TECH CONFERENCE
DotNet
2020

#DotNet2020

Plantándole cara al Legacy Code con NgRx

ORGANIZATION

plain concepts

PLATINUM SPONSORS

Microsoft

COLLABORATORS

CEU | Universidad San Pablo

Thank you!

DotNet2020

#DotNet2020

Iván Reinoso García

Frontend Technical Lead en Plain Concepts

Apasionado de la informática y la tecnología desde pequeño, ha tenido la suerte de poder dedicarse profesionalmente a ello desde hace 11 años. Tras trabajar con diversos lenguajes y tecnologías dentro del ecosistema Microsoft, en los últimos años se ha especializado en tecnologías Frontend, consiguiendo ser un referente en los últimos proyectos en los que ha colaborado.

Actualmente es Frontend Technical Lead en Plain Concepts Madrid, intentando siempre aprender y mejorar junto a sus compañeros, aplicando buenas prácticas, desarrollando código limpio y apoyándose en las últimas tecnologías.

Si no encontráis ningún commit suyo en GitHub seguramente esté viajando, jugando al Super Mario, escuchando algún temazo de música electrónica, leyendo algún comic de Batman o automatizando su casa con el último juguete domótico de Xiaomi.

@ivanirega

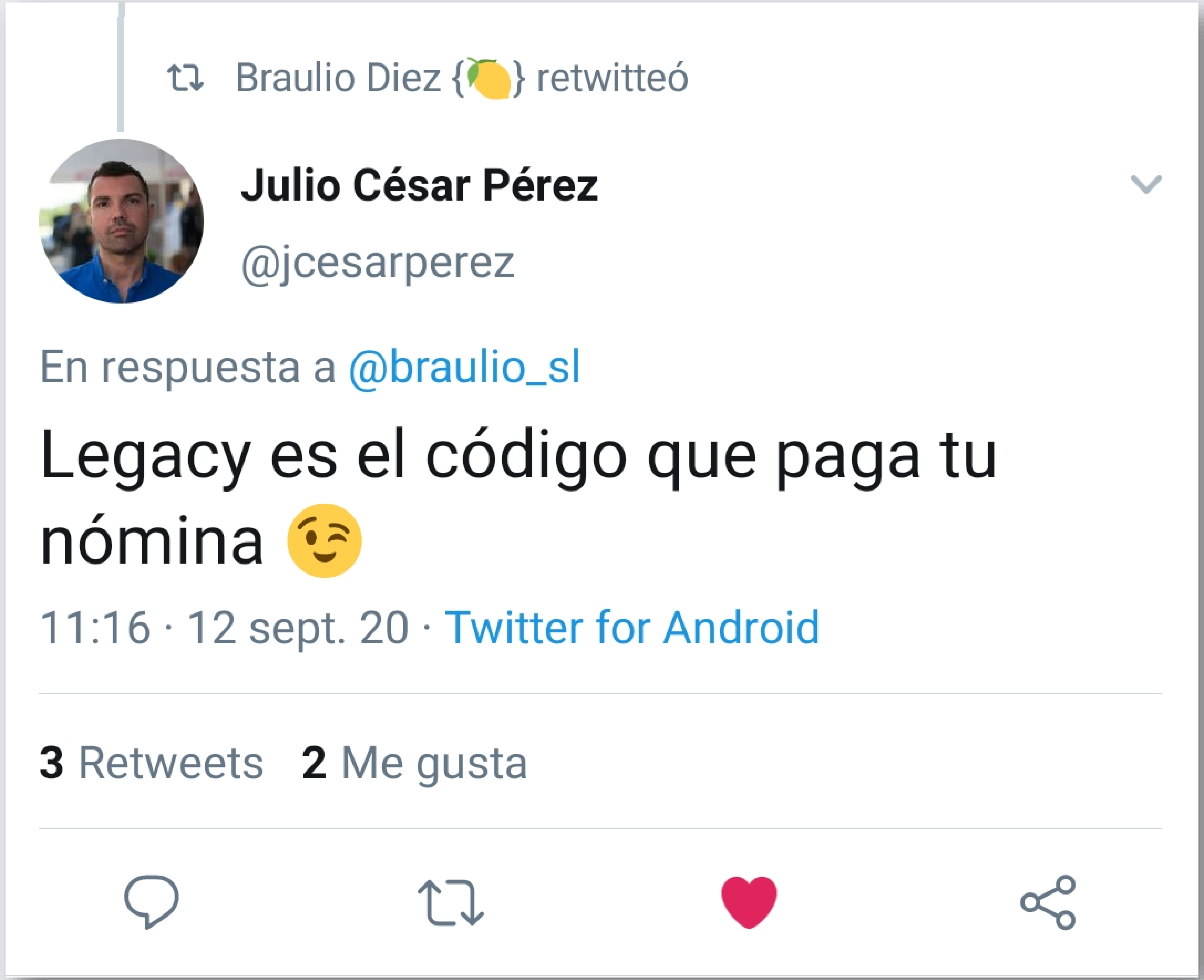
ireinoso@plainconcepts.com

Agenda

- ¿Qué es un Legacy Code?
- Toma de contacto
  - Contexto
  - Demo: Análisis de la funcionalidad
  - Análisis del Legacy Code
  - Primeros cambios
- Conceptos básicos
  - ¿Qué es una gestión centralizada del estado?
  - ¿Qué es el patrón Redux?
  - ¿Qué es NgRx?
  - Container/Presentational Components
  - Estrategia de detección de cambios OnPush
- Tomando el control
  - Implantación gradual del estado centralizado
  - Definiendo la estrategia de testing
  - Últimos cambios
- Próximos pasos
- Lecciones aprendidas
- Referencias
- Questions & Answers

¿Qué es un Legacy Code?

- Todo el código es Legacy Code.
- Código que no gusta a los desarrolladores.
- Código que los desarrolladores no entienden por falta de documentación.
- Código que los desarrolladores heredan.
- Código construido usando prácticas y plataformas obsoletas.
- Código sin tests (Michael Feathers).
- Código que es muy arriesgado cambiar.


Braulio Diez {🍋} retwitteó

Julio César Pérez

@jcesarperez

En respuesta a @braulio_sl

Legacy es el código que paga tu nómina 😉

11:16 · 12 sept. 20 · Twitter for Android

3 Retweets 2 Me gusta

Toma de contacto

Toma de contacto

Contexto

- Proyecto que se desarrolló como un MVP (Minimum Viable Product) por **una sola persona** en tiempo récord.
- Pasa a ser un producto **estratégico** para la compañía.
- Se necesita estabilizar, se han detectado algunos **bugs**.
- Muchas **nuevas funcionalidades** a implementar en un futuro cercano.
- **No se puede parar** completamente el desarrollo para estabilizar.
- Tiene **problemas de rendimiento** en pantallas donde se renderizan muchos datos.

Toma de contacto

Demo: Análisis de funcionalidad

Análisis del Legacy Code: Mezcla de responsabilidades

```
<!-- projects\original\src\app\components\custom-modal\custom-modal.component.html -->
   <!-- El modal me sirve para representar un listado de items seleccionables -->
   <span *ngIf="params && params.input">
     <label
       class="custom-label flex mt-2 ml-3"
       *ngFor="let item of params.input"
     >
         <input
           type="checkbox"
           name="item"
           (change)="changeStatus(item, $event.target.checked)"
           [attr.checked]="null"
           value="item"
         />
       <span class="select-none">{{ item }}</span>
     </label>
   </span>
   <!-- ¡Y también para mostrar un input editable! -->
   <div *ngIf="params && params.createEdit" class="px-5 pt-2 pb-8 mb-4">
       <label class="block text-gray-700 text-sm font-bold mb-2">
         {{ params && params.labelInputText && params.labelInputText.text }}
       </label>
       <input class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight"
         type="text“ [(ngModel)]="params.title"
       />
   </div>
```

Análisis del Legacy Code: GodComponent

```
// projects\original\src\app\components\question-group\question-group.component.ts
export class QuestionGroupComponent implements OnChanges, OnDestroy {
  receiveMessageFromContent(data: any) {
    const { action, value } = data;
    switch (action) {
      // Hace de todo, lo mismo te añade una pregunta que te frie un huevo
      case "content-question-add":
        // [TL;DR]
        break;
      case "content-question-update":
       // [TL;DR]
        break;
      case "content-question-toggle":
        // [TL;DR]
        break;
      case "content-answer-add":
        // [TL;DR]
        break;
      case "content-topic-select":
        // [TL;DR]
        break;
    }
  }
  // Referencias a todos los servicios de la aplicación
  // Más de 600 líneas de código
```

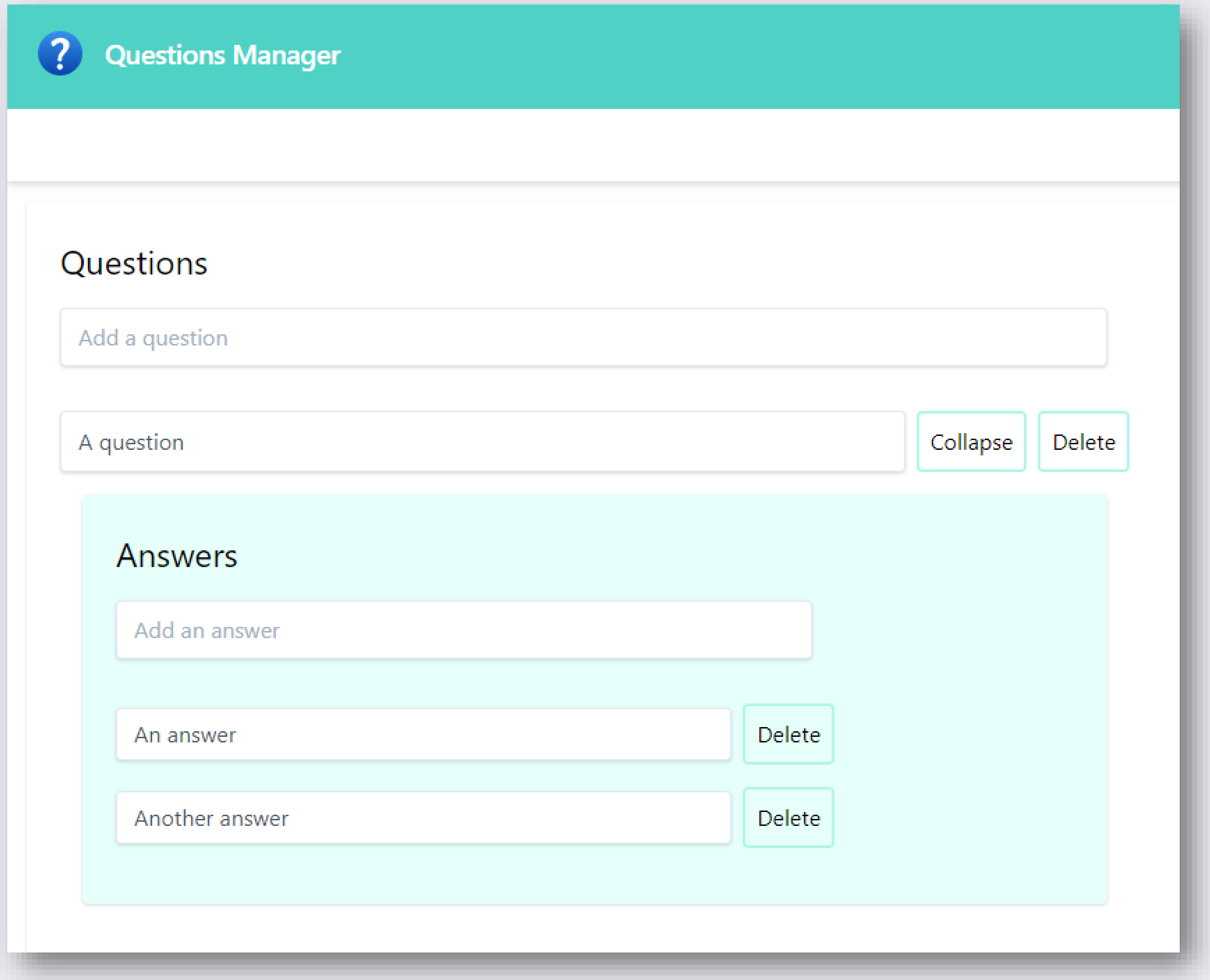
Análisis del Legacy Code: Lógica de negocio en componentes

```
// projects\original\src\app\components\question-group\question-group.component.ts
case "content-question-toggle":
        if (value === this.openedQuestionId) {
          this.openedQuestionId = null;
        } else {
          this.openedQuestionId = value;
        }
        this.getAnswers(value).then(() => this.getQuestionGroup());
        break;

private getQuestionGroup() {
    return this.questionGroupsService
      .get(+this.id)
      .toPromise()
      .then(data => {
        this.questionGroup = data;
        this.questions = this.questionGroup.questions.map(q => ({
          ...q,
          isOpened: q.id === +this.openedQuestionId
        }));
        this.selectedTopicIds = this.questionGroup.topicIds;
        this.numberOfQuestions = this.questionGroup.questions.length;
        this.ref.detectChanges();
      });
  }
```

