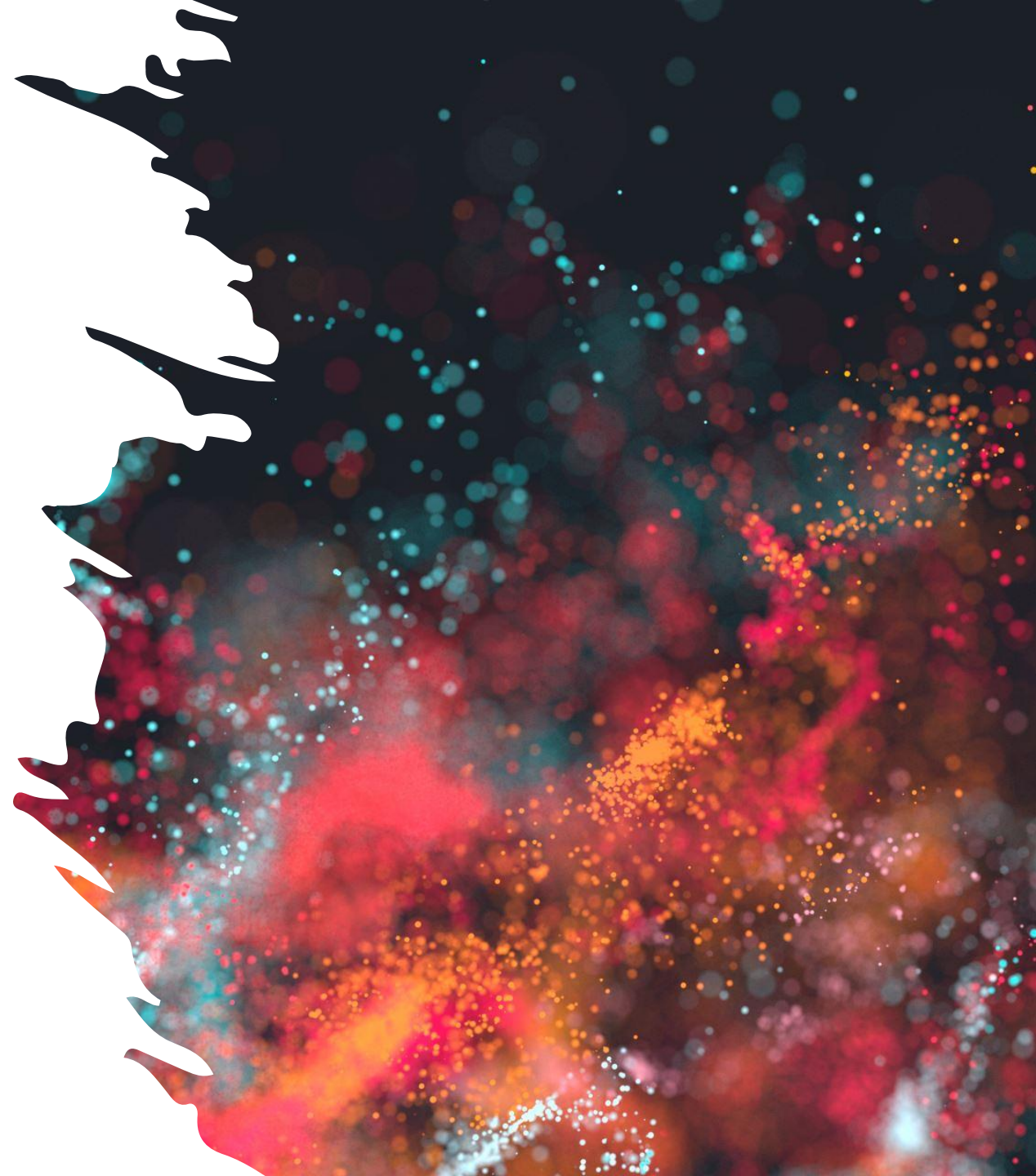


Le projet pas à pas

Commençons avec
Angular

Rémi Pourtier – Benjamin Vella –
Anne Marie Dery





Comment appréhender le démarrage du projet ?

On va procéder par des étapes :

1. Définir les modèles
2. Définir le découpage en composant
3. Comment échanger les données ?

Mock vous avez dit mock ?

Ça vous fait penser à quoi ?

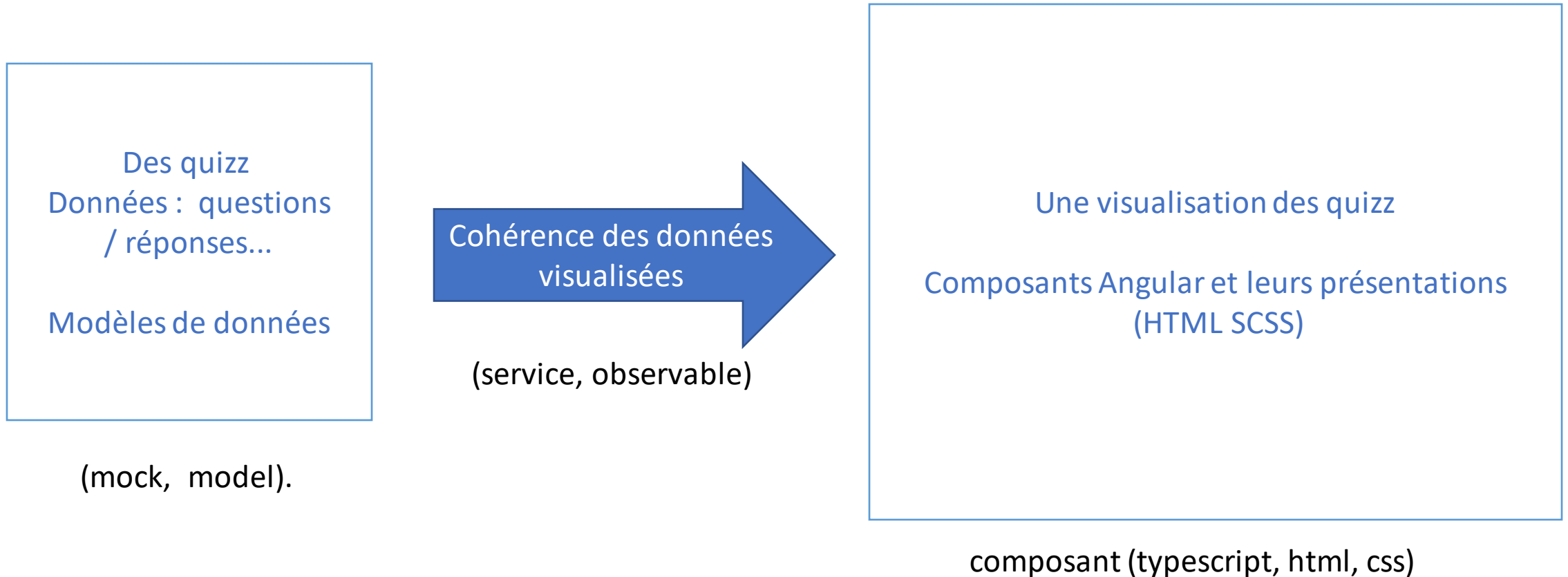
Ça sert à quoi ?

Un **mock** veut dire "imitation" en anglais. Ce concept désigne des fonctions qui ont comme objectif d'imiter le comportement d'autres objets. De manière générale, nous utilisons des **mocks** pour imiter le comportement de librairies ou de modules **que** nous ne souhaitons pas tester.



Architecture simplifiée orientée FRONT

Visualisation des quiz et mock des données



Quelles sont les données que l'on veut manipuler ?

QUIZ : Des questions et pour chaque question des réponses possibles

Des utilisateurs : Des seniors et des soignants

- **Des troubles** : Le trouble et ses caractéristiques



Encore plus près du code : partie Données

quiz-list.mock.ts

```
1 import { Quiz } from '../models/quiz.model';
2 import { Question } from '../models/question.model';
3
4 export const QUESTION_ACTOR: Question = {
5   id: '1',
6   label: 'Jean Gabin a joué dans...',
7   answers: [
8     {
9       value: 'Les tuches II',
10      isCorrect: false,
11    },
12    {
13      value: 'La grande illusion',
14      isCorrect: true,
15    }
16  ]
17 };
18
19 export const QUIZ_LIST: Quiz[] = [
20   {
21     id: '1',
22     name: 'Les Acteurs', // What's happening if I change this value..?
23     theme: 'Actor',
24     questions: [],
25   },
26   {
27     id: '2',
28     name: 'Les technos WEB',
29     questions: [],
30   }
31 ];
```

8 lines (7 sloc) | 153 Bytes

```
1 import { Question } from '../question.model';
2
3 export interface Quiz {
4   id: string;
5   name: string;
6   theme?: string;
7   questions: Question[];
8 }
```

11 lines (10 sloc) | 179 Bytes

```
1 export interface Answer {
2   type?: string;
3   value: string;
4   isCorrect: boolean;
5 }
6
7 export interface Question {
8   id: string;
9   label: string;
10  answers: Answer[];
11 }
```



TD : définir les données

- MODELES :
 - Comprendre les modèles de la correction ([ps6-correction-td1-td2-v2/front-end/src/models/](#))
 - Ecrire les modèles utiles pour votre maquette à placer dans le folder **models** de votre projet
- MOCK
 - Comprendre le mocks de la correction ([ps6-correction-td1-td2-v2/front-end/src/mocks/quiz-list.mock.ts](#))
 - Ecrire vos mocks dans le répertoire **mock**

Que veut-on voir côté client ?

Pour les séniors

- Sélectionner un quiz pour pouvoir jouer

Pour les soignants

- Voir les quiz et pouvoir les créer
- (les modifier,..)
- Voir les séniors pour leur associer des troubles et des quiz en fonction des troubles



Illustration by Freepik

Encore plus près du code : composant User

```
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';
import { User } from '../../models/user.model';
```

```
@Component({
  selector: 'app-user',
  templateUrl: './user.component.html',
  styleUrls: ['./user.component.scss']
})
export class UserComponent implements OnInit {

  @Input()
  user: User;

  @Output()
  deleteUser: EventEmitter<User> = new EventEmitter<User>();

  constructor() { }

  ngOnInit(): void {
  }

  delete() {
    this.deleteUser.emit(this.user);
  }
}
```

```
<div class="card">
  <h2>
    <i class="fas fa-user"></i>
    {{user.firstName}} {{user.lastName}}
  </h2>
  <button class="button-card" (click)="delete()">Delete</button>
</div>
```

```
@import '../../styles/card.scss';
```

Encore plus près du code : composant Liste de User

```
import { Component, OnInit } from '@angular/core';
import { User } from '../../../models/user.model';
import { UserService } from '../../../services/user.service';

@Component({
  selector: 'app-user-list',
  templateUrl: './user-list.component.html',
  styleUrls: ['./user-list.component.scss']
})
export class UserListComponent implements OnInit {

  public userList: User[] = [];

  constructor(private userService: UserService) {
    this.userService.users$.subscribe((users: User[]) => {
      this.userList = users;
    });
  }

  ngOnInit(): void {
  }

  deleteUser(user: User): void {
    this.userService.deleteUser(user);
  }
}
```

```
<app-user-form></app-user-form>
<div class="user-list">
  <div class="user" *ngFor="let user of userList">
    <app-user [user]="user" (deleteUser)="deleteUser($event)"></app-user>
  </div>
</div>
```

```
.user-list {
  display: flex;
}
```

Et pour les quiz
sauriez-vous faire ?



