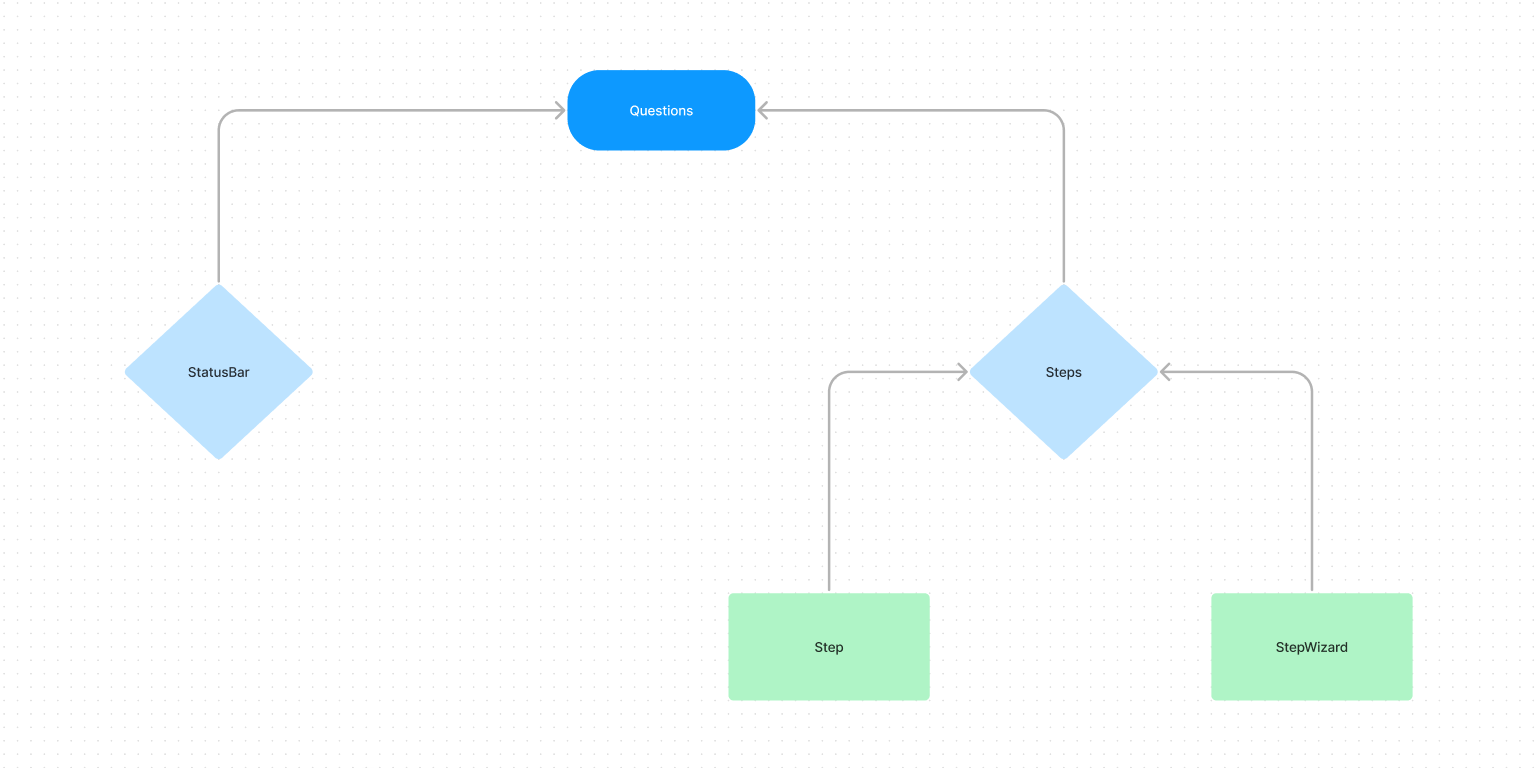
* **Question :** Récupère la liste des questions de l'exercice ainsi que le type d'exercice.

- les fonctions utilisées : **setExercice** initialise l'exercice sélectionné basant sur l’identifiant de l’objet sélectionnée.