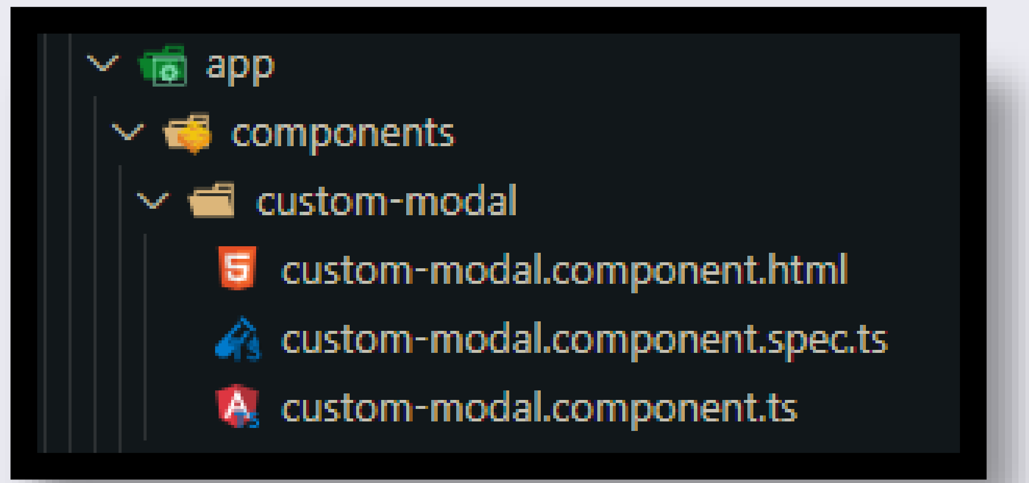
Análisis del Legacy Code

Problemas de rendimiento

- Cuando existe un gran número de preguntas y respuestas, la aplicación empieza a ralentizarse.
- Confirmamos que usa la estrategia de detección de cambios por defecto.
- Versión 7 de Angular.

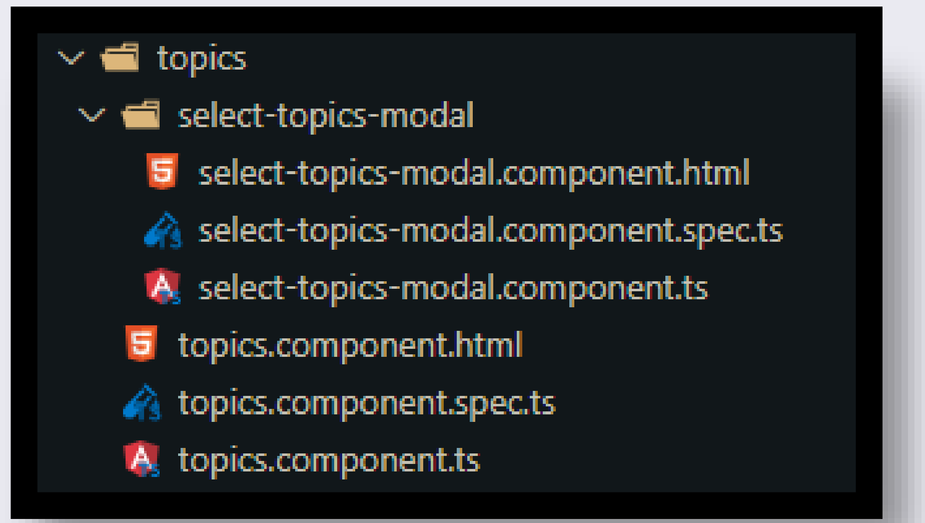


Primeros cambios: Separando responsabilidades



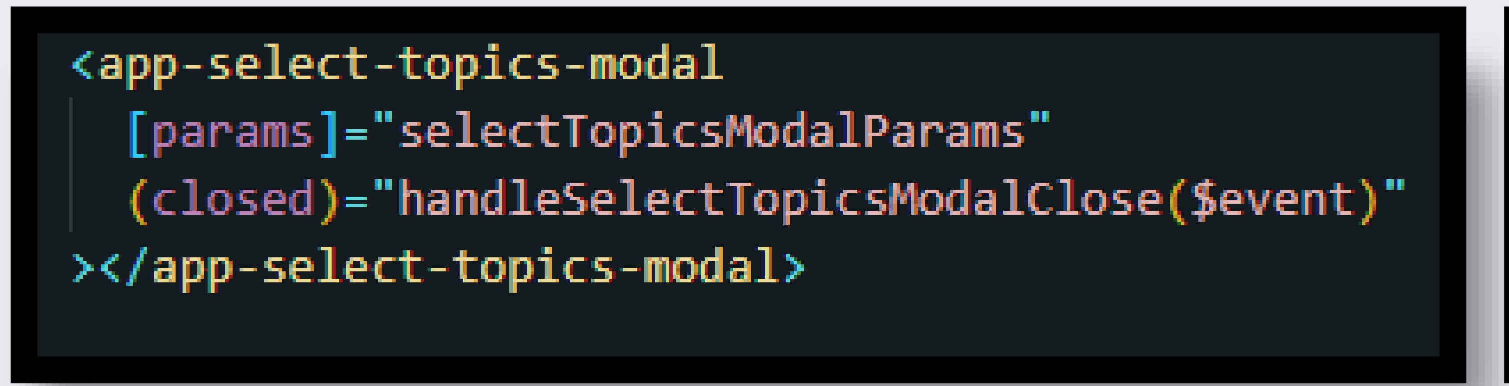
```
app
  components
    custom-modal
      custom-modal.component.html
      custom-modal.component.spec.ts
      custom-modal.component.ts
```

1. Copia

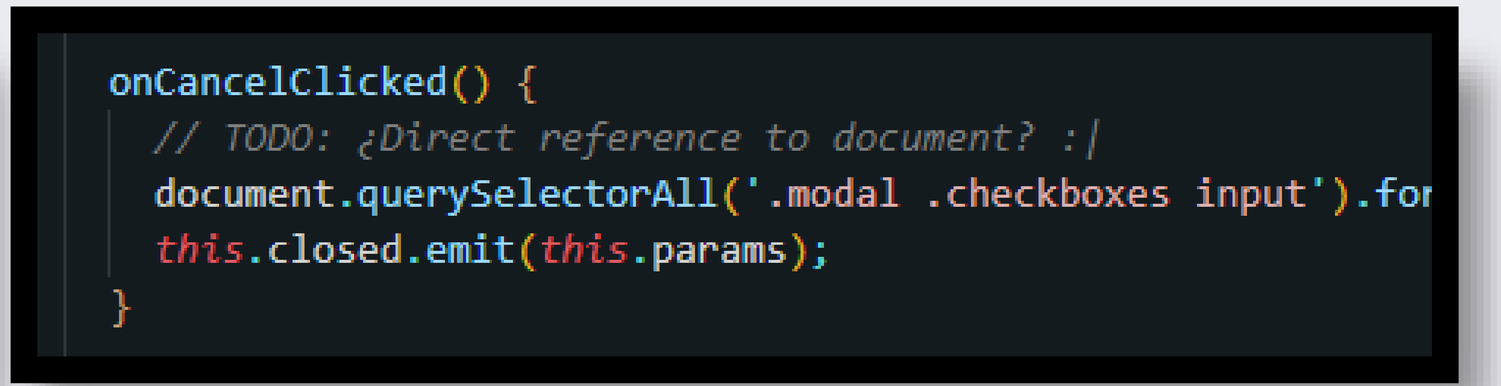


```
topics
  select-topics-modal
    select-topics-modal.component.html
    select-topics-modal.component.spec.ts
    select-topics-modal.component.ts
  topics.component.html
  topics.component.spec.ts
  topics.component.ts
```

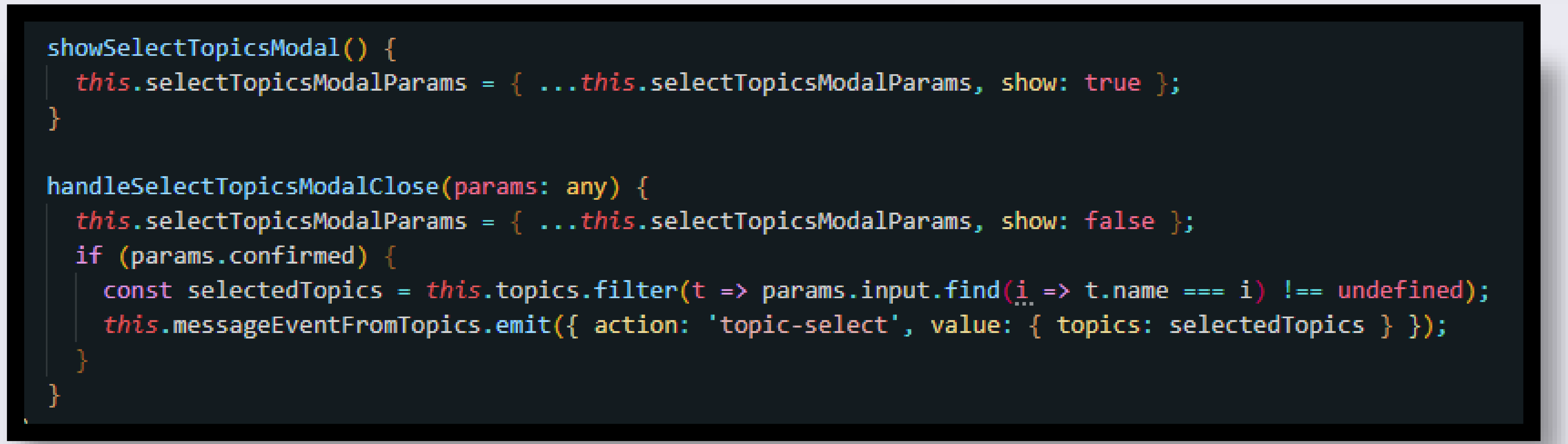
2. Adaptación



```
<app-select-topics-modal
  [params]="selectTopicsModalParams"
  (closed)="handleSelectTopicsModalClose($event)"
></app-select-topics-modal>
```