Encore plus près du code : composant quiz

```
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';
import { Quiz } from '../../../models/quiz.model';
```

```
@Component({
  selector: 'app-quiz',
  templateUrl: './quiz.component.html',
  styleUrls: ['./quiz.component.scss']
})
export class QuizComponent implements OnInit {
```

```
  @Input()
  quiz: Quiz;
```

```
  @Output()
  quizSelected: EventEmitter<boolean> = new EventEmitter<boolean>();
```

```
  @Output()
  editQuiz: EventEmitter<Quiz> = new EventEmitter<Quiz>();
```

```
  @Output()
  deleteQuiz: EventEmitter<Quiz> = new EventEmitter<Quiz>();
```

```
  constructor() {
  }
```

```
  ngOnInit(): void {
  }
```

```
  selectQuiz(): void {
    this.quizSelected.emit(true);
  }
```

```
  edit(): void {
    this.editQuiz.emit(this.quiz);
  }
```

```
  delete(): void {
    this.deleteQuiz.emit(this.quiz);
  }
```

```
}
```

```
<div class="card">
  <h2>
    <i class="fab fa-leanpub"></i>
    {{quiz.name}}
  </h2>
  <button class="button-card" (click)="selectQuiz()">Select</button>
  <button class="button-card" (click)="delete()">Delete</button>
  <button class="button-card edit" routerLink="/edit-quiz/{{quiz.id}}">Edit</button>
</div>
```

5 lines (4 sloc) | 71 Bytes

```
1 @import '../../../styles/card.scss';
2
3 .edit {
4     margin-left: 0.3em;
5 }
```

Encore plus près du code : composants liste de quizzes

```
1 import { Component, OnInit } from '@angular/core';
2 import { Router } from '@angular/router';
3 import { QuizService } from '../../services/quiz.service';
4 import { Quiz } from '../../models/quiz.model';
5
6 @Component({
7   selector: 'app-quiz-list',
8   templateUrl: './quiz-list.component.html',
9   styleUrls: ['./quiz-list.component.scss']
10 })
11 export class QuizListComponent implements OnInit {
12
13   public quizList: Quiz[] = [];
14
15   constructor(private router: Router, public quizService: QuizService) {
16     this.quizService.quizzes$.subscribe((quizzes: Quiz[]) => {
17       this.quizList = quizzes;
18     });
19   }
20
21   ngOnInit(): void {
22   }
23
24   quizSelected(selected: boolean): void {
25     console.log('event received from child:', selected);
26   }
27
28   editQuiz(quiz: Quiz): void {
29     this.router.navigate(['/edit-quiz/' + quiz.name]);
30   }
31
32   deleteQuiz(quiz: Quiz): void {
33     this.quizService.deleteQuiz(quiz);
34   }
35 }
```

```
1 <app-quiz-form></app-quiz-form>
2 <div class="quiz-list">
3   <div class="quiz" +ngForm="let quiz of quizList">
4     <!--Inputs & Output allow communication between parent & child components.-->
5     <!--More information: https://angular.io/guide/component-interaction-->
6     <app-quiz [quiz]="quiz" (quizSelected)="quizSelected($event)" (deleteQuiz)="deleteQuiz($event)" (editQuiz)="editQuiz($event)"></app-quiz>
7   </div>
8 </div>
```

```
1 .quiz-list {
2   display: flex;
3 }
```

TD : structurer les pages avec vos composants

PARTIE SOIGNANTS

1. Comprendre les codes de la correction pour les composants ([ps6-correction-td1-td2-v2/front-end/src/app/](#))
2. Ecrire vos composants (la structure les fichiers .ts)
3. Ecrire les fichiers HTML et SCSS pour vous rapprocher au plus de votre maquette

PARTIE ACCUEILLIS

Procéder de même pour la partie Jeu



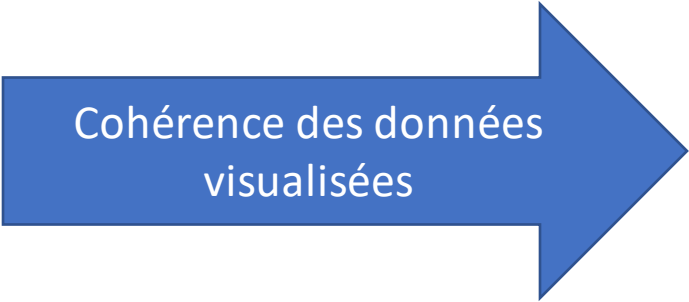
Comment relier les données à leur visualisation ?

- **Gérer la cohérence entre les données stockées et les données montrées**

Une architecture simplifiée : communication données et Front

Un service **permet de faire le lien** **en** important le mock et en servant ensuite d'initialisation au flux d'un observable

Le composant QuizList s'abonne au flux de l'observable et peut ainsi stocker les données transmises par le flux dans une variable **quizList**



Cohérence des données
visualisées

Décomposition en composants ?

Create a new Quiz

Title

Theme

Create

Les grands joueurs

Select

Delete

Edit

Les films des années 80

Select

Delete

Edit

Décomposition en composants


Create a new Quiz

Title

Theme


Create

QuizForm

 **Les grands joueurs**

Select Delete Edit

Quiz

 **Les films des années 80**

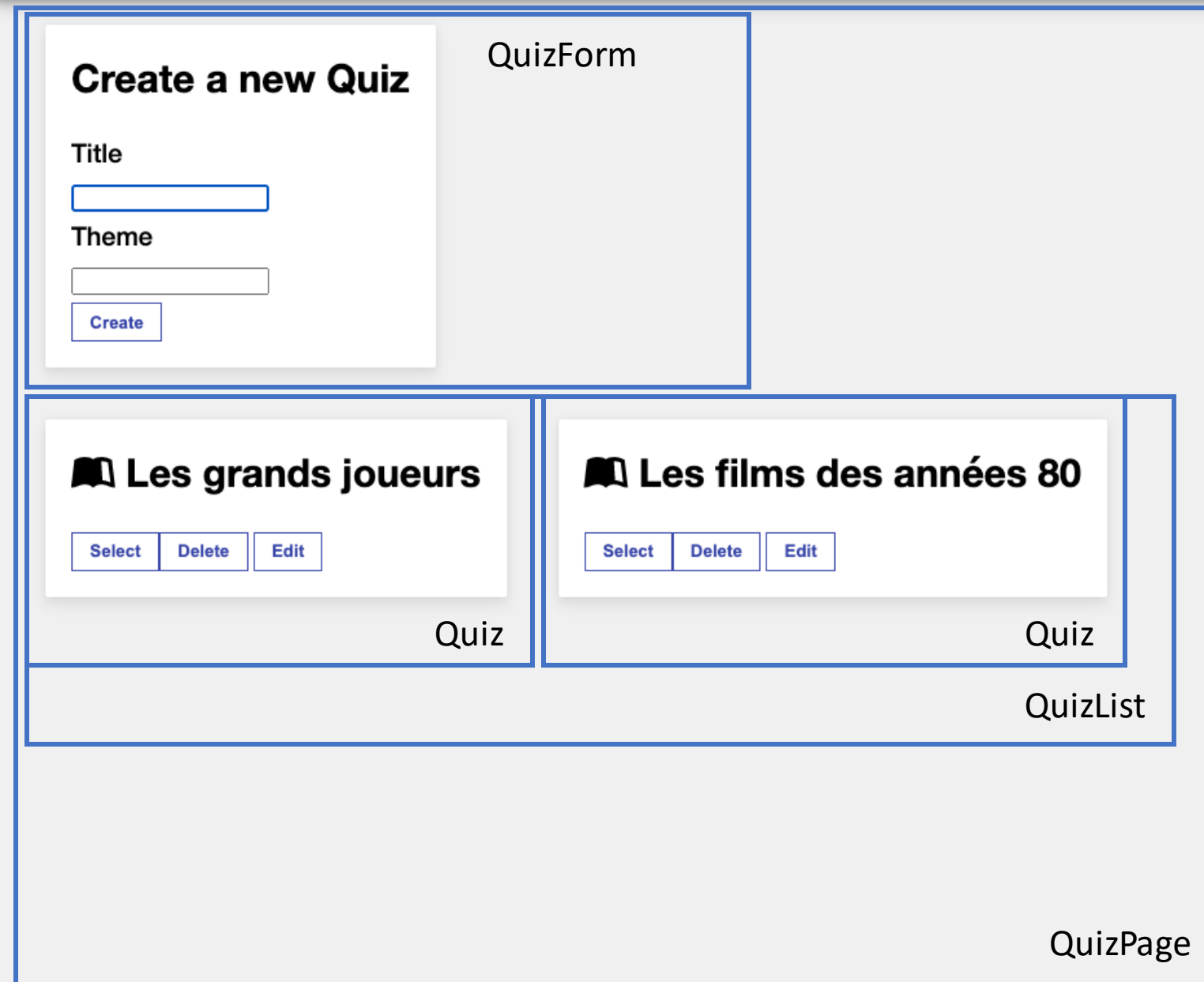
Select Delete Edit

Quiz

QuizList

Décomposition en composants

Approche simplifiée
À voir plus tard



Penser au(x)
modèle(s)

Un modèle ?

Create a new Quiz

QuizForm

Title

Theme

Create

 **Les grands joueurs**

Select

Delete

Edit

Quiz

 **Les films des années 80**

Select

Delete

Edit

Quiz

QuizList

QuizPage

Comment partager /
échanger les données ?

Un service ?
Pour quelle(s)
donnée(s)

?

?

?

?

Create a new Quiz

QuizForm

Title

Theme

Create

Les grands joueurs

Select

Delete

Edit

Quiz

Les films des années 80

Select

Delete

Edit

Quiz

QuizList

QuizPage

Comment partager /
échanger les données ?

Des @Input ?

Create a new Quiz

Title

Theme

Create

QuizForm

@Input ?