**setQuestions** initialise la liste des questions basant sur la liste des questions.

**GetExoType** récupère le type de l'exercice sélectionné.

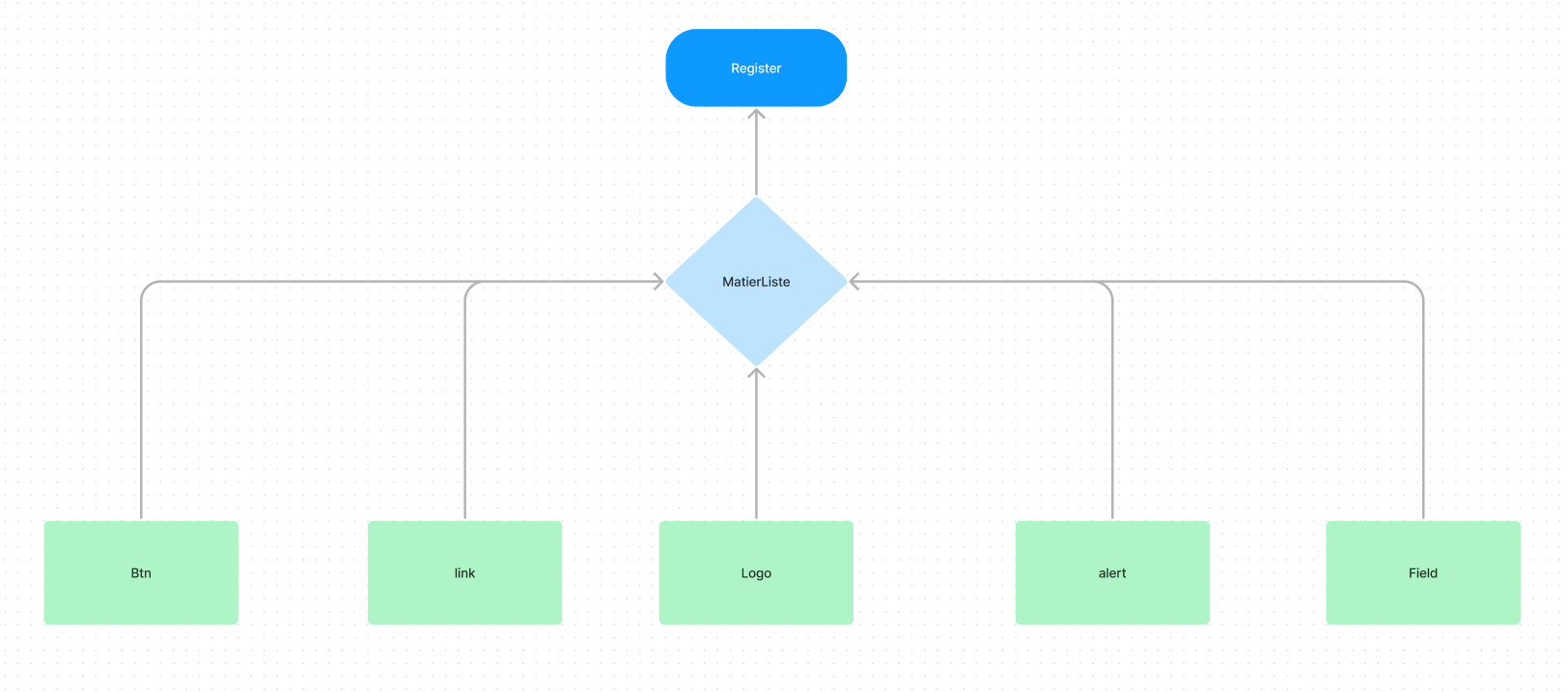
- le schéma suivi :



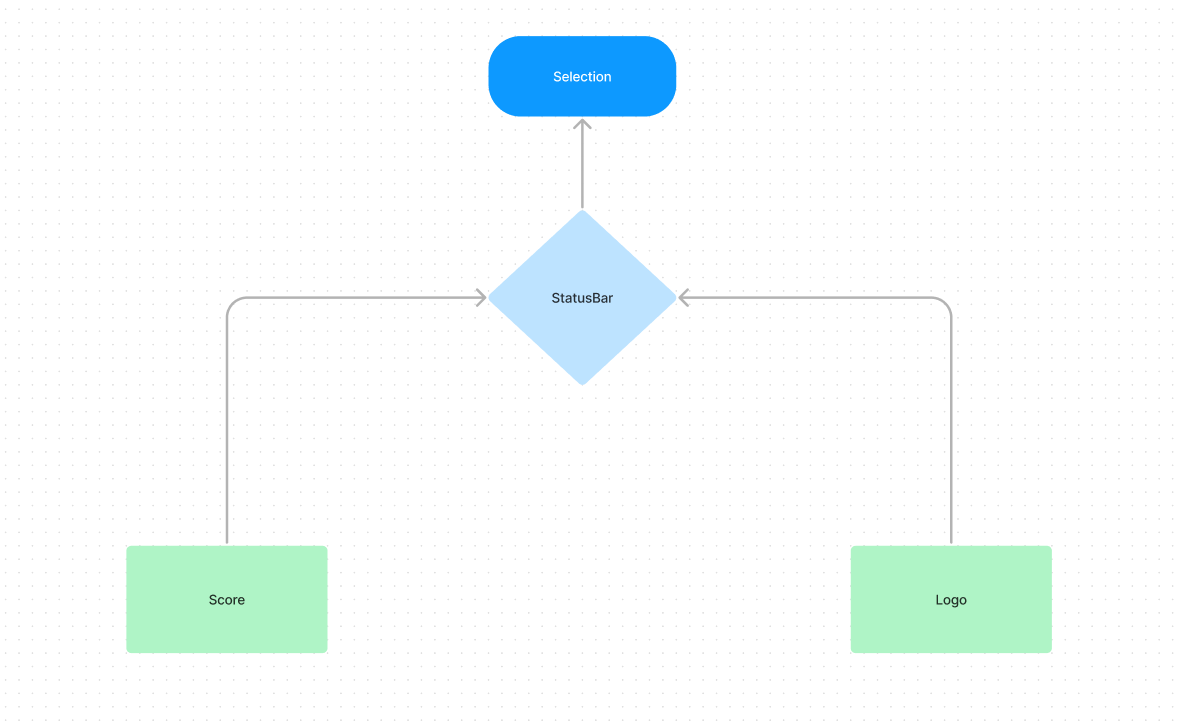
* **Register :** le composant responsable de l’enregistrement du client ”élève” base sur une registrform.