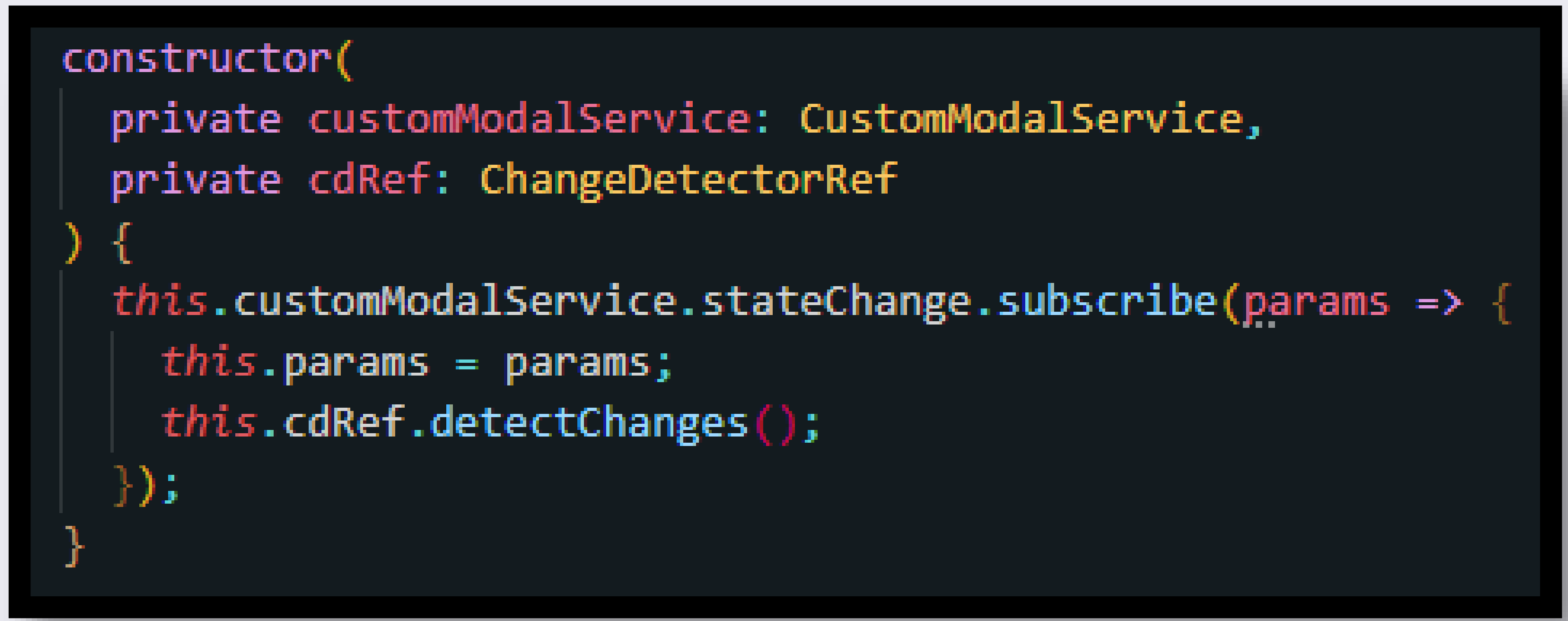


```
onCancelClicked() {
  // TODO: ¿Direct reference to document? :|
  document.querySelectorAll('.modal .checkboxes input').for
  this.closed.emit(this.params);
}
```

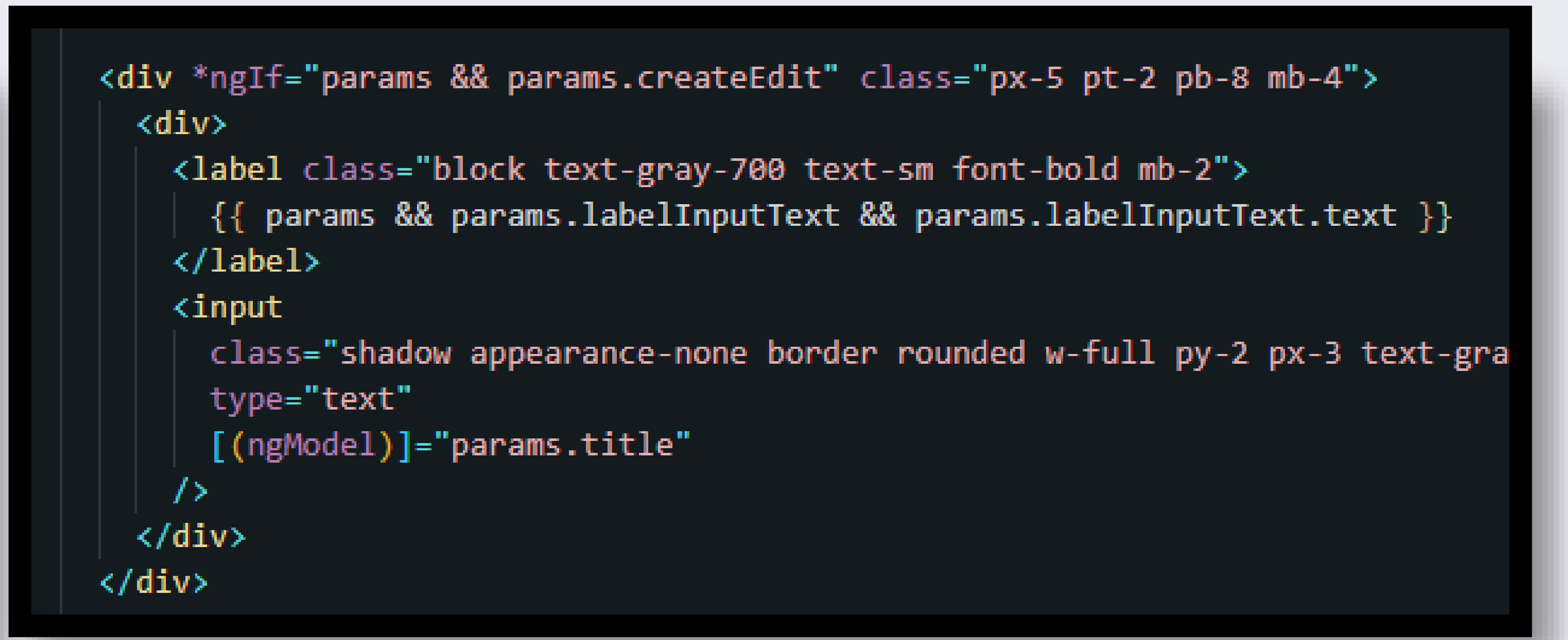


```
showSelectTopicsModal() {
  this.selectTopicsModalParams = { ...this.selectTopicsModalParams, show: true };
}

handleSelectTopicsModalClose(params: any) {
  this.selectTopicsModalParams = { ...this.selectTopicsModalParams, show: false };
  if (params.confirmed) {
    const selectedTopics = this.topics.filter(t => params.input.find(i => t.name === i) !== undefined);
    this.messageEventFromTopics.emit({ action: 'topic-select', value: { topics: selectedTopics } });
  }
}
```

3. Limpieza



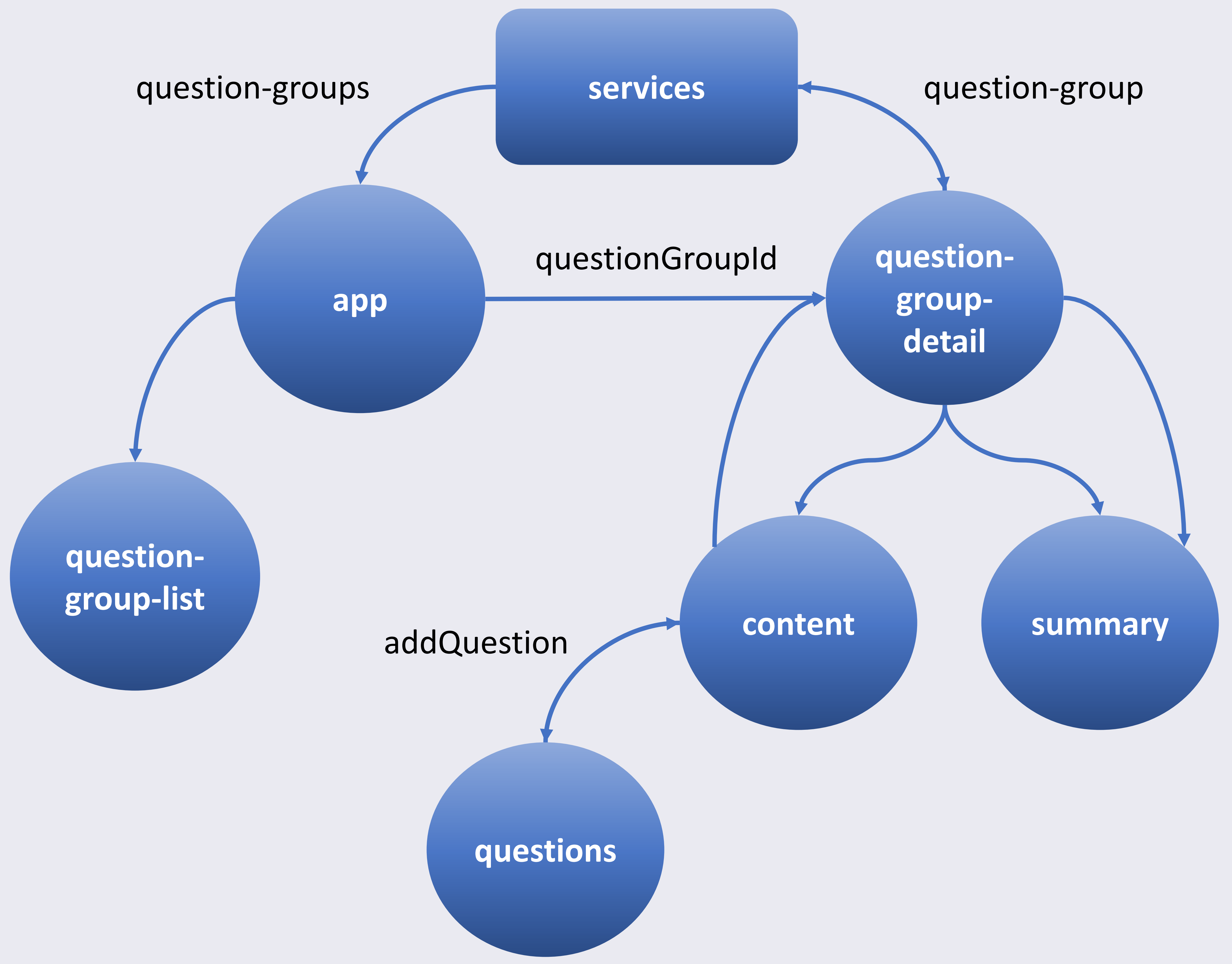
```
constructor(
  private customModalService: CustomModalService,
  private cdRef: ChangeDetectorRef
) {
  this.customModalService.stateChange.subscribe(params => {
    this.params = params;
    this.cdRef.detectChanges();
  });
}
```