Les grands joueurs

SelectDeleteEdit

@Input ?Quiz

Les films des années 80

SelectDeleteEdit

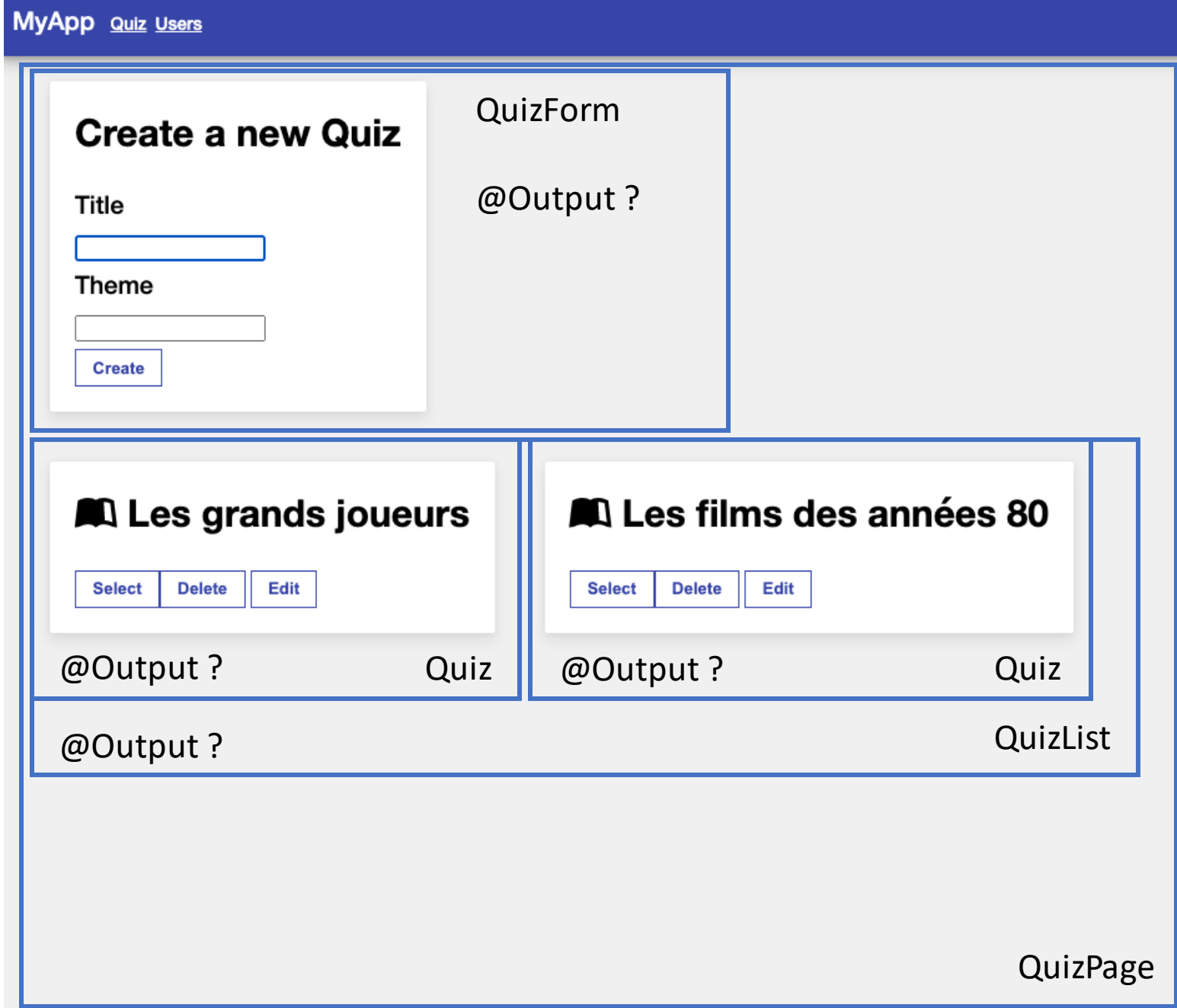
@Input ?Quiz

@Input ?QuizList

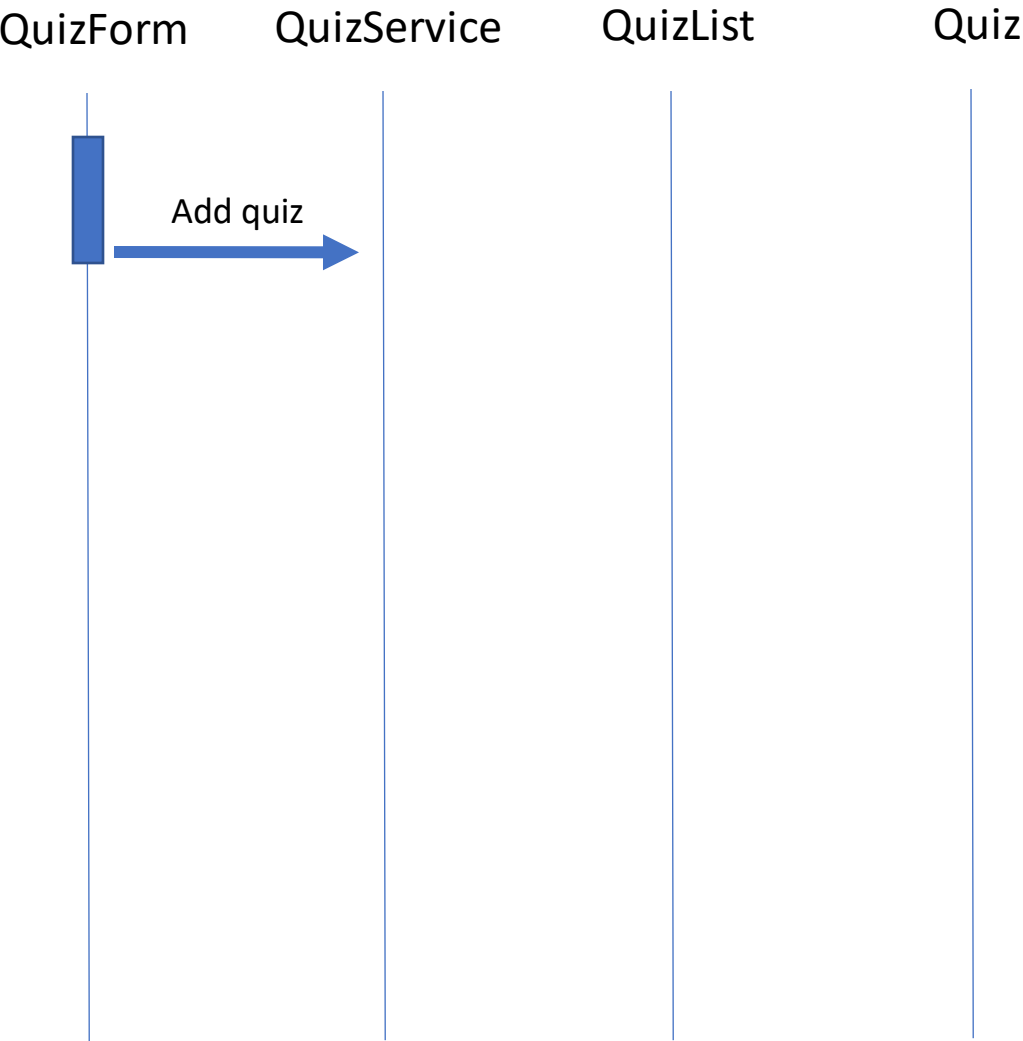
QuizPage

Comment partager /
échanger les données ?

Des @Output ?