- le code utilise : une portion de code qui nous permet de vérifier la similarité des mots de passe entre et enregistre, l’existence de l’email et enfin de compte d’enregistrer les informations mentionnées et crypter le mot de passe par la fonction **bcrypt.**

- le schéma suivi :



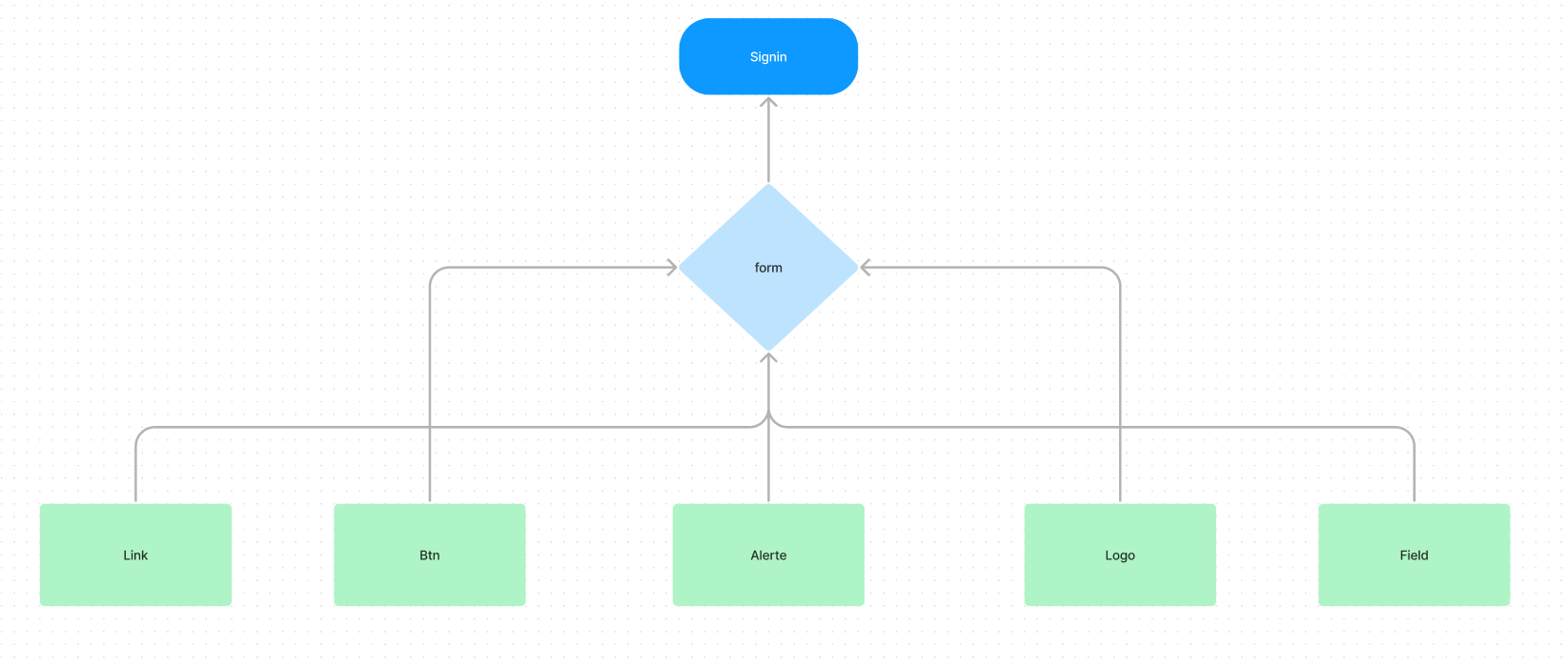
* **Selection :**



* **Signin :** le composant responsable de connexion du client a son compte base sur une form.

- le code utilise : une portion de code qui nous permet de vérifier l’existence des informations rentrées sur notre base de données.

- le schéma suivi :



* **Theme :** Affiche la liste des thèmes et initialise l'état correspondant au thème sélectionné.

- les fonctions utilisées : **selectionHandeler** qui base sur la fonction **setTheme** qui nous permet d’initialiser le theme.