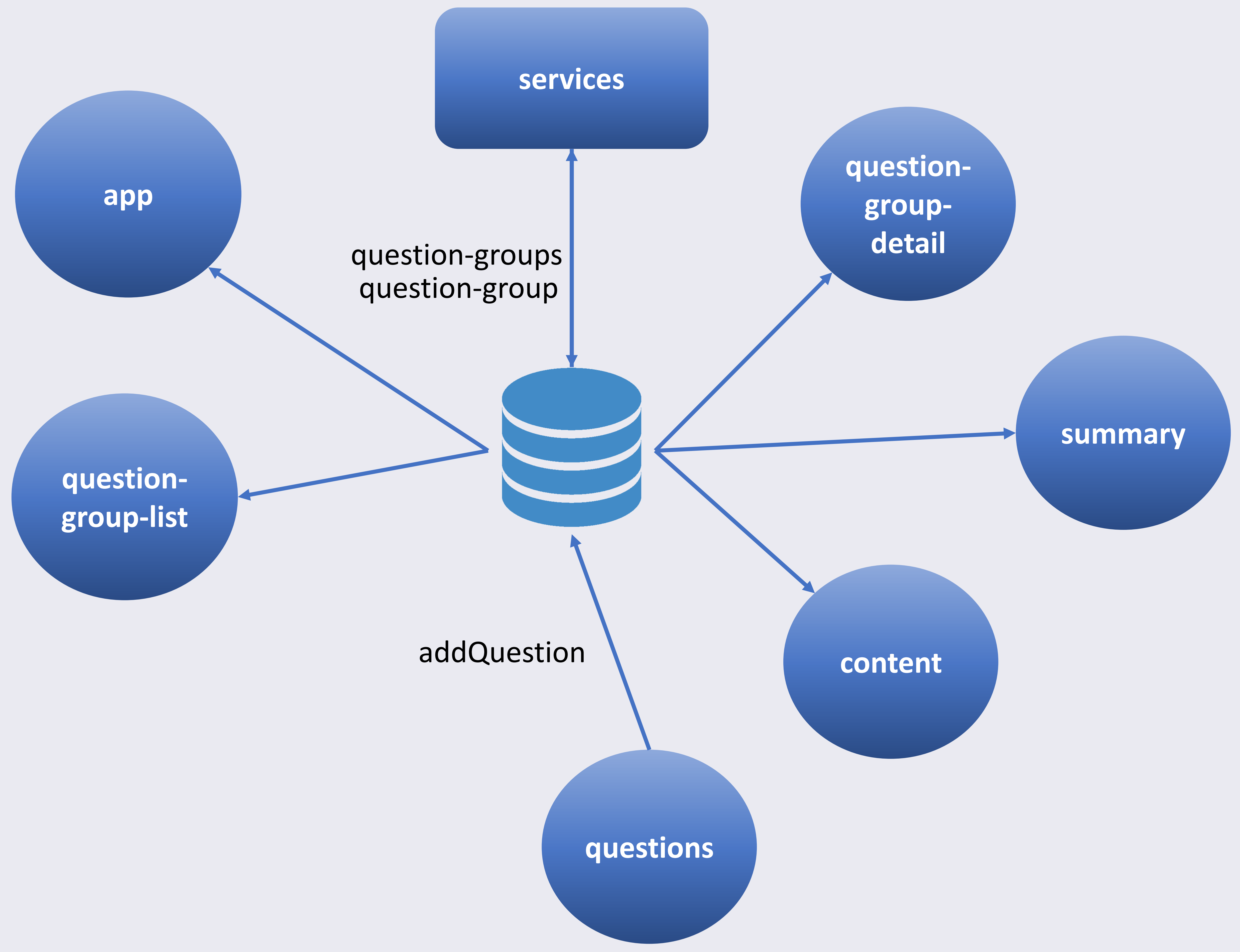


```
<div *ngIf="params && params.createEdit" class="px-5 pt-2 pb-8 mb-4">
  <div>
    <label class="block text-gray-700 text-sm font-bold mb-2">
      {{ params && params.labelInputText && params.labelInputText.text }}
    </label>
    <input
      class="shadow appearance-none border rounded w-full py-2 px-3 text-gra
      type="text"
      [(ngModel)]="params.title"
    />
  </div>
</div>
```

Conceptos básicos

¿Qué es una gestión centralizada del estado?

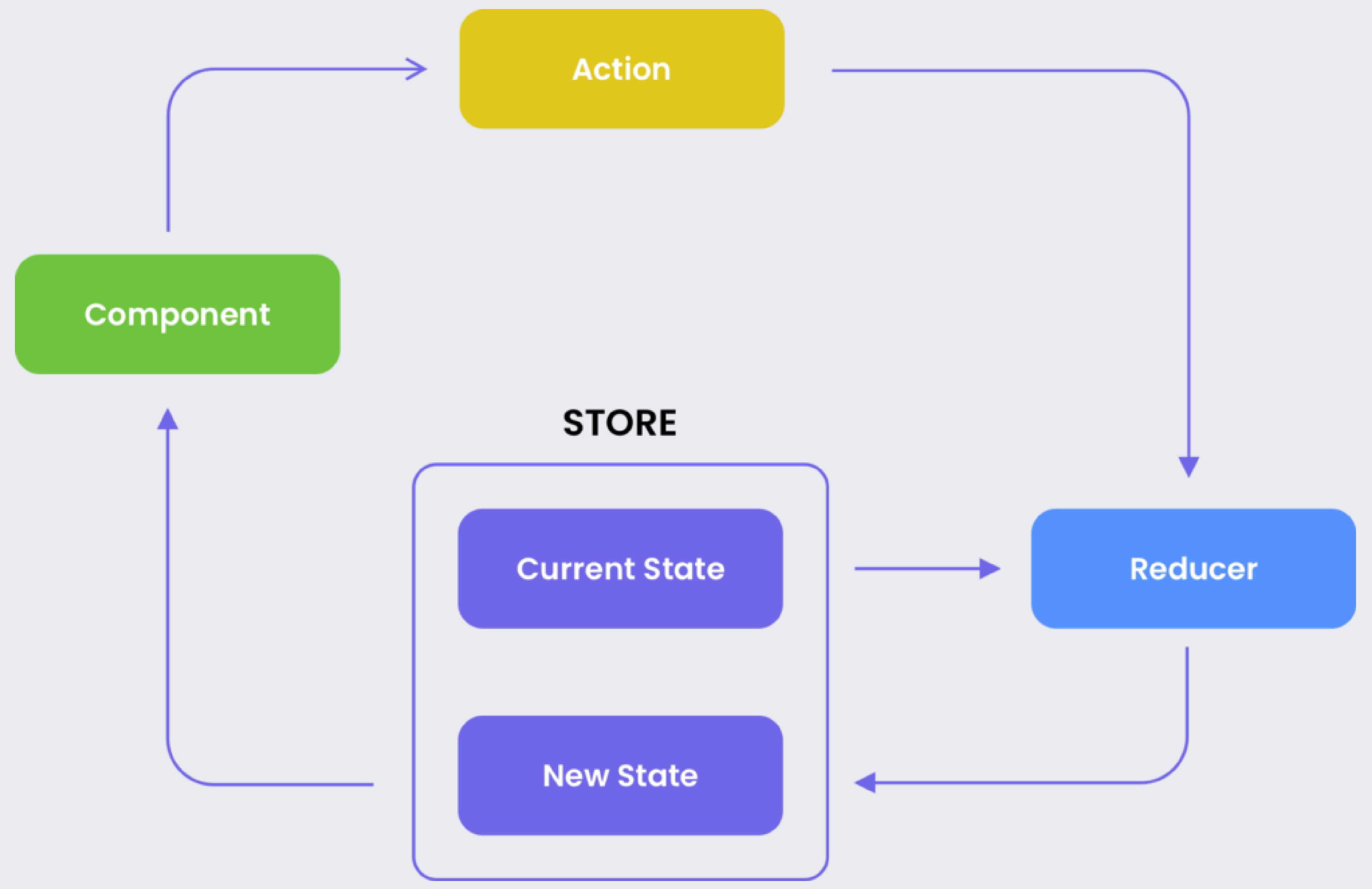




¿Qué es el patrón Redux?

Se basa en 3 principios:

- **Única fuente de la verdad.**
- **El estado es de solo lectura.**
- **Los cambios se realizan con funciones puras.**



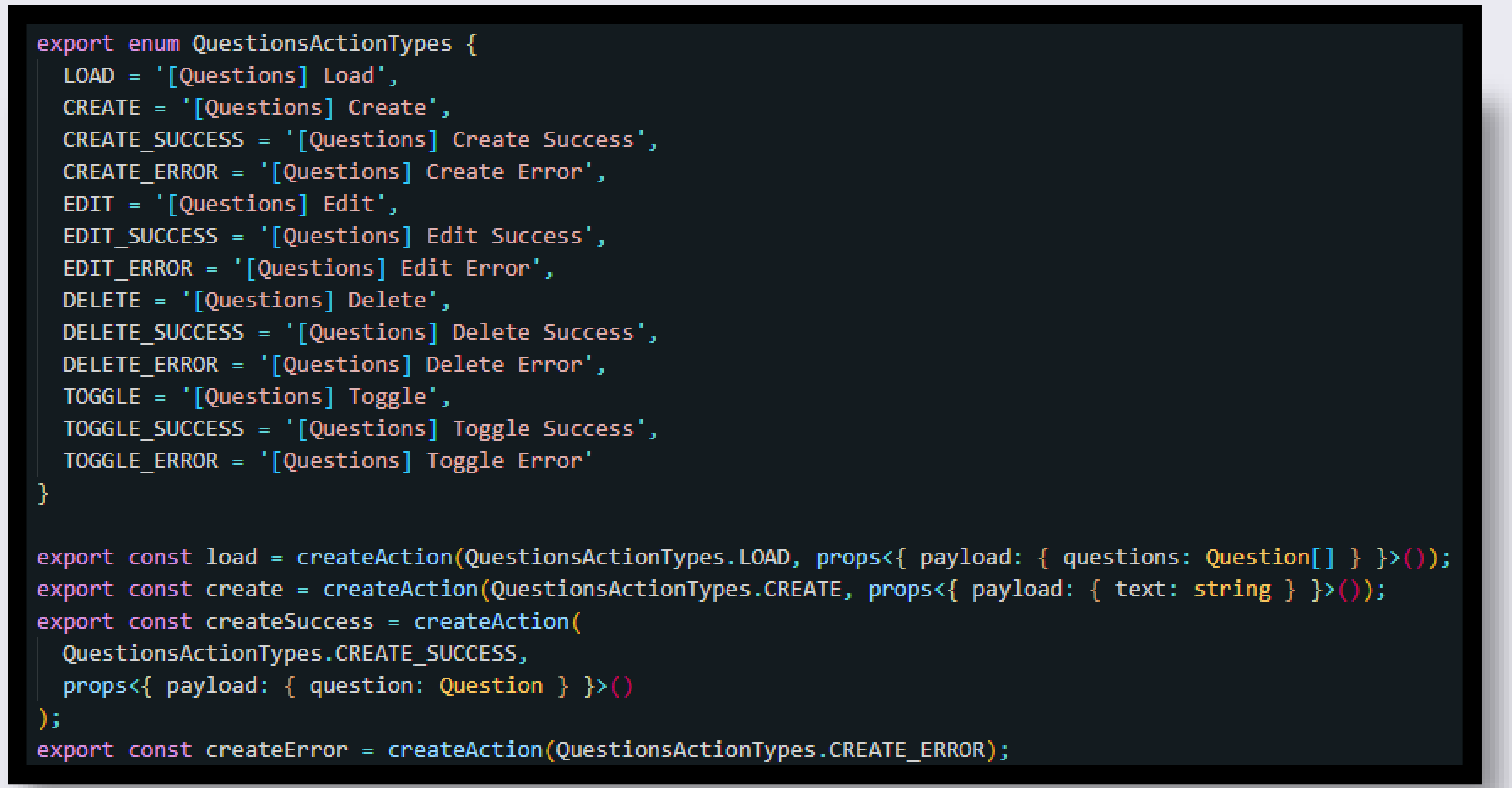
¿Qué es NgRx?