Séquence d'ajout et de suppression d'un quiz



Create a new Quiz

QuizForm

Title

Theme

Create

Les grands joueurs

SelectDeleteEdit

Quiz

Les films des a

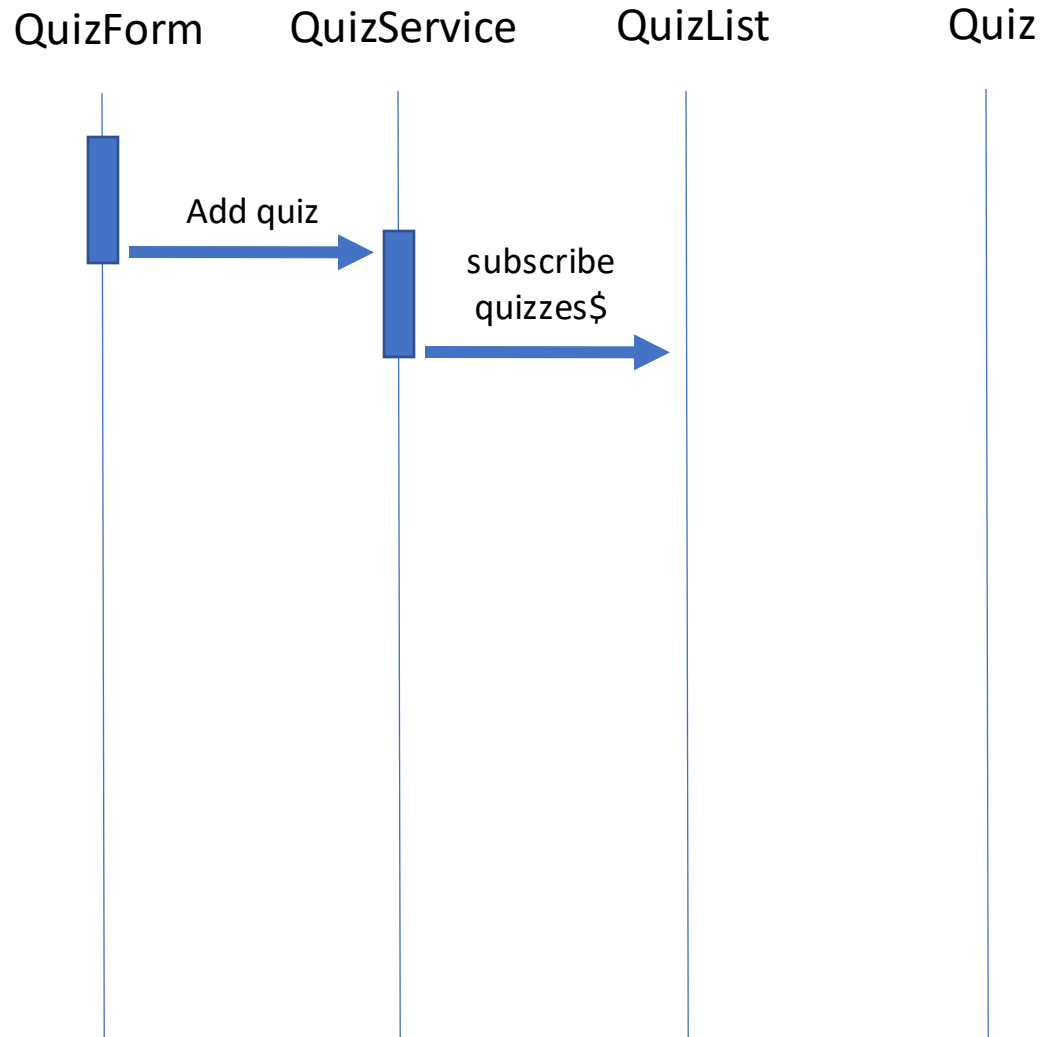
SelectDeleteEdit

Quiz

QuizList

QuizPage

Séquence d'ajout et de suppression d'un quiz



Create a new Quiz

Title

Theme

Create

QuizForm

Les grands joueurs

Select Delete Edit

Quiz

Les films des a

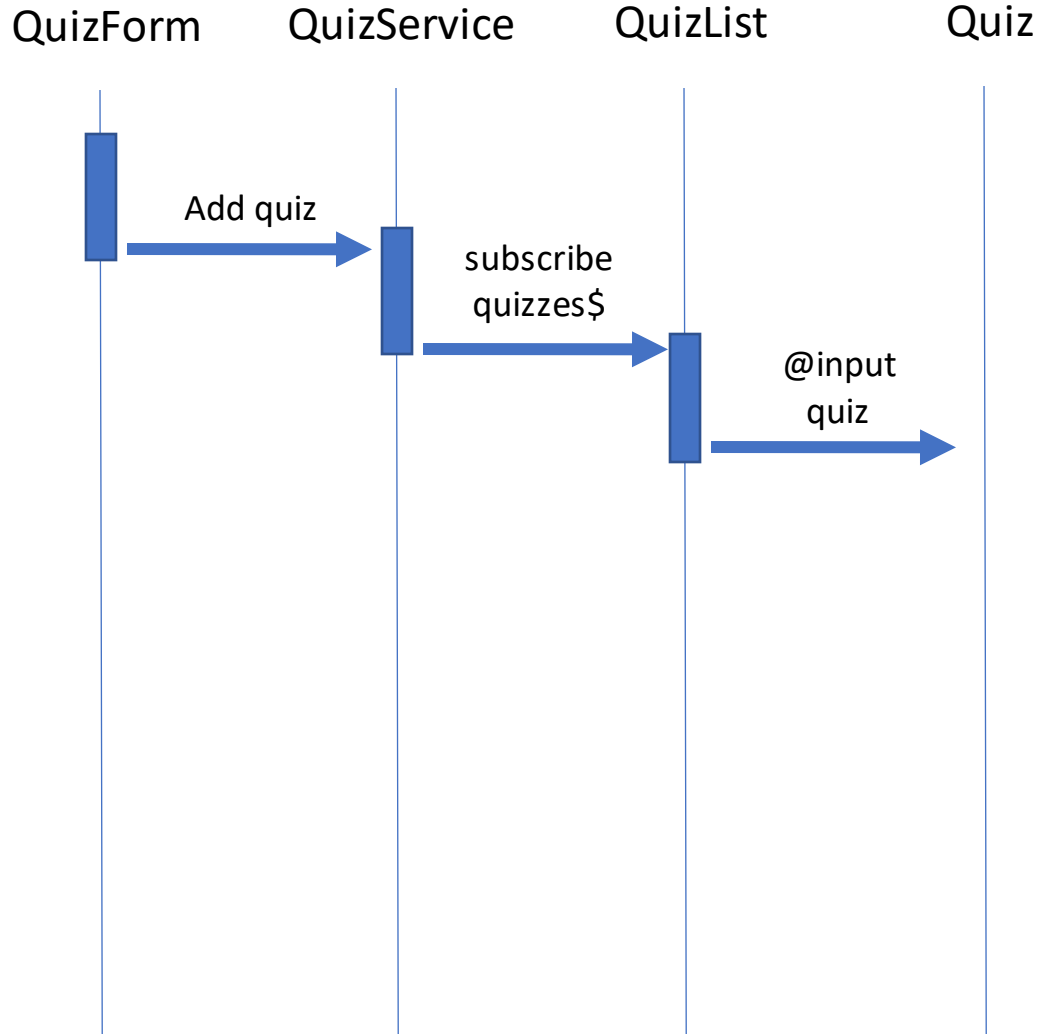
Select Delete Edit

Quiz

QuizList

QuizPage