- le schéma suivi :

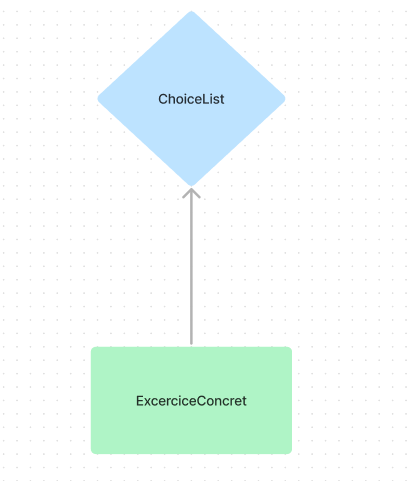
# 

# **Molécule**

# Plusieurs atomes peuvent former une molécule (par exemple un <IconButton />). Les Molécules sont également assez simples et seraient probablement sans état, mais peuvent Contenir un état de bascule simple ou similaire.

# **Liste des Molécules :**

* **ChoiceList :** un composant de catégorie molécule qui permet d’afficher une liste des chois et/ou des réponses.



* **CompetenceList :** nous permet de récupère et affiche la liste des compétences par thème.

- liste des fonctions : **setCompetences** Modifie la liste des compétences.

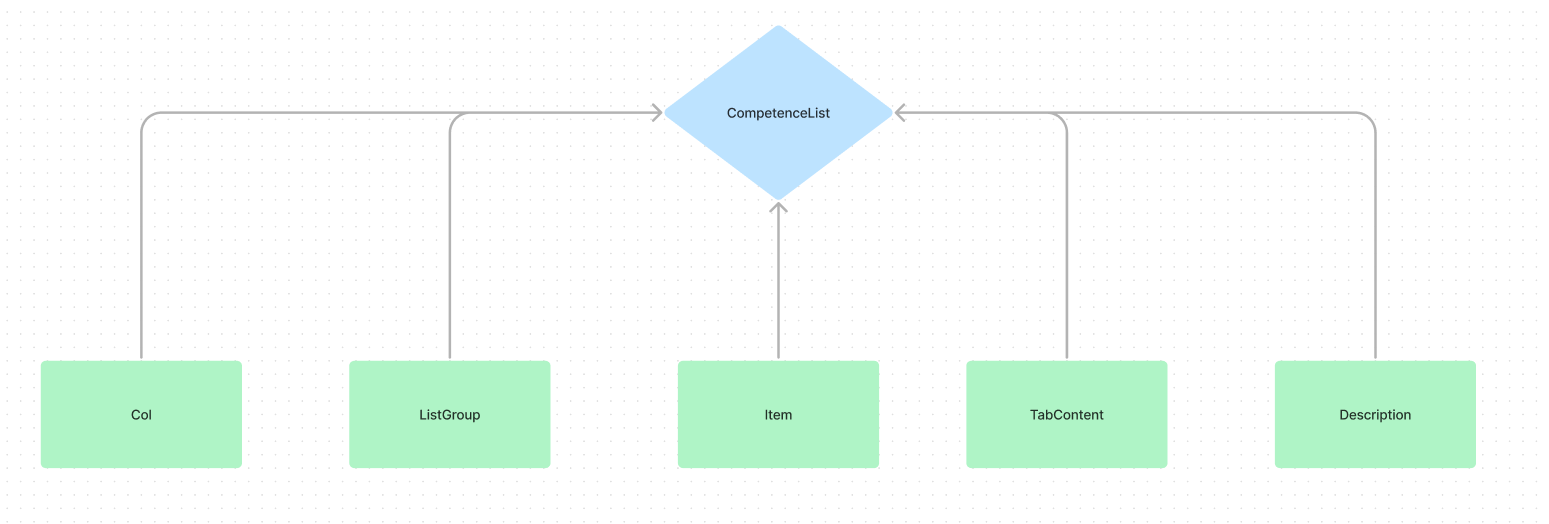
**setTheme** définit la valeur du thème.

**setActiveRoute** permet de changer la référence vers la page courante.

**GetThemecompetence** récupère la liste des Compétences par thème.

**GetAllCompetence** récupère la liste de tous les thèmes.

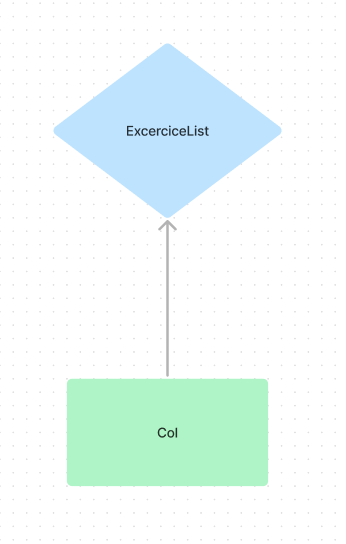
- le schéma suivi :



* **ExcerciceList :** nous permet de récupère et affiche la liste des Exercices.

- liste des fonctions : **setActiveRoute** permet de changer la référence vers la page courante.