Implementación del patrón Redux en Angular con RxJS.

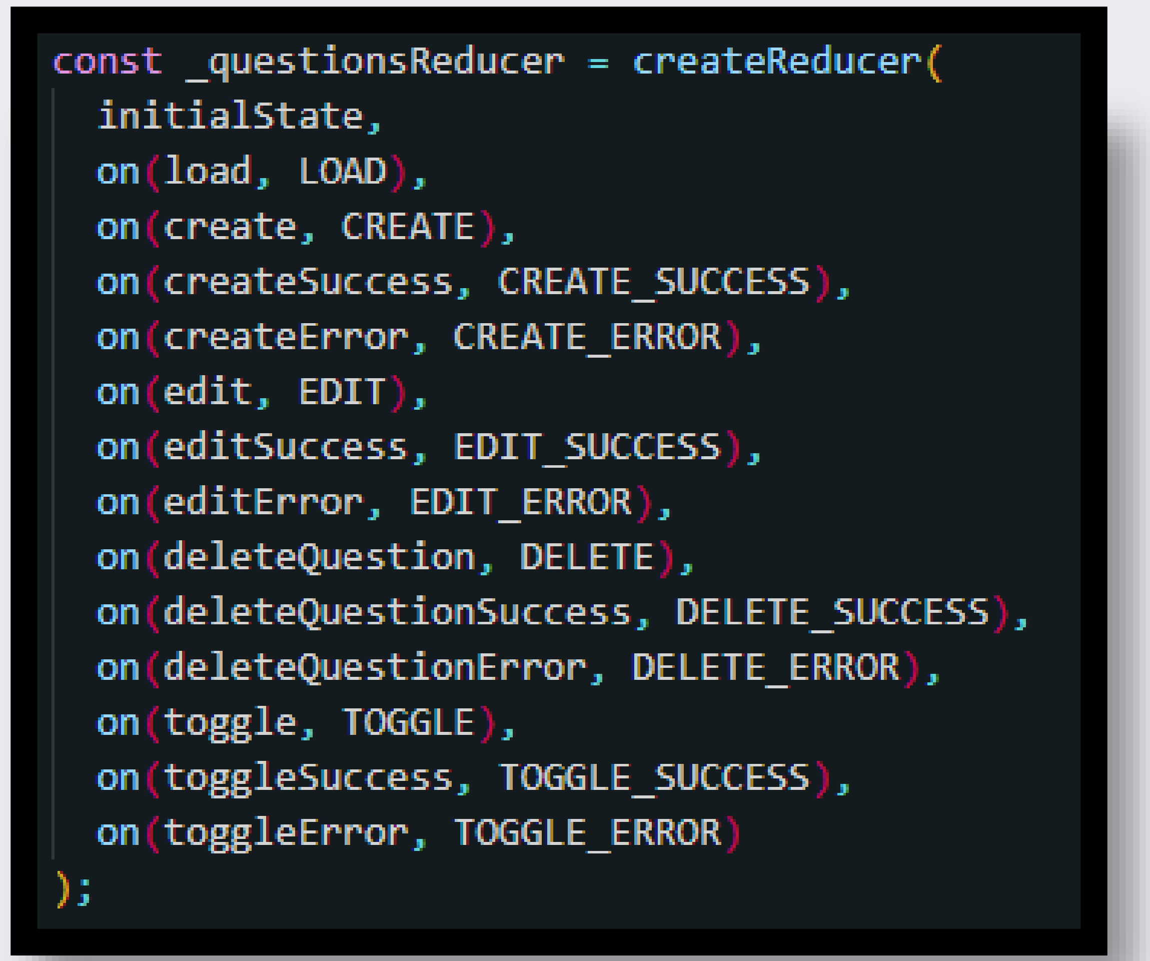
Características principales:

- Estado compartido.
- Inmutabilidad y rendimiento.
- Encapsulación.
- Serializable: Store Devtools.
- Testable.

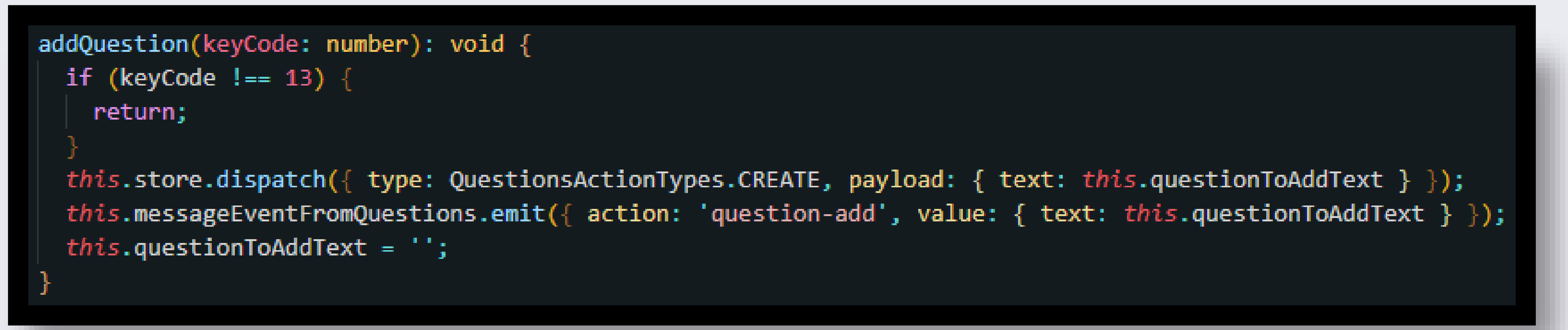
¿Qué es NgRx? Actions & Reducers



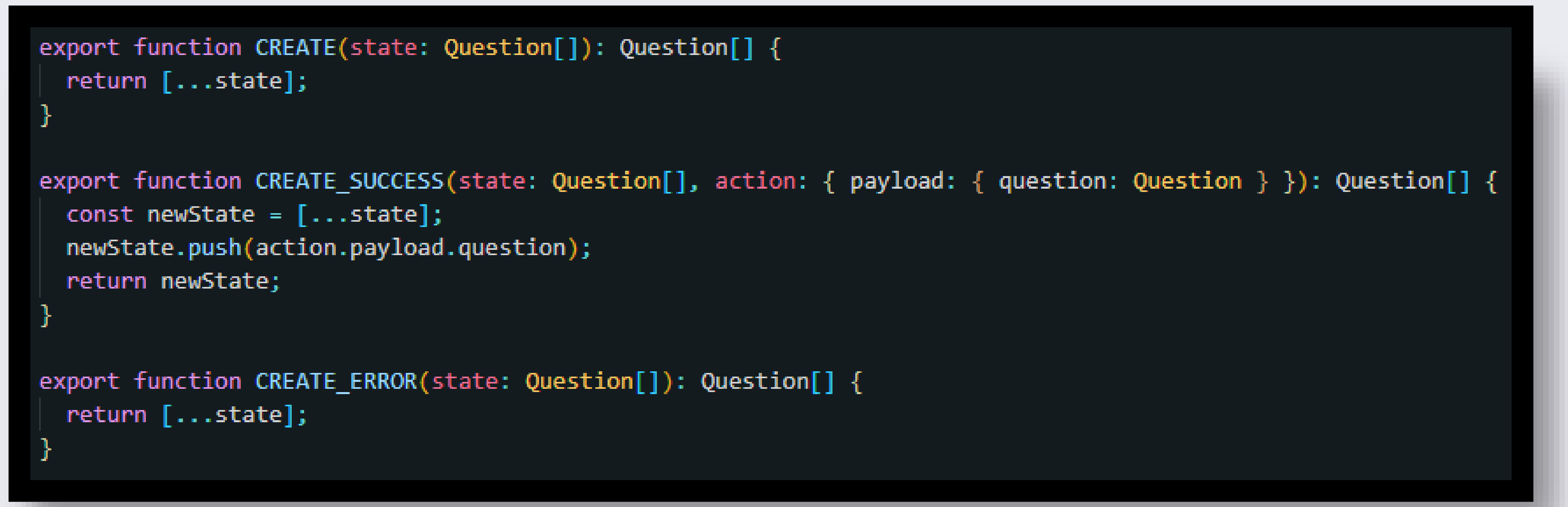
```
export enum QuestionsActionTypes {
  LOAD = '[Questions] Load',
  CREATE = '[Questions] Create',
  CREATE_SUCCESS = '[Questions] Create Success',
  CREATE_ERROR = '[Questions] Create Error',
  EDIT = '[Questions] Edit',
  EDIT_SUCCESS = '[Questions] Edit Success',
  EDIT_ERROR = '[Questions] Edit Error',
  DELETE = '[Questions] Delete',
  DELETE_SUCCESS = '[Questions] Delete Success',
  DELETE_ERROR = '[Questions] Delete Error',
  TOGGLE = '[Questions] Toggle',
  TOGGLE_SUCCESS = '[Questions] Toggle Success',
  TOGGLE_ERROR = '[Questions] Toggle Error'
}

export const load = createAction(QuestionsActionTypes.LOAD, props<{ payload: { questions: Question[] } }>());
export const create = createAction(QuestionsActionTypes.CREATE, props<{ payload: { text: string } }>());
export const createSuccess = createAction(
  QuestionsActionTypes.CREATE_SUCCESS,
  props<{ payload: { question: Question } }>()
);
export const createError = createAction(QuestionsActionTypes.CREATE_ERROR);
```



```
const _questionsReducer = createReducer(
  initialState,
  on(load, LOAD),
  on(create, CREATE),
  on(createSuccess, CREATE_SUCCESS),
  on(createError, CREATE_ERROR),
  on(edit, EDIT),
  on(editSuccess, EDIT_SUCCESS),
  on(editError, EDIT_ERROR),
  on(deleteQuestion, DELETE),
  on(deleteQuestionSuccess, DELETE_SUCCESS),
  on(deleteQuestionError, DELETE_ERROR),
  on(toggle, TOGGLE),
  on(toggleSuccess, TOGGLE_SUCCESS),
  on(toggleError, TOGGLE_ERROR)
);
```