Séquence d'ajout et de suppression d'un quiz



Create a new Quiz

Title

Theme

Create

QuizForm

Les grands joueurs

Select Delete Edit

Quiz

Les films des a

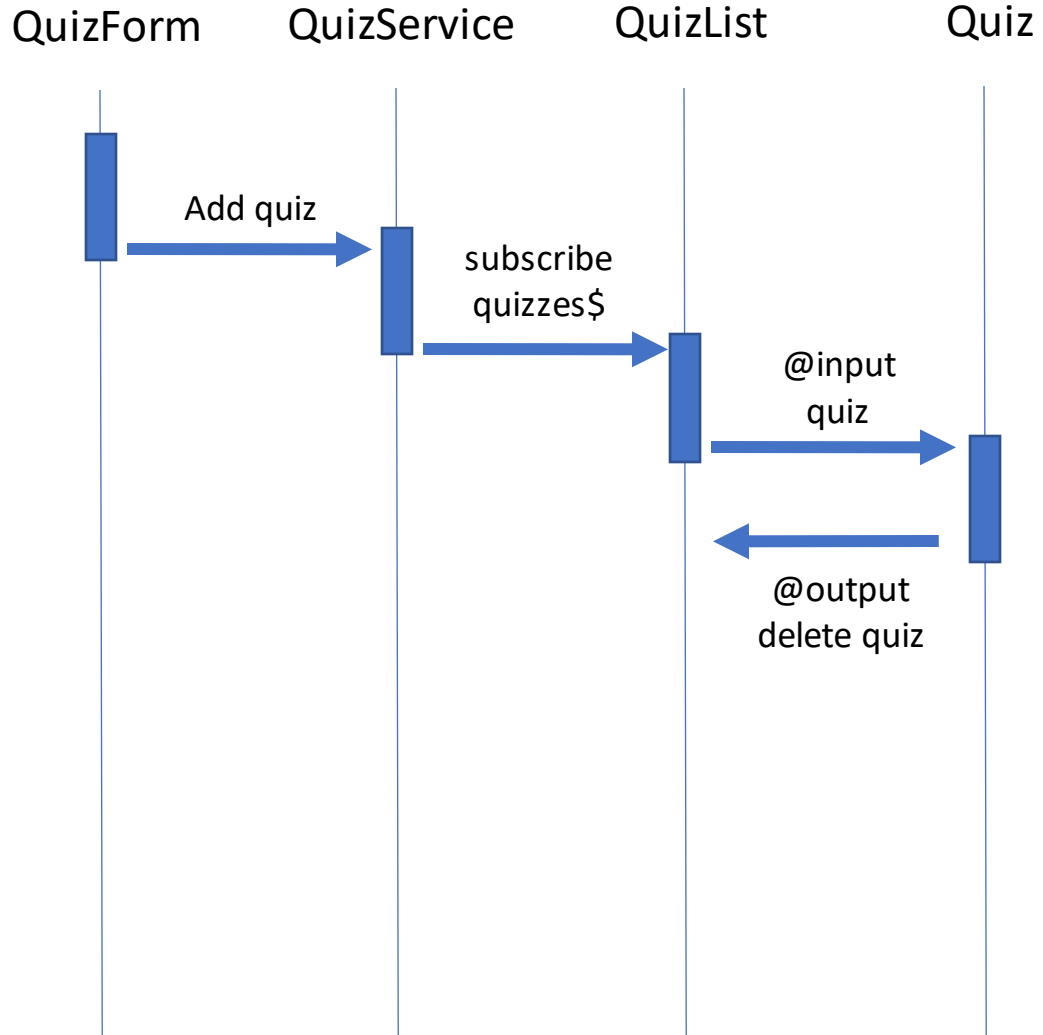
Select Delete Edit

Quiz

QuizList

QuizPage

Séquence d'ajout et de suppression d'un quiz



Create a new Quiz

Title

Theme

Create

QuizForm

Les grands joueurs

Select Delete Edit

Quiz

Les films des a

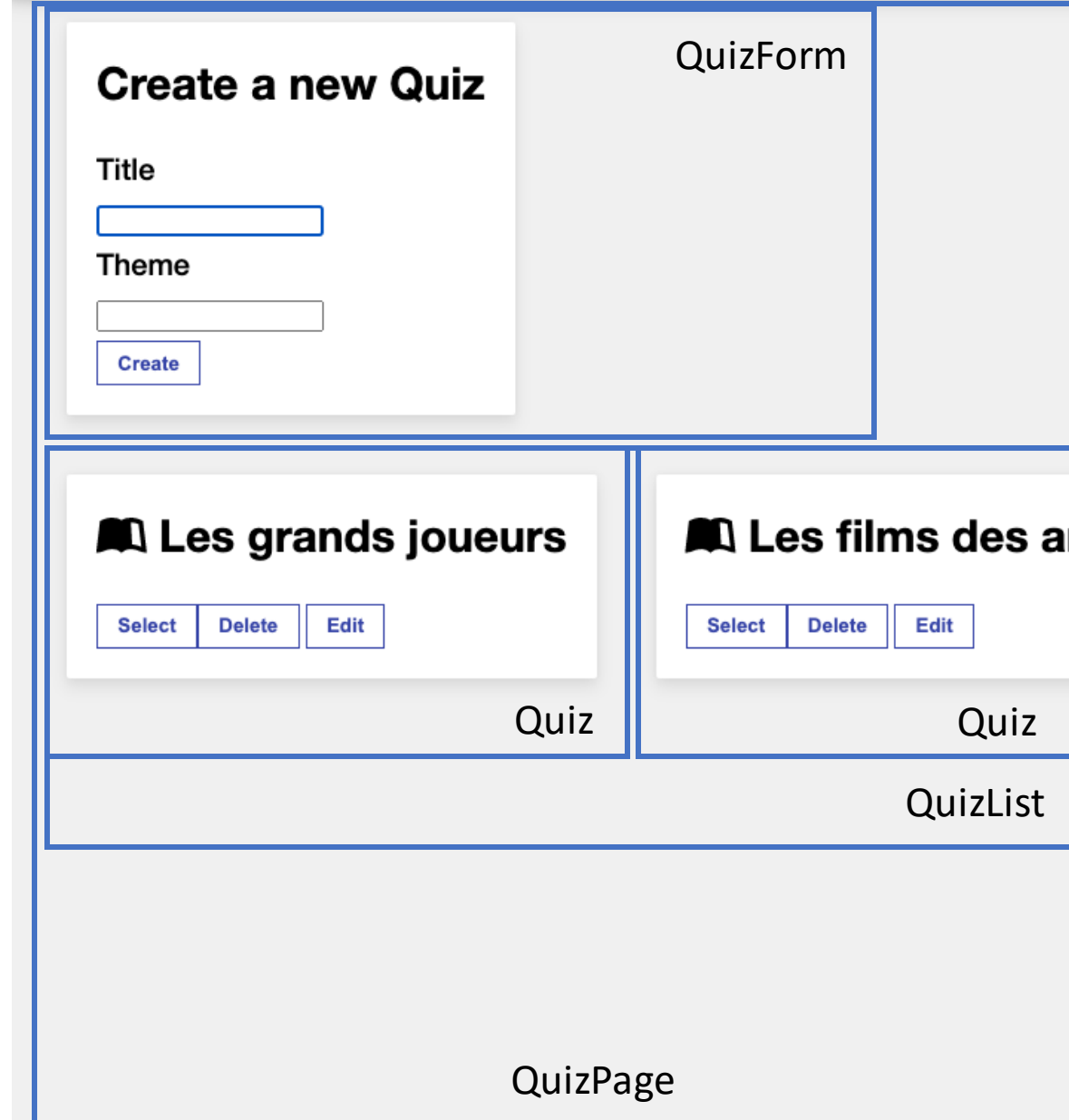
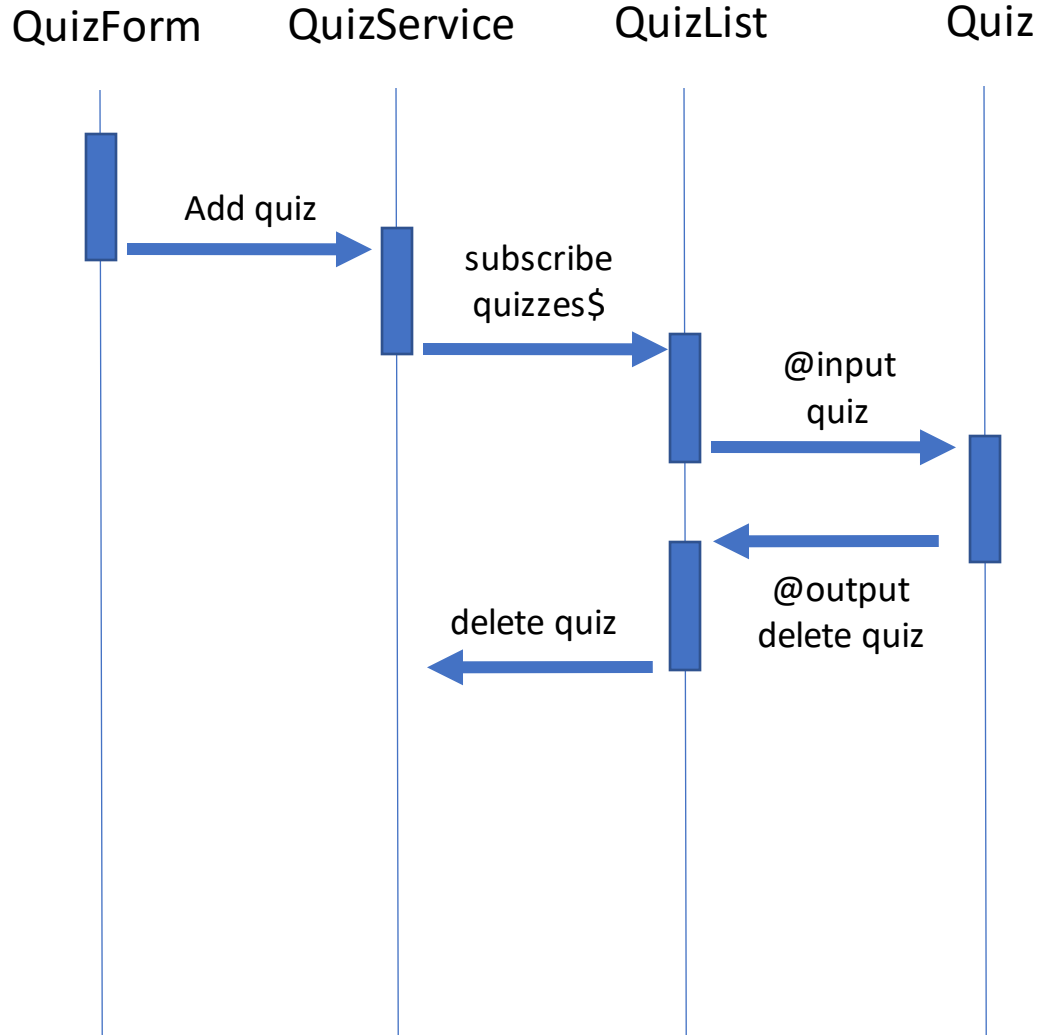
Select Delete Edit