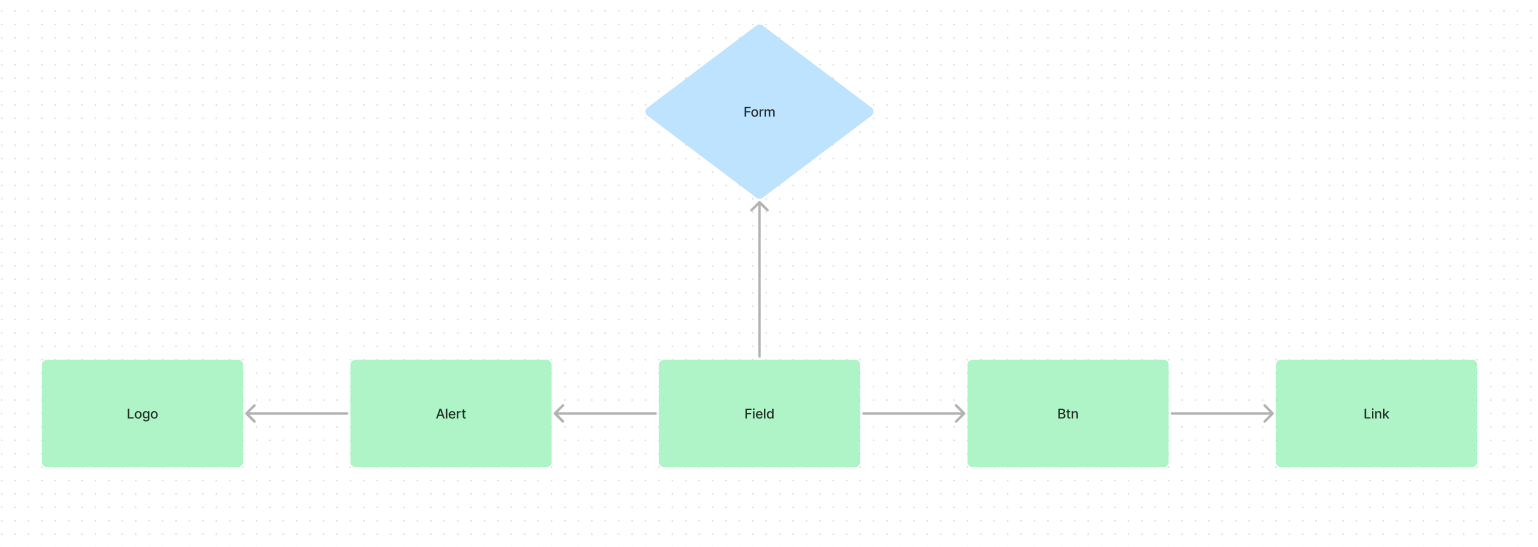
- le schéma suivi :



* **Form :** une molécule représente la forme utilise pour la connexion. Il base sur une alerte, Button, Link, logo, Field comme des atoms.

- le code de fonctionnement de la forme se trouve dans les organisms dont la forme fait partie de leur construction.

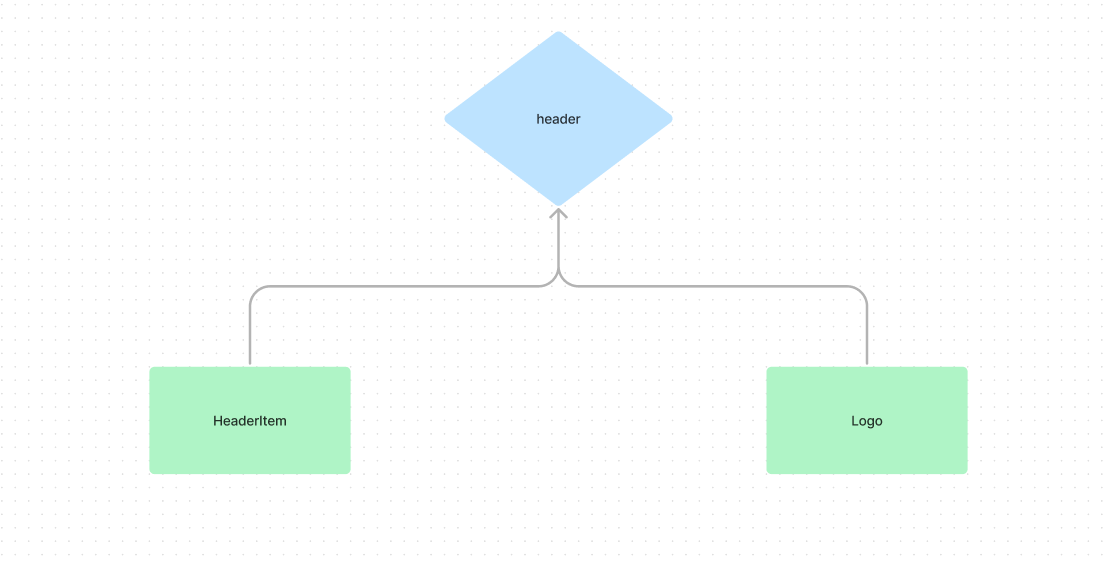
- le schéma suivi :



* **Header :** un composant qui représente une partie de vue avec des buttons qui contient des liens vers différents page (matiere, theme, competence, Excercice) construisent a partir d’un logo et une liste d’item.