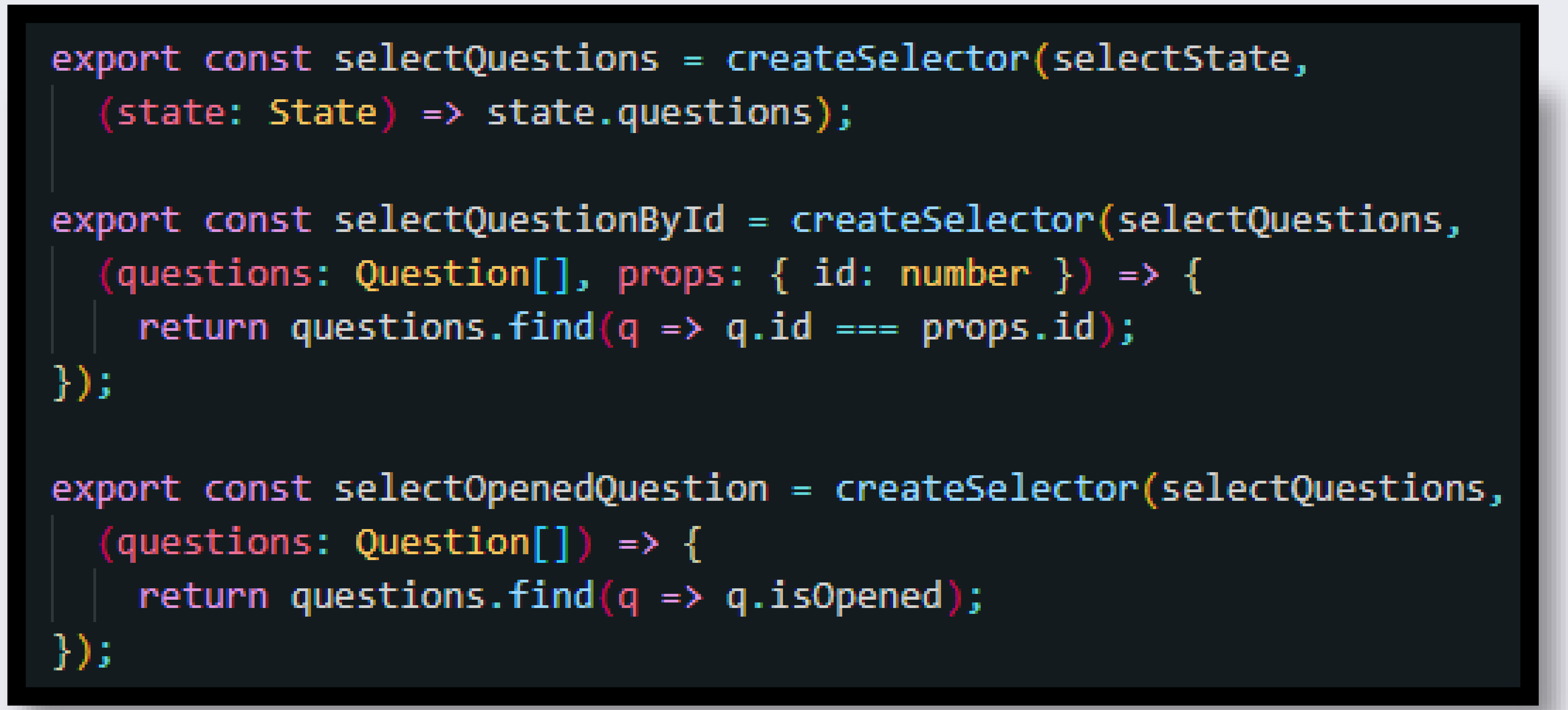


```
addQuestion(keyCode: number): void {
  if (keyCode !== 13) {
    return;
  }
  this.store.dispatch({ type: QuestionsActionTypes.CREATE, payload: { text: this.questionToAddText } });
  this.messageEventFromQuestions.emit({ action: 'question-add', value: { text: this.questionToAddText } });
  this.questionToAddText = '';
}
```

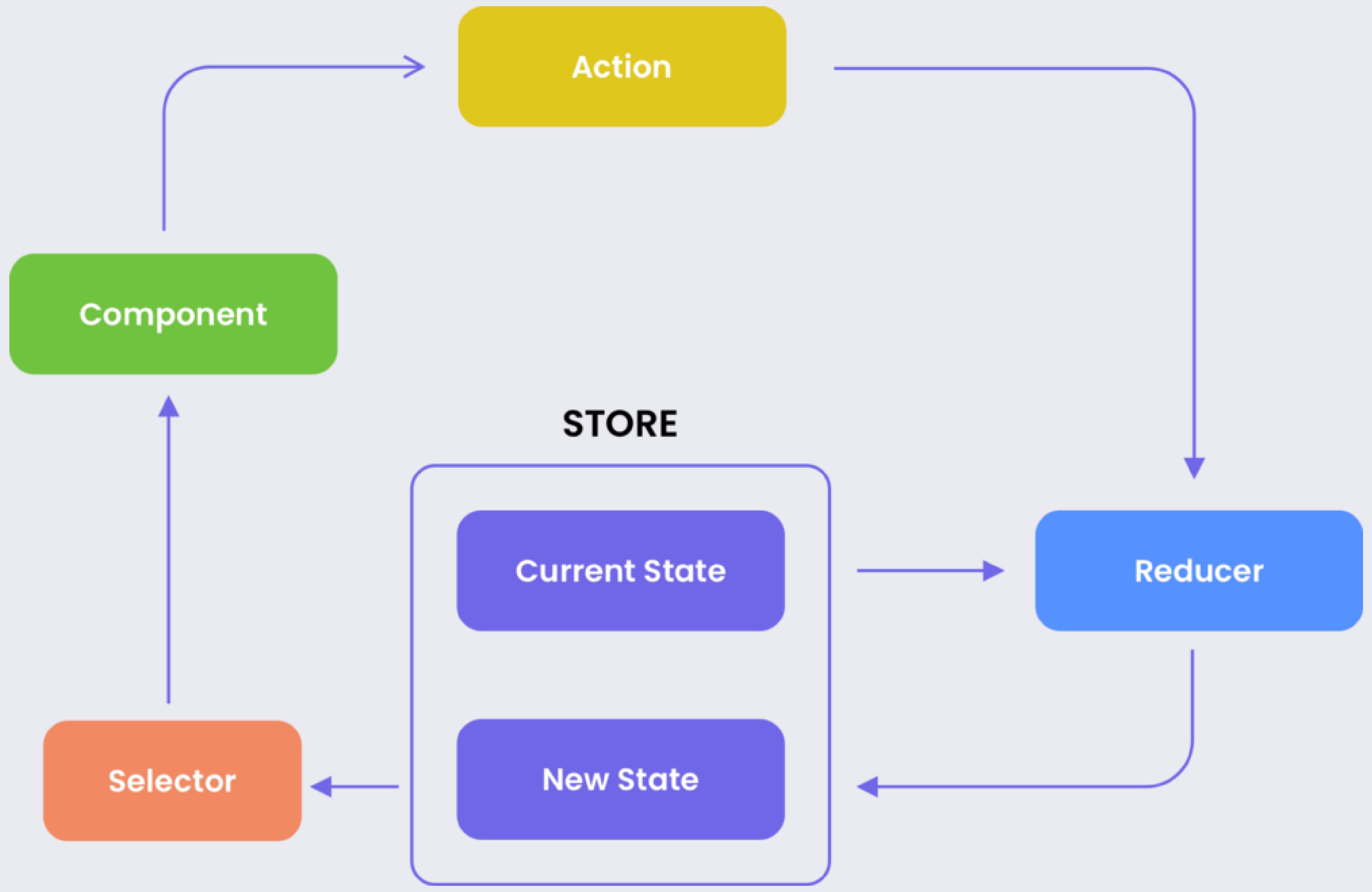


```
export function CREATE(state: Question[]): Question[] {
  return [...state];
}

export function CREATE_SUCCESS(state: Question[], action: { payload: { question: Question } }): Question[] {
  const newState = [...state];
  newState.push(action.payload.question);
  return newState;
}

export function CREATE_ERROR(state: Question[]): Question[] {
  return [...state];
}
```