Quiz

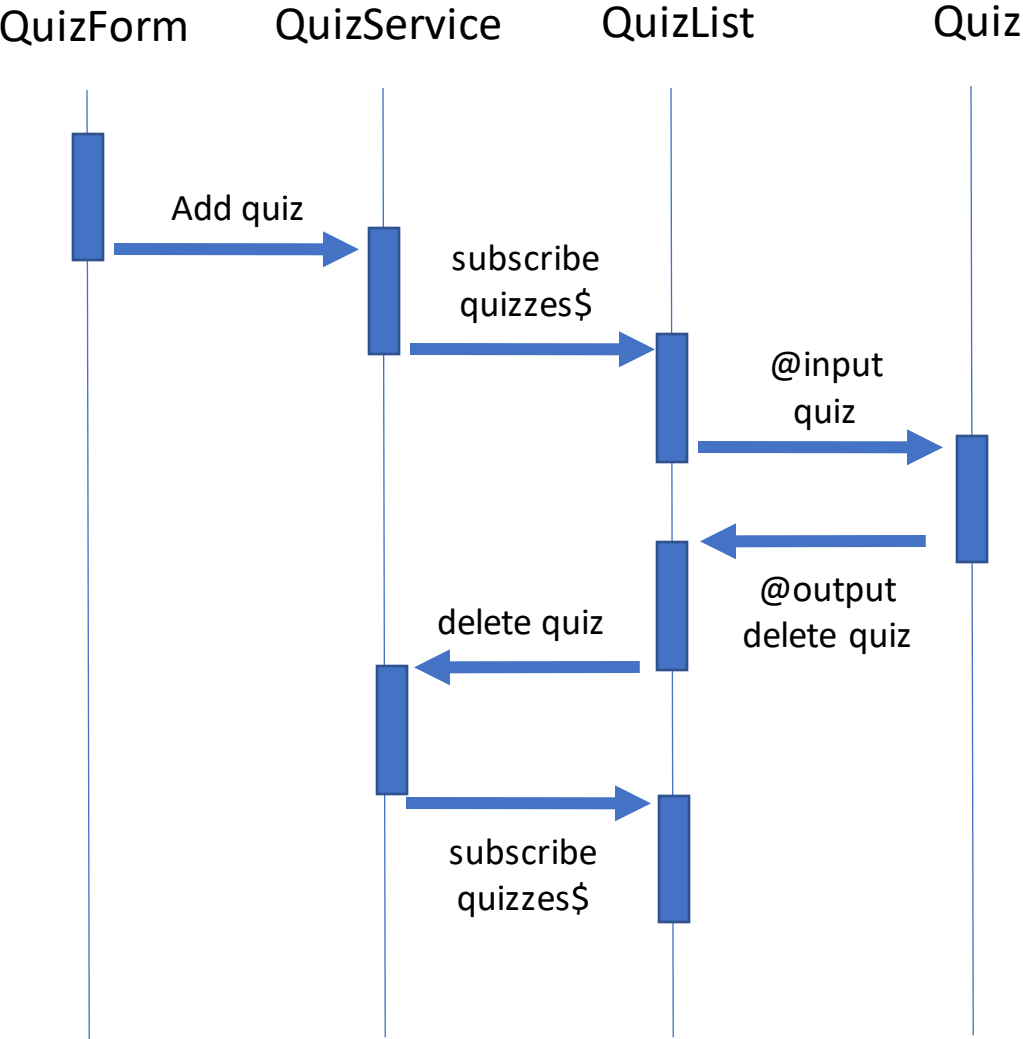
QuizList

QuizPage

Séquence d'ajout et de suppression d'un quiz



Séquence d'ajout et de suppression d'un quiz



Create a new Quiz

Title

Theme

Create

QuizForm

Les grands joueurs

Select Delete Edit

Quiz

Les films des a

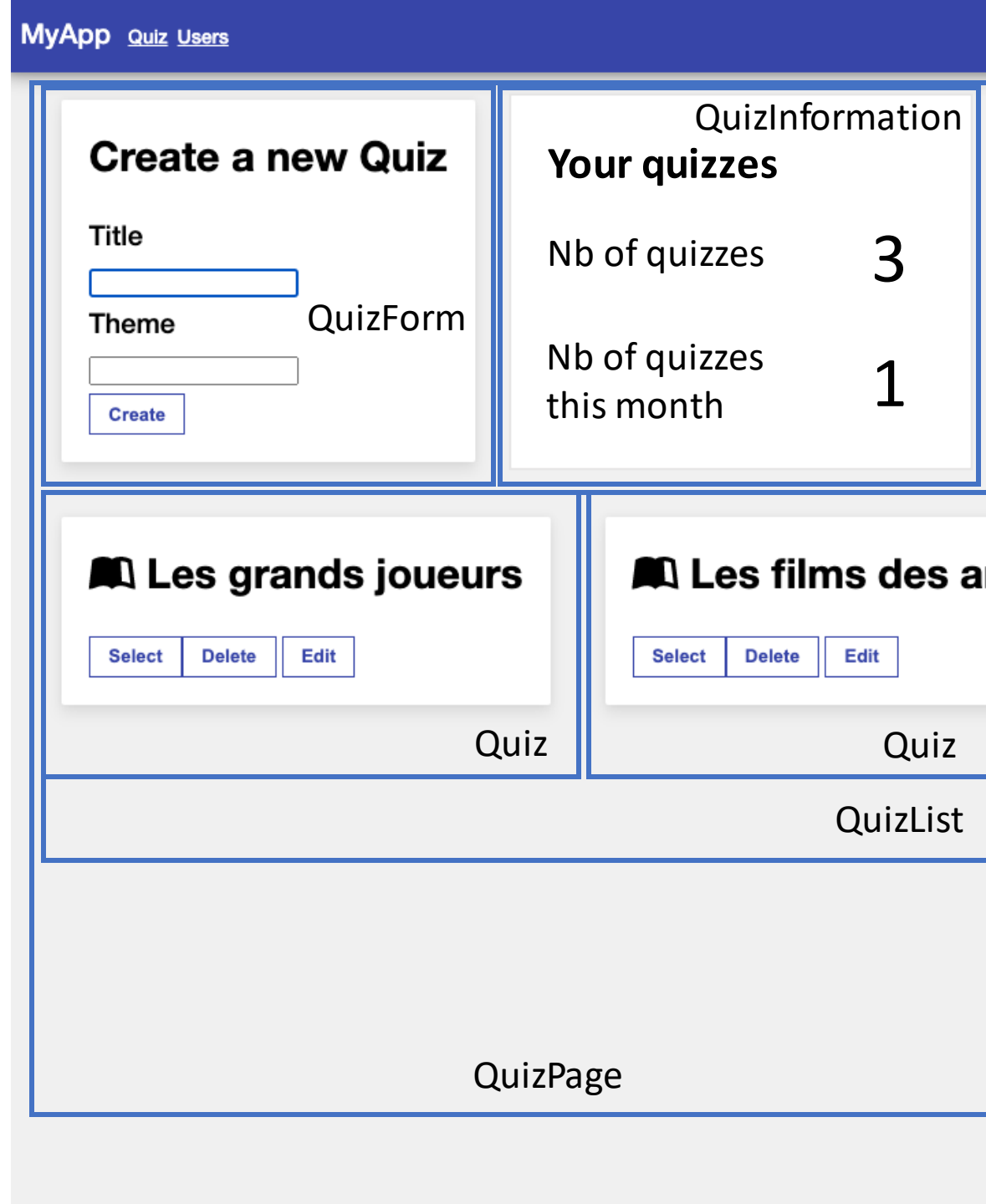
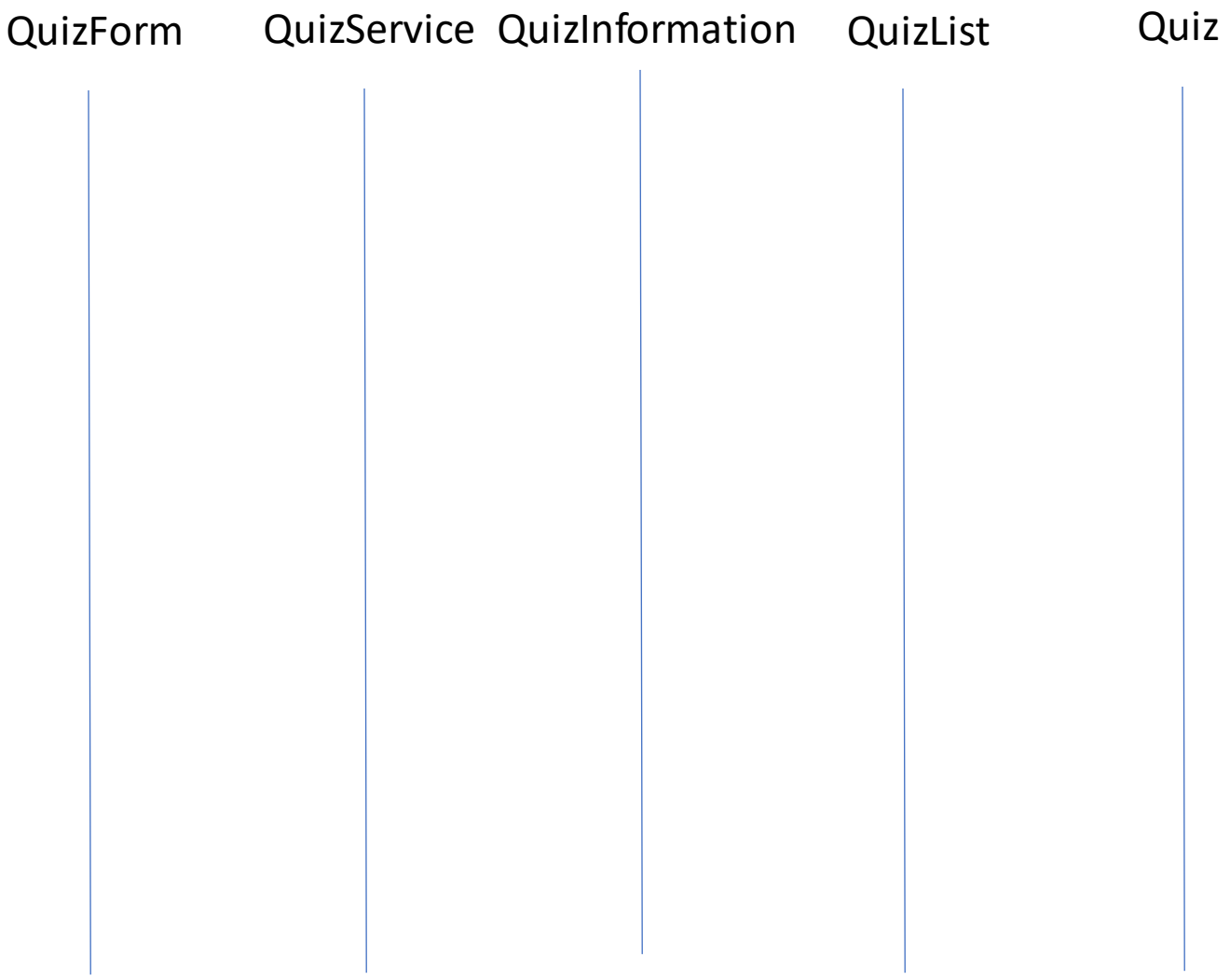
Select Delete Edit

Quiz

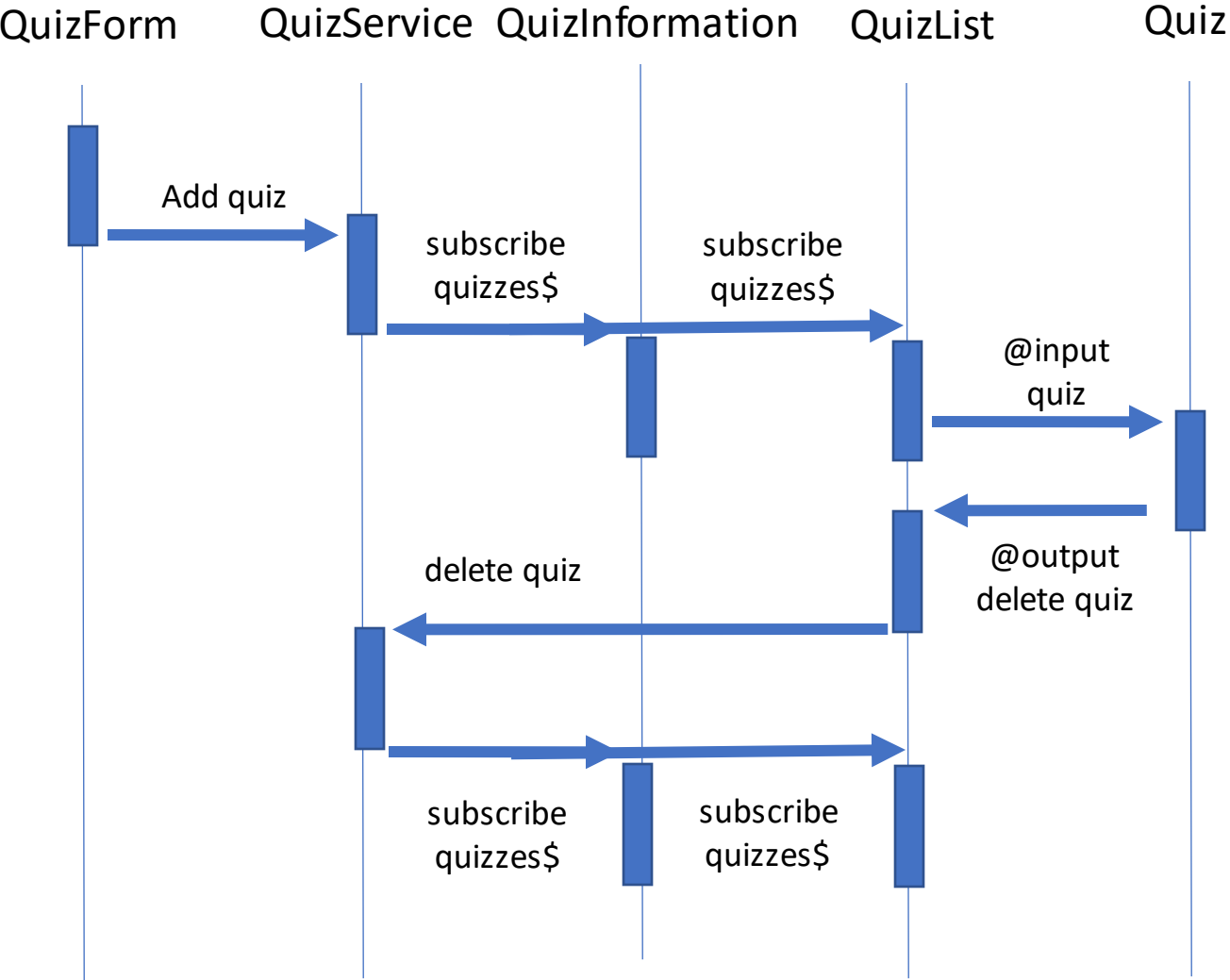
QuizList

QuizPage

Séquence d'ajout et de suppression d'un quiz



Séquence d'ajout et de suppression d'un quiz



Create a new Quiz

Title

Theme

Create

QuizForm

Nb of quizzes

3

Nb of quizzes this month

1

Les grands joueurs

Select Delete Edit

Quiz

Les films des a

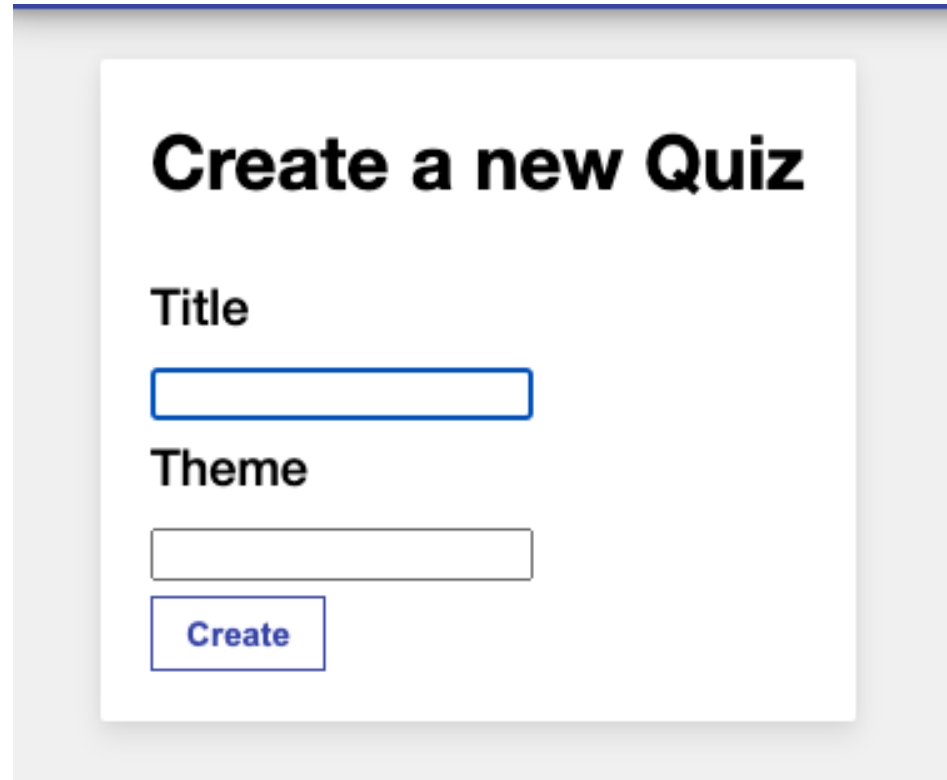
Select Delete Edit

Quiz

QuizList

QuizPage

Les formulaires



Create a new Quiz

Title

Theme

Create

Les formulaires



Classe du composant

```
public quizForm: FormGroup;

constructor(public FormBuilder: FormBuilder, public quizService: QuizService) {
  this.quizForm = this.formBuilder.group({
    name: [''],
    theme: ['']
  });
}

addQuiz(): void {
  const quizToCreate: Quiz = this.quizForm.getRawValue() as Quiz;
  this.quizService.addQuiz(quizToCreate);
}
```