- les fonctions utilisées : **setMenuItem** qui permet de modifier la liste des liens vers différents pages.

- le schéma suivi :



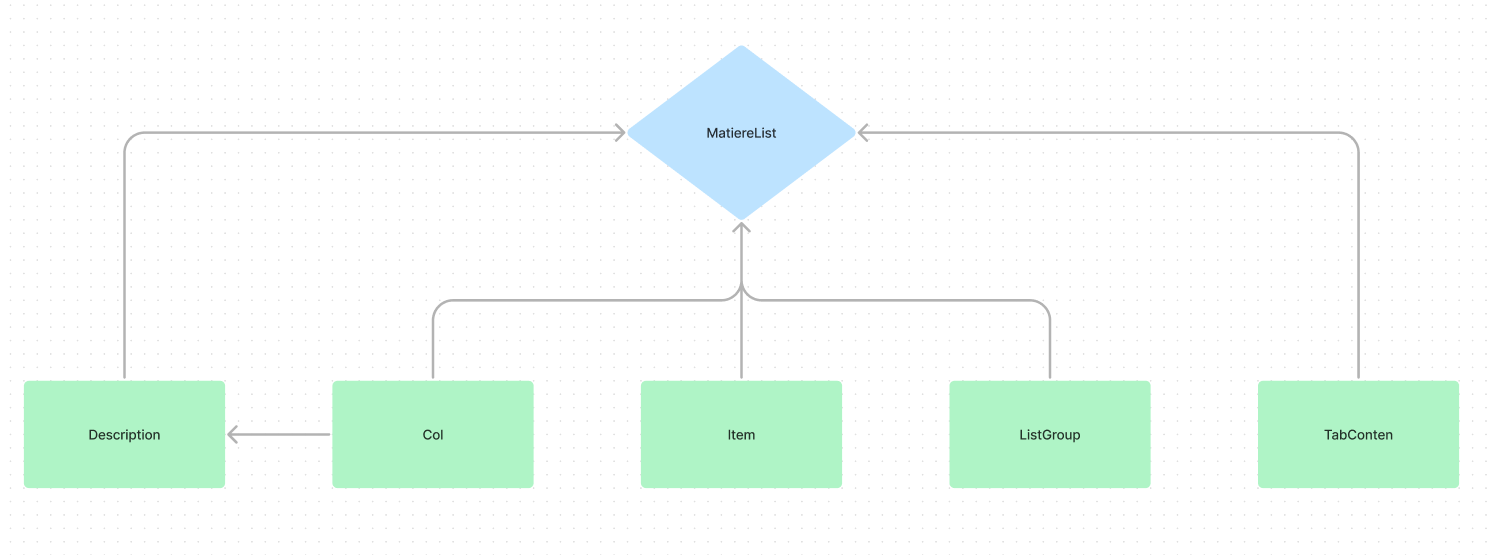
* **MatiereList :** un composant de catégorie molécule qui permet d’afficher la liste des matières et leurs descriptions.

- les fonctions utilisées : **setMatiere** qui initialise la liste des matières.

**GetAllMatiere** récupère la liste de toutes les matières et les charges dans le state.

- une portion de code qui modifier la référence vers la page courante.

- le schéma suivi :



* **Navbar :** un composant qui représente partie d’une vue ou se trouve les deux Button Next et forward.

- les fonctions utilisées : **setActiveRoute** permet de changer la référence vers la page courante.

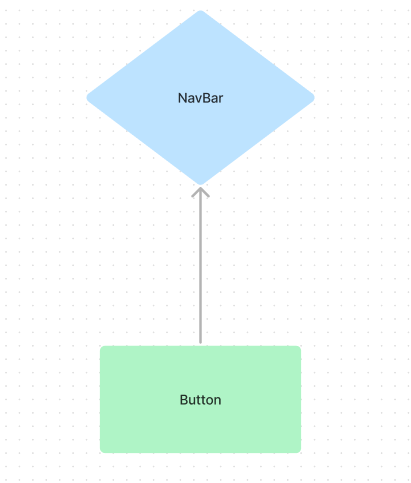
**setBtn** permet de définir si c'est un bouton de réinitialisation ou de retour.