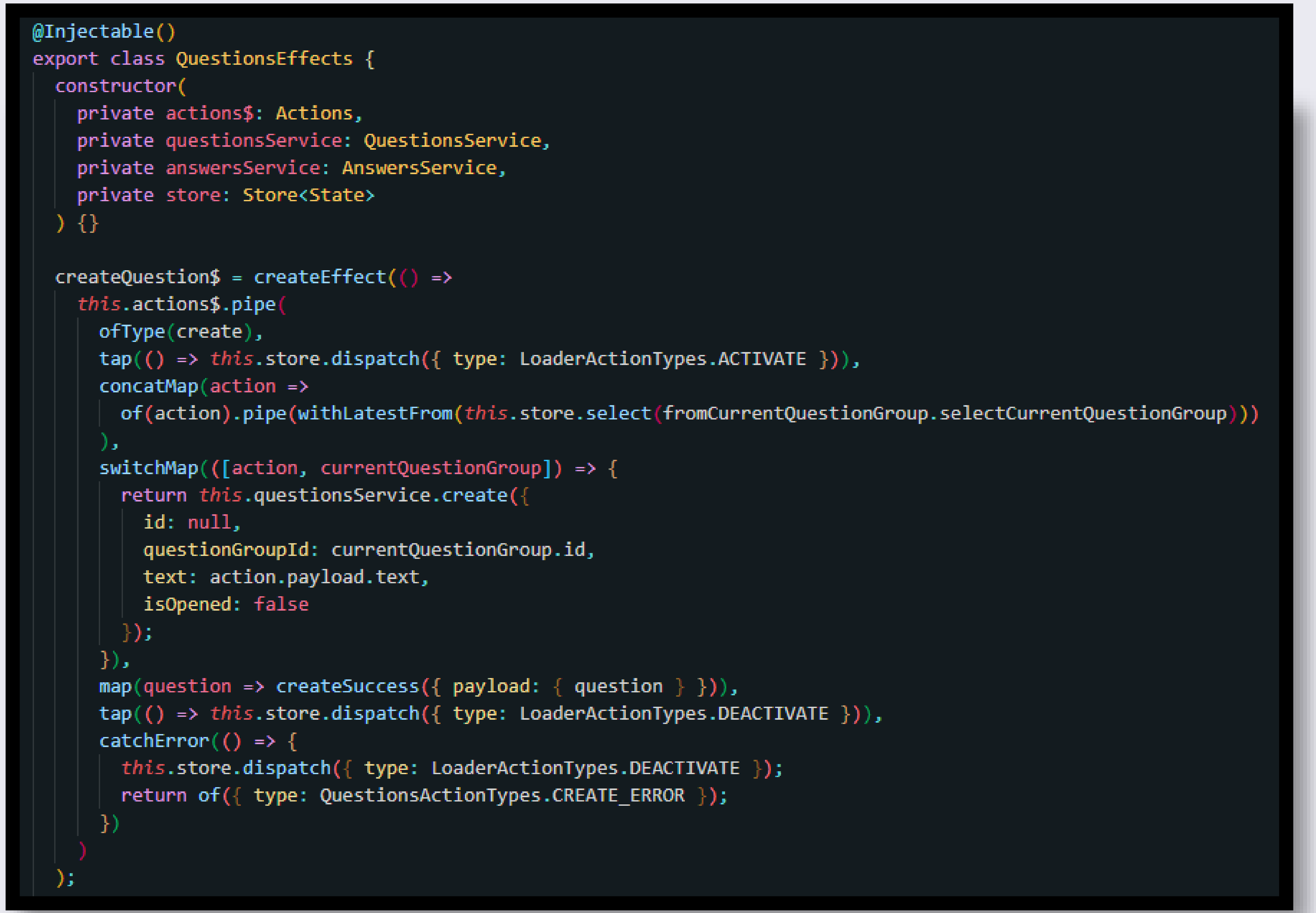
¿Qué es NgRx? Selectors



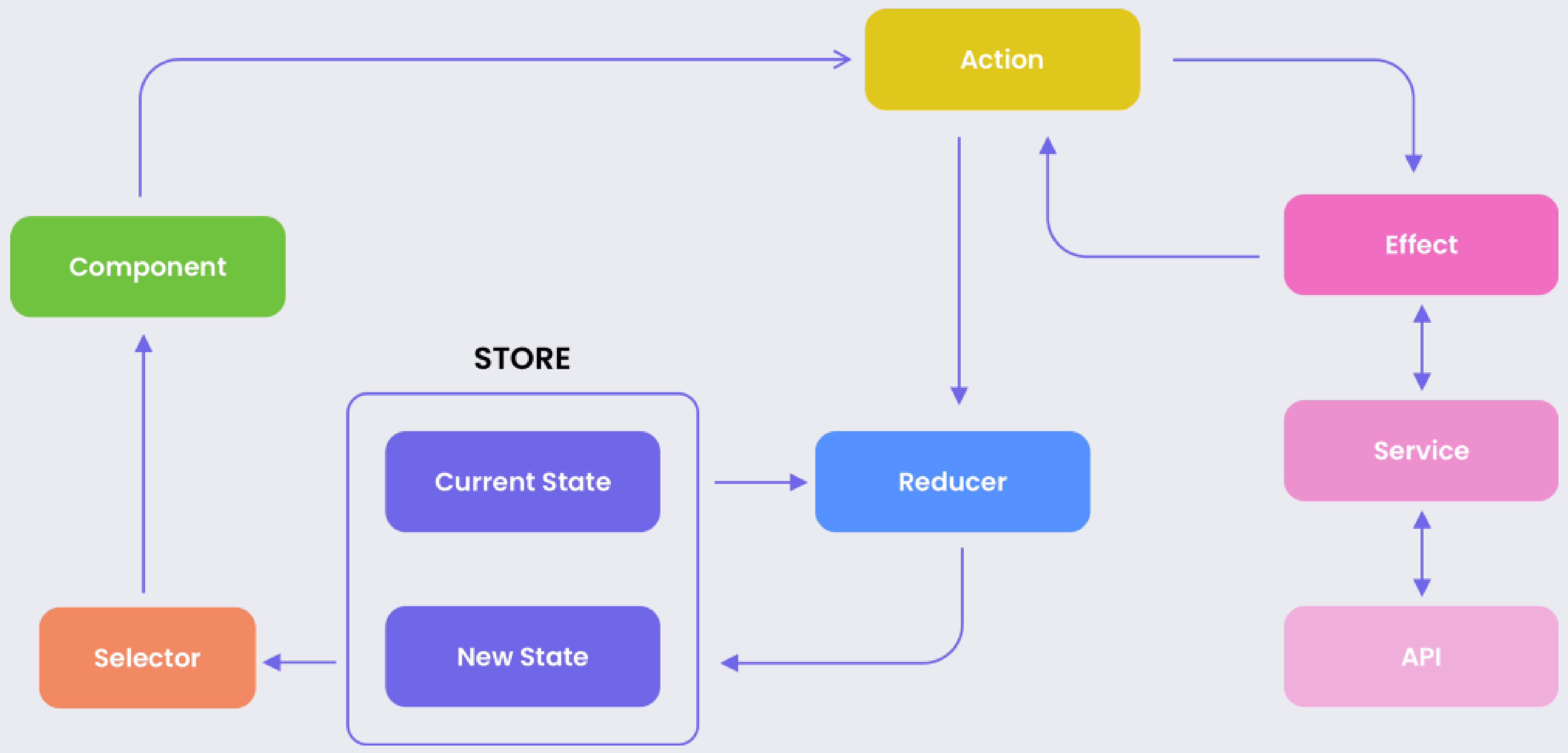
```
export const selectQuestions = createSelector(selectState,
  (state: State) => state.questions);

export const selectQuestionById = createSelector(selectQuestions,
  (questions: Question[], props: { id: number }) => {
    return questions.find(q => q.id === props.id);
});

export const selectOpenedQuestion = createSelector(selectQuestions,
  (questions: Question[]) => {
    return questions.find(q => q.isOpened);
});
```



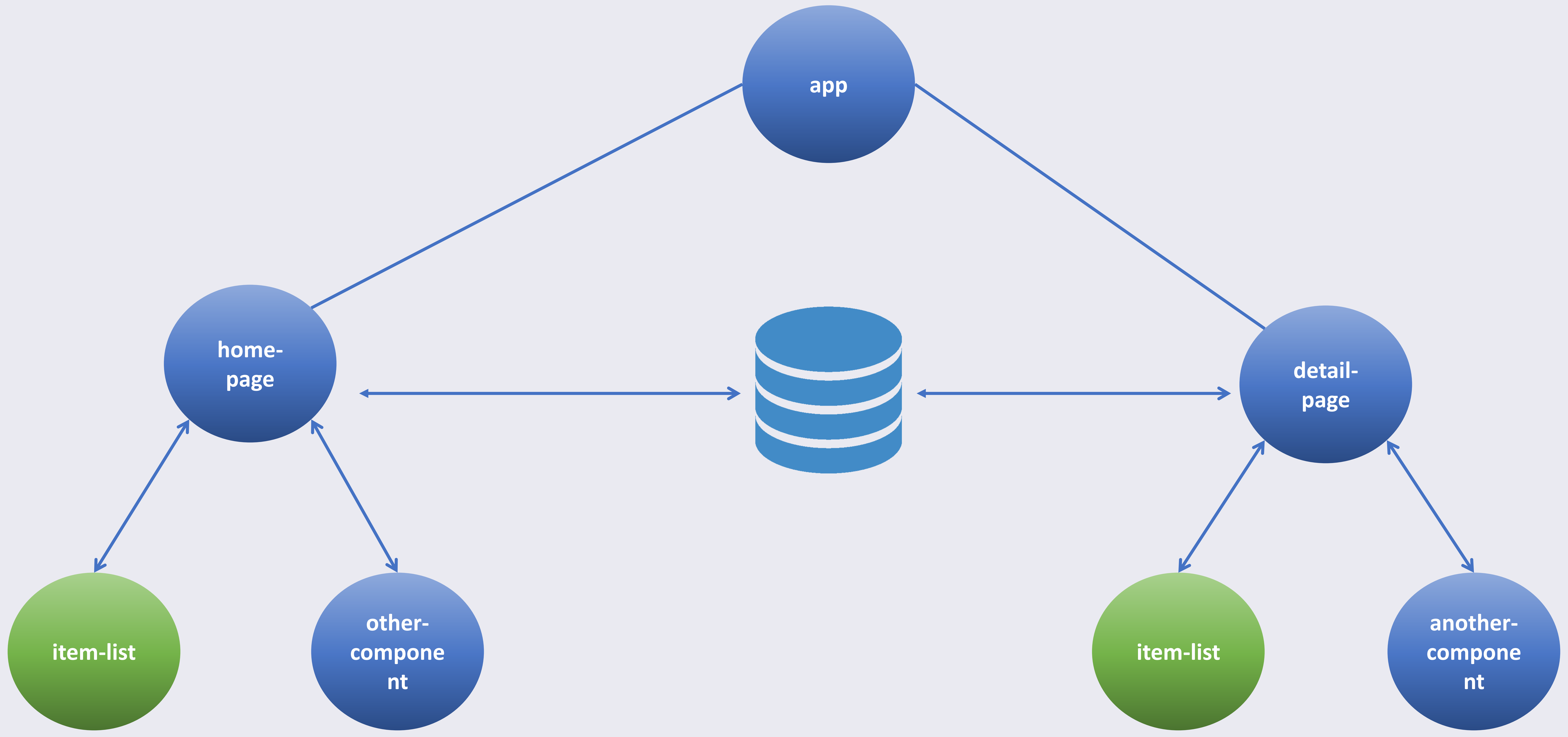
¿Qué es NgRx? Effects



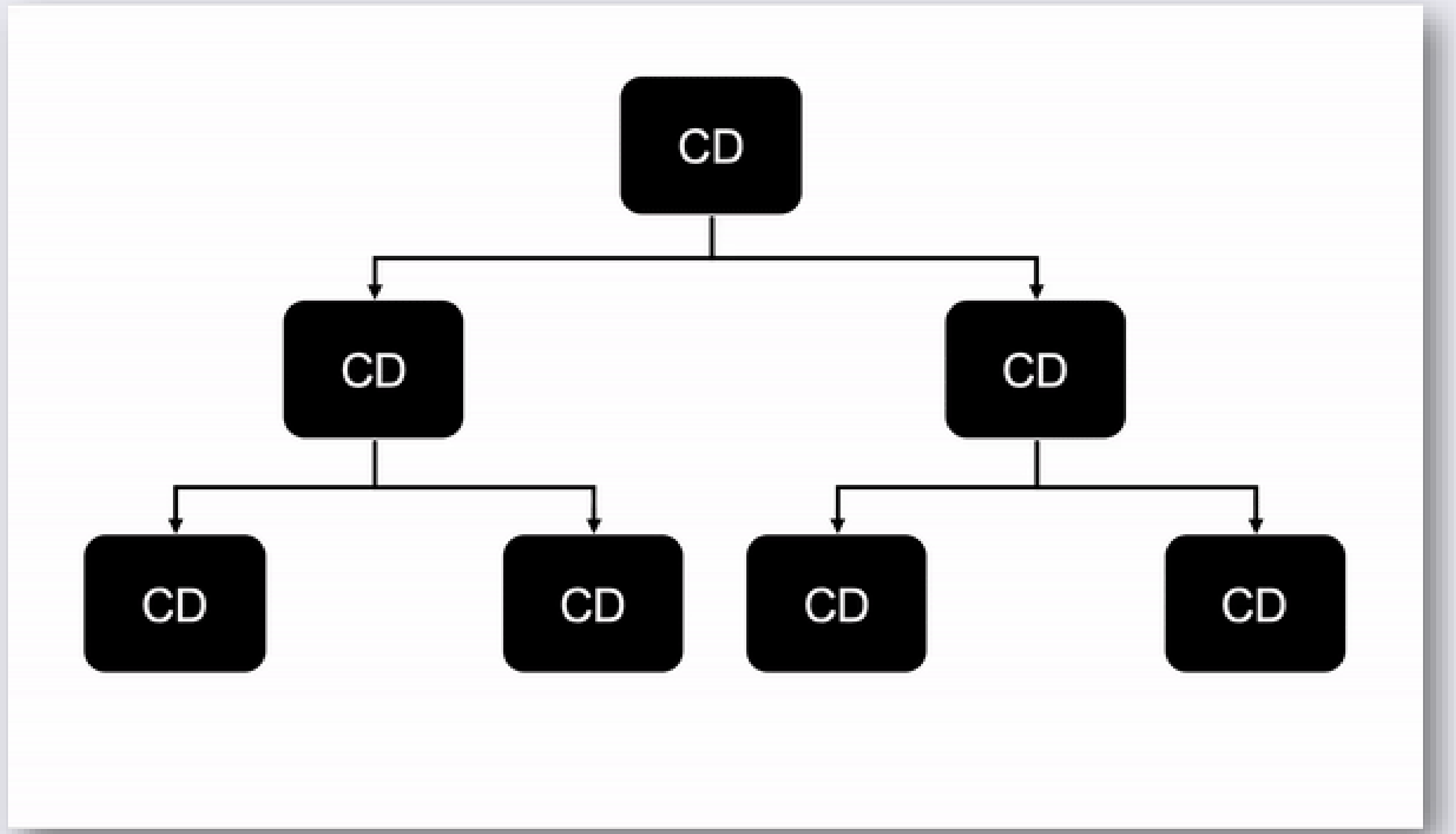
```
@Injectable()
export class QuestionsEffects {
  constructor(
    private actions$: Actions,
    private questionsService: QuestionsService,
    private answersService: AnswersService,
    private store: Store<State>
  ) {}

  createQuestion$ = createEffect(() =>
    this.actions$.pipe(
      ofType(create),
      tap(() => this.store.dispatch({ type: LoaderActionTypes.ACTIVATE })),
      concatMap(action =>
        of(action).pipe(withLatestFrom(this.store.select(fromCurrentQuestionGroup.selectCurrentQuestionGroup)))
      ),
      switchMap(([action, currentQuestionGroup]) => {
        return this.questionsService.create({
          id: null,
          questionGroupId: currentQuestionGroup.id,
          text: action.payload.text,
          isOpened: false
        });
      }),
      map(question => createSuccess({ payload: { question } })),
      tap(() => this.store.dispatch({ type: LoaderActionTypes.DEACTIVATE })),
      catchError(() => {
        this.store.dispatch({ type: LoaderActionTypes.DEACTIVATE });
        return of({ type: QuestionsActionTypes.CREATE_ERROR });
      })
    )
  );
```