1) Déclaration et initialisation

Les formulaires



Classe du composant

```
public quizForm: FormGroup;

constructor(public FormBuilder: FormBuilder, public quizService: QuizService) {
  this.quizForm = this.formBuilder.group({
    name: [''],
    theme: ['']
  });
}

addQuiz(): void {
  const quizToCreate: Quiz = this.quizForm.getRawValue() as Quiz;
  this.quizService.addQuiz(quizToCreate);
}
```



Vue du composant

```
<form class="quiz-form" [formGroup]="quizForm">
  <label for="title">
    Title <br>
    <input class="input-custom" type="text" id="name" [formControlName]='name'>
  </label>
  <br>
  <label for="theme">
    Theme <br>
    <input class="input-custom" type="text" id="theme" [formControlName]='theme'>
  </label>
  <br>
  <button type="submit" class="button-card" (click)="addQuiz()">Create</button>
</form>
```

- 2) Déclaration de notre attribut FormGroup dans le formulaire du composant

Les formulaires



Classe du composant



Vue du composant

```
public quizForm: FormGroup;

constructor(public FormBuilder: FormBuilder, public QuizService: QuizService) {
  this.quizForm = this.FormBuilder.group({
    name: [''],
    theme: ['']
  });
}

addQuiz(): void {
  const quizToCreate: Quiz = this.quizForm.getRawValue() as Quiz;
  this.QuizService.addQuiz(quizToCreate);
}
```

```
<form class="quiz-form" [formGroup]="quizForm">
  <label for="title">
    Title <br>
    <input class="input-custom" type="text" id="name" [formControlName]='name'>
  </label>
  <br>
  <label for="theme">
    Theme <br>
    <input class="input-custom" type="text" id="theme" [formControlName]='theme'>
  </label>
  <br>
  <button type="submit" class="button-card" (click)="addQuiz()">Create</button>
</form>
```

- 3) Déclaration des champs de notre attribut FormGroup dans les champs du formulaire

Les formulaires



Classe du composant

```
public quizForm: FormGroup;

constructor(public FormBuilder: FormBuilder, public quizService: QuizService) {
  this.quizForm = this.formBuilder.group({
    name: [''],
    theme: ['']
  });
}

addQuiz(): void {
  const quizToCreate: Quiz = this.quizForm.getRawValue() as Quiz;
  this.quizService.addQuiz(quizToCreate);
}
```



Vue du composant

```
<form class="quiz-form" [formGroup]="quizForm">
  <label for="title">
    Title <br>
    <input class="input-custom" type="text" id="name" [formControlName]='name'>
  </label>
  <br>
  <label for="theme">
    Theme <br>
    <input class="input-custom" type="text" id="theme" [formControlName]='theme'>
  </label>
  <br>
  <button type="submit" class="button-card" (click)="addQuiz()">Create</button>
</form>
```

- 4) Définition de la fonction à appeler à la soumission du formulaire

Les formulaires



Classe du composant

```
public quizForm: FormGroup;

constructor(public FormBuilder: FormBuilder, public quizService: QuizService) {
  this.quizForm = this.formBuilder.group({
    name: [''],
    theme: ['']
  });
}

addQuiz(): void {
  const quizToCreate: Quiz = this.quizForm.getRawValue() as Quiz;
  this.quizService.addQuiz(quizToCreate);
}
```



Vue du composant

```
<form class="quiz-form" [formGroup]="quizForm">
  <label for="title">
    Title <br>
    <input class="input-custom" type="text" id="name" [formControlName]='name'>
  </label>
  <br>
  <label for="theme">
    Theme <br>
    <input class="input-custom" type="text" id="theme" [formControlName]='theme'>
  </label>
  <br>
  <button type="submit" class="button-card" (click)="addQuiz()">Create</button>
</form>
</div>
<!-- Uncomment below to see your form structure and how it changes-->
<!-- <br><br>{{quizForm.getRawValue() | json}} -->
```

5) Debug
Décommenter la ligne

Les formulaires

Create a new Quiz

Title

Theme

Create

```
{ "name": "", "theme": "" }
```

5) Debug

Les formulaires



Classe du composant

```
public quizForm: FormGroup;

constructor(public FormBuilder: FormBuilder, public quizService: QuizService) {
  this.quizForm = this.formBuilder.group({
    name: [''],
    theme: ['']
  });
}

addQuiz(): void {
  const quizToCreate: Quiz = this.quizForm.getRawValue() as Quiz;
  this.quizService.addQuiz(quizToCreate);
}
```



Vue du composant

```
<form class="quiz-form" [formGroup]="quizForm">
  <label for="title">
    Title <br>
    <input class="input-custom" type="text" id="name" [formControlName]='name'>
  </label>
  <br>
  <label for="theme">
    Theme <br>
    <input class="input-custom" type="text" id="theme" [formControlName]='theme'>
  </label>
  <br>
  <button type="submit" class="button-card" (click)="addQuiz()">Create</button>
</form>
```