**setBtnText** Change le texte du bouton de droite

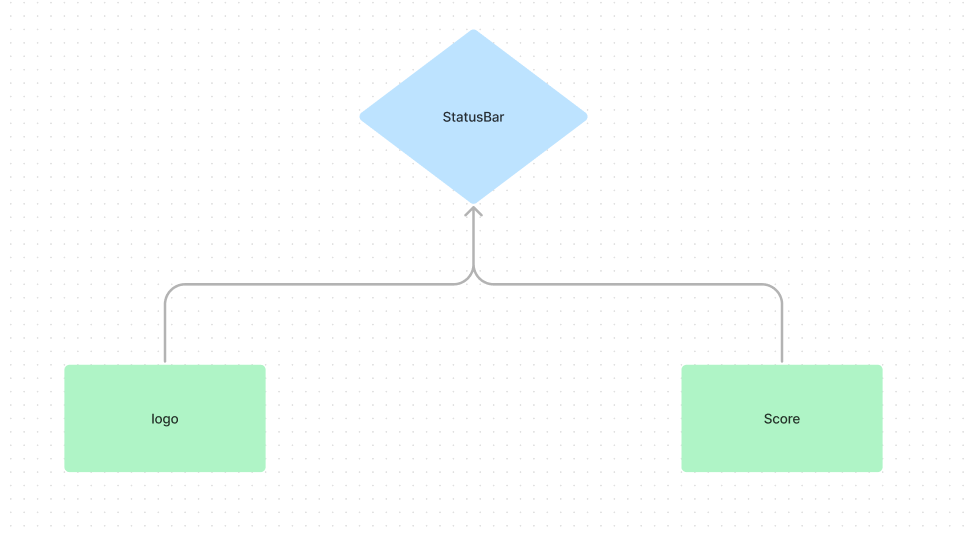
**setResetBtn** permet d'initialiser le bouton de réinitialisation

**Next** cette fonction est exécutée une fois que le bouton "valider" est cliqué, elle évalue les réponses de la question, si les réponses sont correctes, on passe à la prochaine question.

- le schéma suivi :



* **StatusBar :** le composant qui nous permet d’afficher le score basant sur les atoms logo et score.



* **Step :** permet la Définition d'une question de l'exercice courant, construit a partir d’une choiceList, snackbar, alert, navbar.

- les fonctions utilisees : **setPairAnswers** modifie la valeur du Key.

**setKey** modifie la valeur du Key.

**setChoices** initialise la liste des choix d'une question.

**setReponses** initialise la liste des réponses d'une question.

**setCorrection** modifie la valeur de cette variable.

**setOpen** modifie la valeur de cette variable.

**setNoter** modifie la valeur de cette variable.

**setConsigne** modifie la valeur du Key

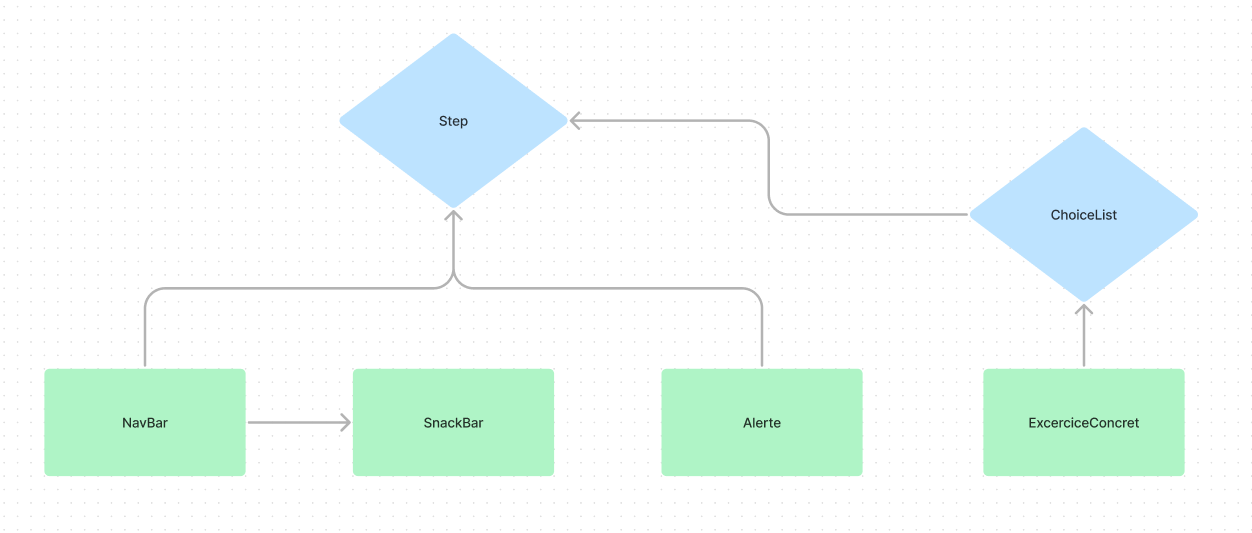
**setLibelle** modifie le libelle de la question.