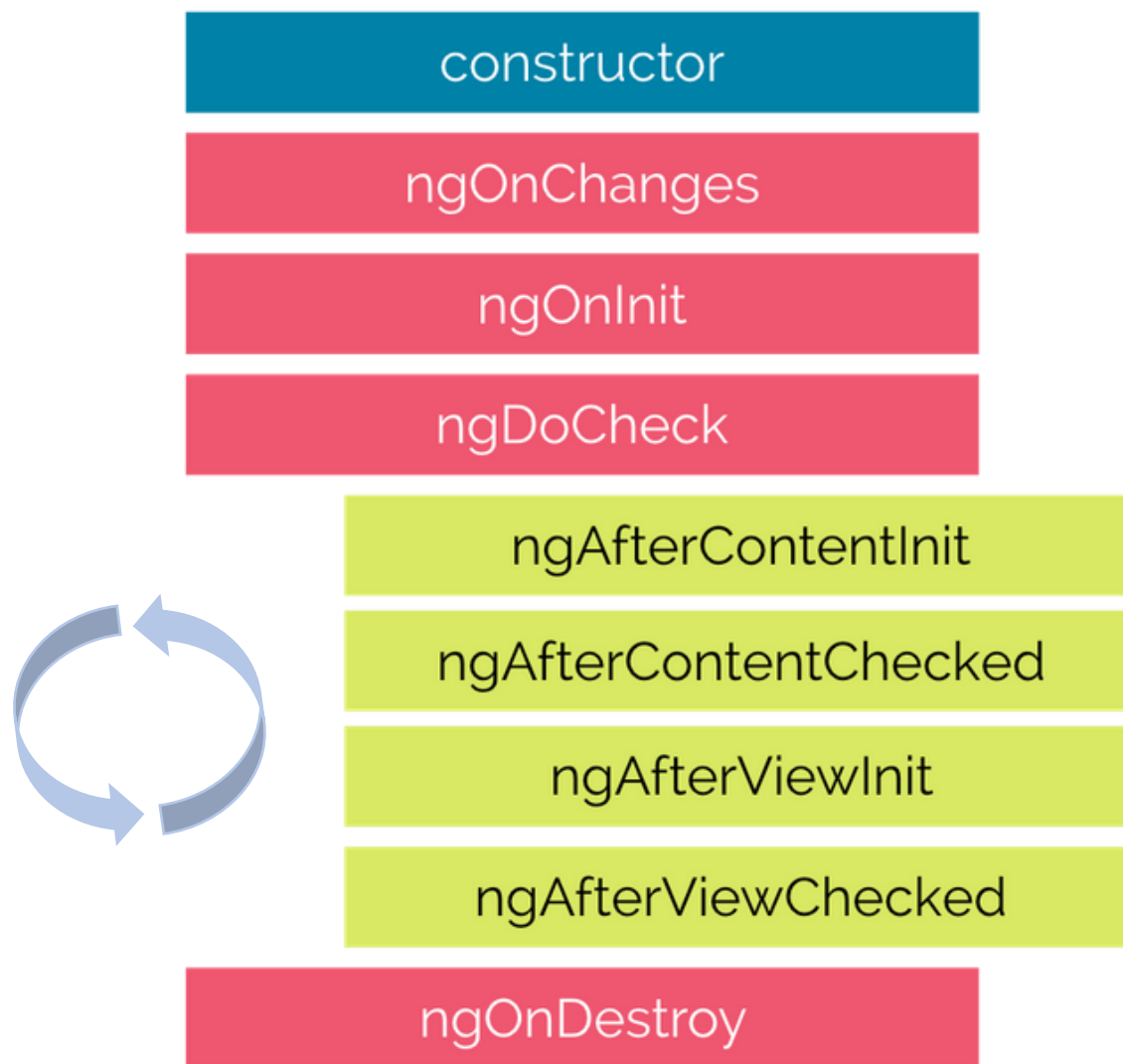
Create a new Quiz

Title

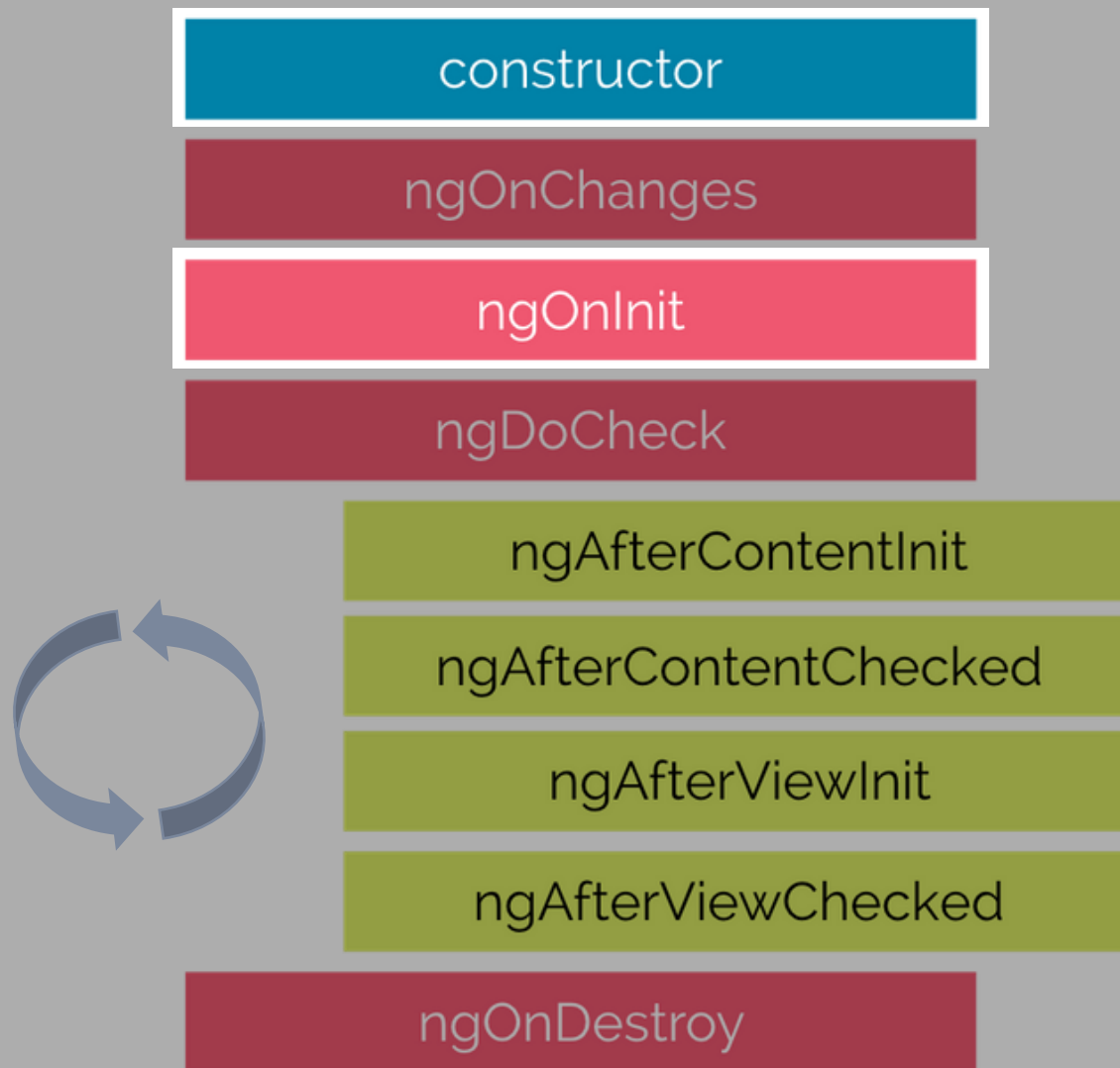
Theme

Create

Le cycle de vie du composant



Le cycle de vie du composant



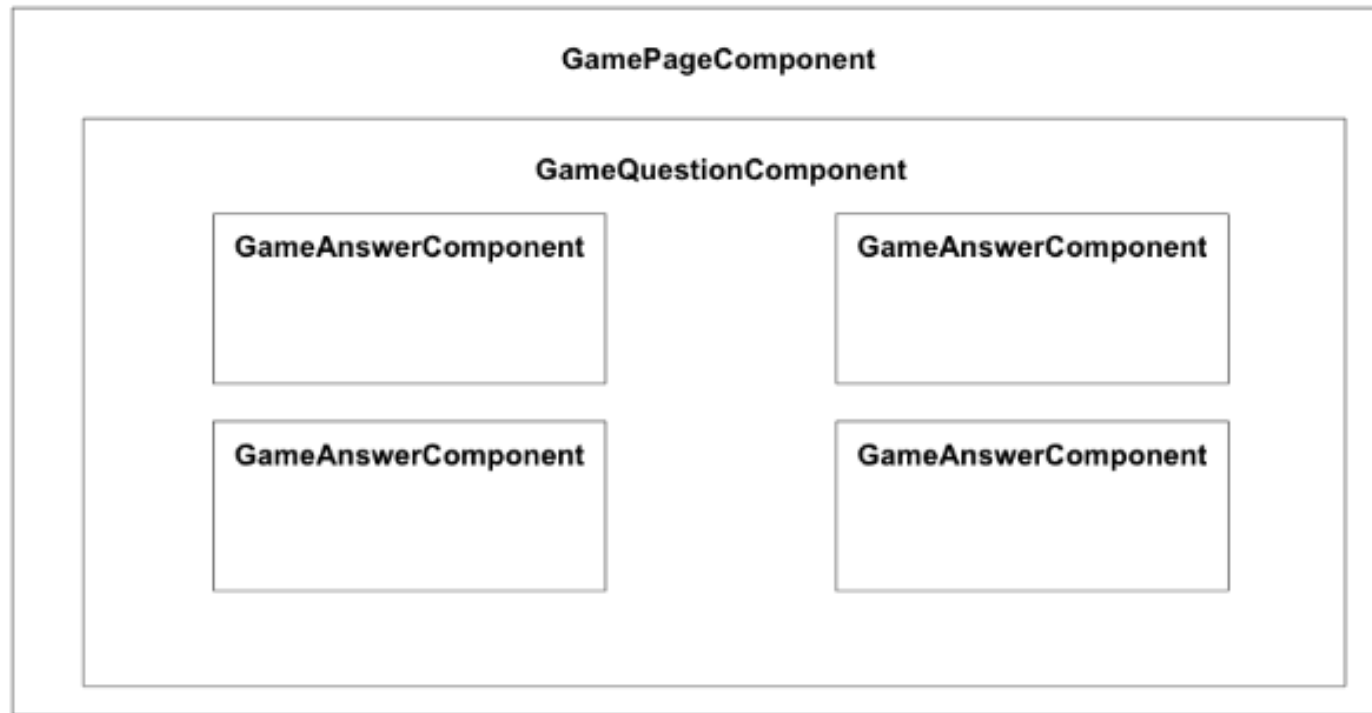
Le cycle de vie du composant

```
@Input()
quiz: Quiz;
constructor(public quizService: QuizService) {
  console.log('from constructor: this.quiz = ', this.quiz);
  this.quizService.quizzes$.subscribe((quizzes) => {
    console.log('into subscribe', quizzes);
  });
}

ngOnInit(): void {
  console.log('from ngOnInit: this.quiz = ', this.quiz);
}
```

```
from constructor: this.quiz = undefined          quiz.component.ts:25
into subscribe ► [{...}]                        quiz.component.ts:26
from ngOnInit: this.quiz =                      quiz.component.ts:30
► {id: '1', name: 'Les Acteurs', theme: 'Actor', questions: Array(0)}
```

Autour du modèle Quiz : IHM pour les accueillis ou résidents



Quels sont les modèles ?

Quels services écrire ?

Où placer des inputs et pourquoi ?

Où placer des outputs et pourquoi ?

Où placer des observer observables et pourquoi ?

TD : ajouter du comportement

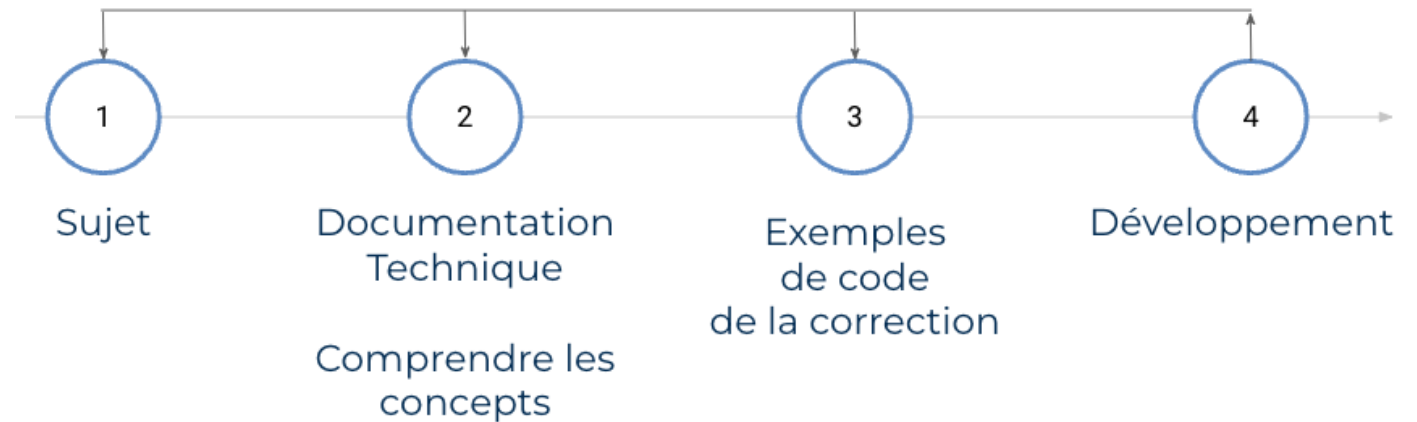
1. Comprendre les codes fournis pour les @input et @output dans les composants
2. Comprendre les services de la correction ([ps6-correction-td1-td2-v2/front-end/src/services/](#))
3. implémenter au fur et à mesure la navigation entre pages et les transferts de données en écrivant vos services dans le folder **services**
4. Implémenter les méthodes @input et @output dans les différents composants



TD : Informations importantes

1. Pas de back-end pour le moment ! On travaille dans le dossier front-end depuis votre repo starter de classroom

2. Processus de progression :

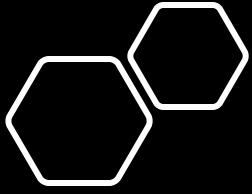


3. Commencez sur les 2 fonctionnalités :

1. Jouer/afficher un quiz
2. Afficher la liste des quiz et ajouter un quiz

Voir LMS

Travailler en pair programming



Prochains cours

- Comment bien gérer un jeu par utilisateur ?
 - Des types de données
 - Des spécificités CSS / HTML ?
-
- Comment écrire le serveur en NodeJS ?

AFFAIRE A SUIVRE