**setVides** initialise la liste des réponses d'une question.

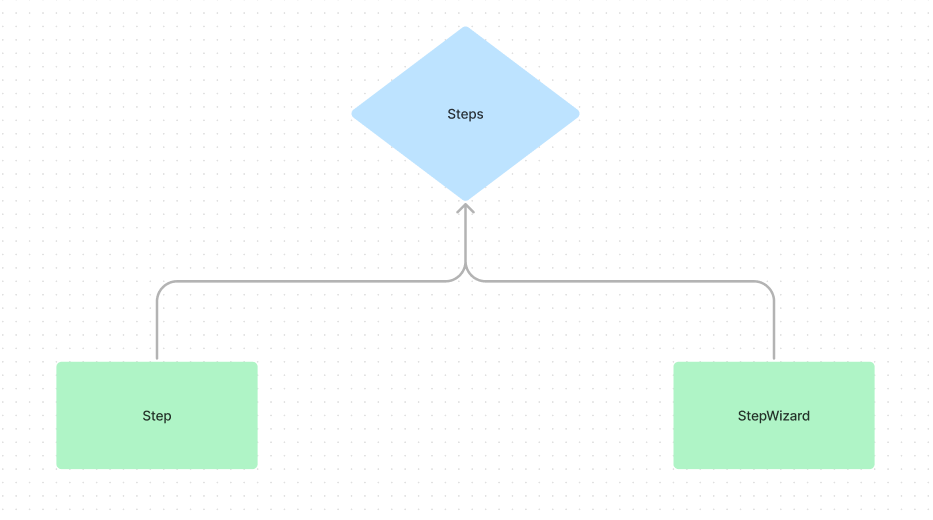
**setDone** initialise la liste des réponses d'une question.

**setStyle** initialise la liste des réponses d'une question.

- le schéma suivi :



* **Steps :** le composant qui Permet d'afficher les questions étape par étape a partir du composant précèdent “step”.



* **ThemeList :**  la molécule qui affiche la liste des thèmes récupères pour la matiere sélectionnée en utilisant ListGroup, Item et description.

- les fonctions utilisees : **setThemes** initialise la liste des thèmes.

**setMatiere** modifie la matière sélectionnée.

**setActiveRoute** permet de changer la référence vers la page courante.

- le schéma suivi :

# 

# **Atom**

# Le plus petit composant possible sera l’Atome (par exemple un <Icon />, <Paragraphe /> ou un <Button />). La méthodologie de conception atomique détermine un atome comme des composants qui ne peuvent pas être décomposés davantage et qui ont encore un sens. Il serait donc logique de créer une liste en tant qu’atome, bien qu’elle contienne des éléments de liste, car les éléments de liste (<li>) n’ont aucune valeur à moins qu’ils ne soient représentés dans <ul> ou <ol>. Les atomes sont par définition super simples. Dans React, ceux-ci seraient très certainement apatrides.

# **Liste des Atomes :**

* **Alerte :** un composant graphique qui “une fenêtre” à afficher en cas d’erreur.
* **Btn :** le composant graphique Button est utilisé pour soumettre notre form.
* **Card :** un composant qui affiche une carte et permet de contrôler son état (tournée, cache).

- les fonctions utilisees : **setCardResultsContext** Ajoute une combinaison dans la liste.

**setIsFlipped** Modifie l'état de la carte.

**setMatch** Modifie la valeur du match (3 états : 'match' , 'no match', '')

**setCardFlipped** Incrémente/Décrémente le nombre de carte tournées

**setFirstChosenCard** initialise une référence vers la première carte sélectionnée.

**setSecondChosenCard** Initialise une référence vers la deuxième carte sélectionnée.

**FlipCard** Initialise la première et la deuxième carte sélectionnées.

**turnCard** Cache une carte et décrémente le nombre de carte tournées.

**HideCard** Vérifie si cela concerne la bonne carte avant de la cacher.

**ShowCard** Rend une carte visible et incrémente le nombre de cartes visibles.

**Evaluate** Compare les 2 cartes choisies et retourne le résultat dans le state

* **Description :** le composant qui donne une Description de la matière, thème, leçon, ou de l'exercice.
* **Exit :** c’est l’élément de sortir, il est représenté par une image.
* **Field :** le composant qui donne l’entrée des informations mentionnée par le client, il se trouve dans les Forms et toute composant qui nécessite une réponse du client.
* **HeaderItem :** le composant responsable de laDéfinition d'une des sections du Header (matières, thèmes, exercices ... ).
* **Item :** un composant qui nous permet de définir un item de la liste (liste des matières, des compétences, des exercices.)
* **Logo :** le logo du site.
* **PrivateRoute :** le composant qui nous dirige vers la route “connexion” si le login est bien vérifié.
* **Proposition :** un composant qui affiche un choix ou une réponse :

Cas dnd : affiche des textes glissables

Cas flip : affiche une carte

Cas Link : dans le cas des choix affiche un text

Cas des réponses affiche des images

Cas select : affiche les choix sous forme de boutons.

* **PublicRoute :** le composant qui nous dirige vers la route “home” si le login est bien vérifié.
* **Score :** le responsable d’afficher le score dépendant en la progression et les récompenses et un scorebox.
* **Scorebox :** la forme d’afficher le score qui dans ce cas une forme linéaire.

# 