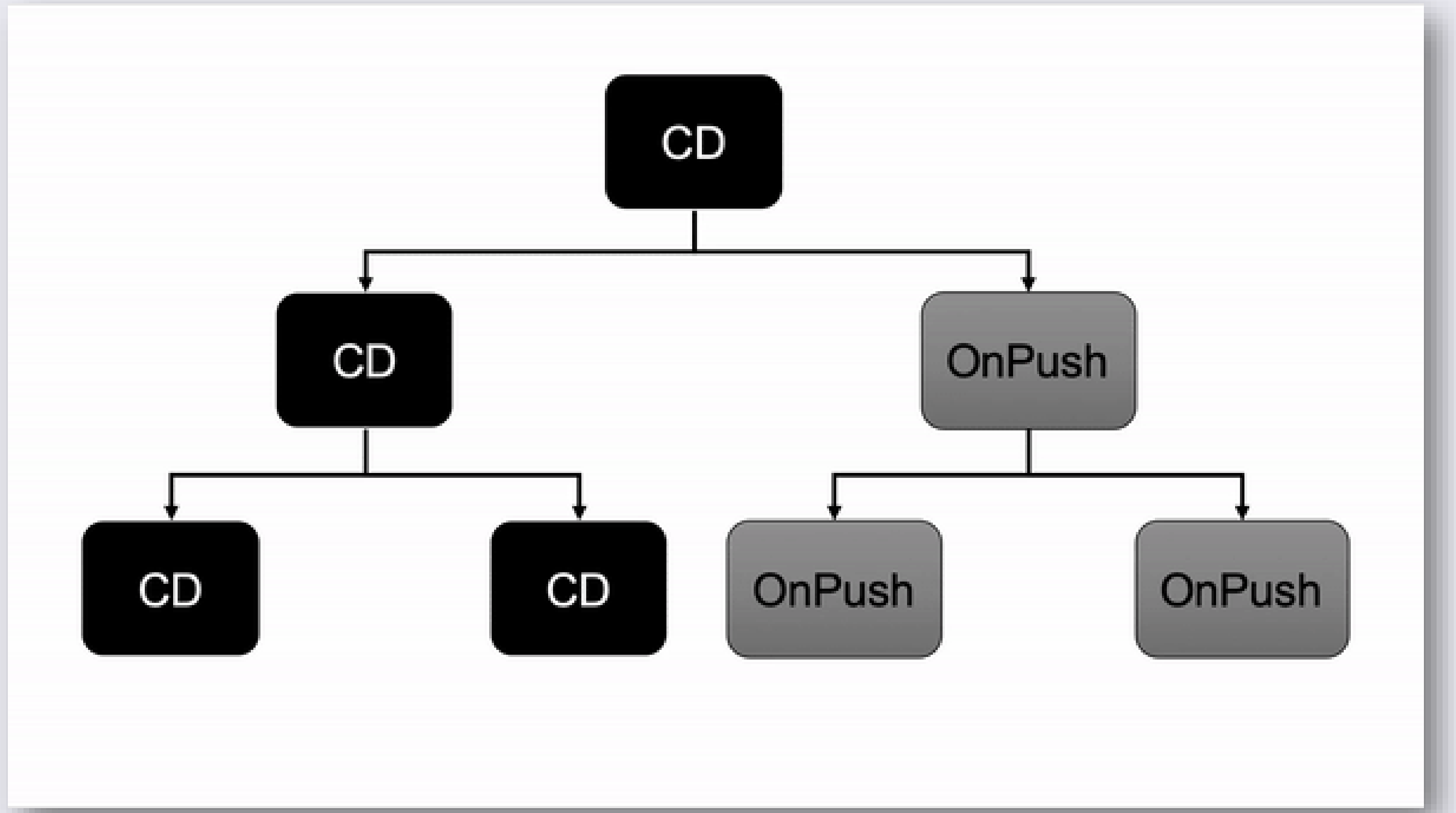


Container/Presentational Components



Estrategia de detección de cambios OnPush



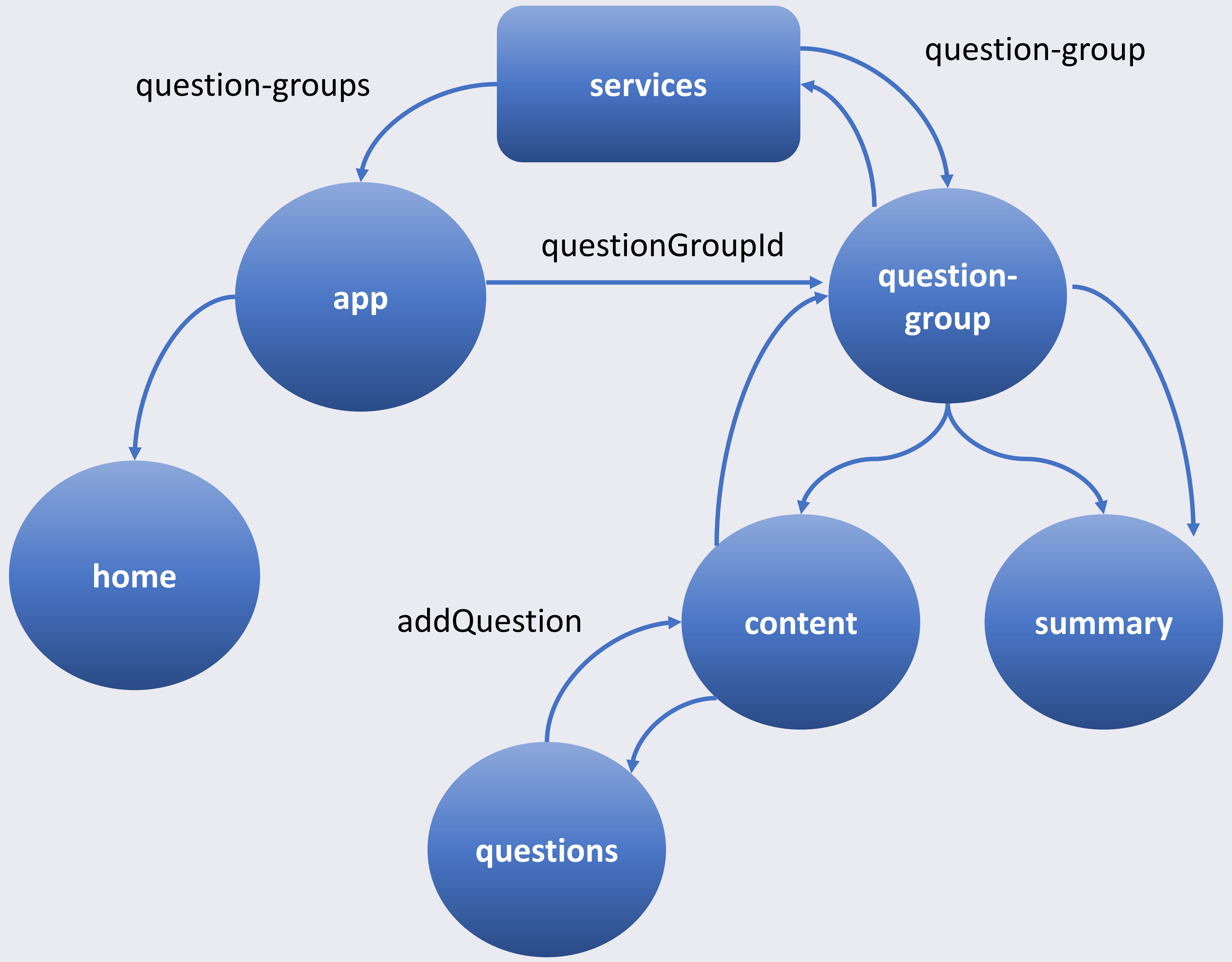


DotNet2020

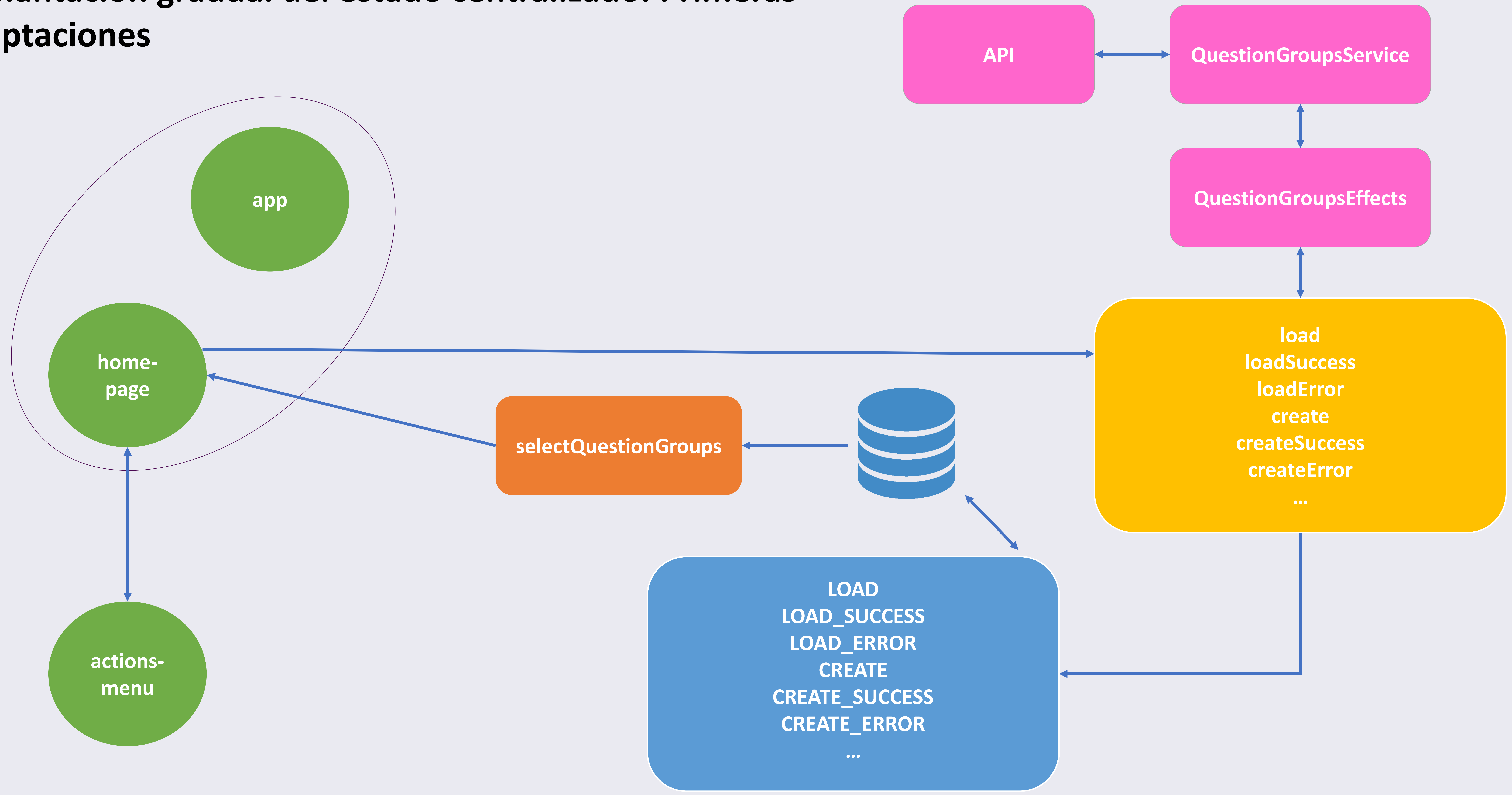
#DotNet2020

Tomando el control

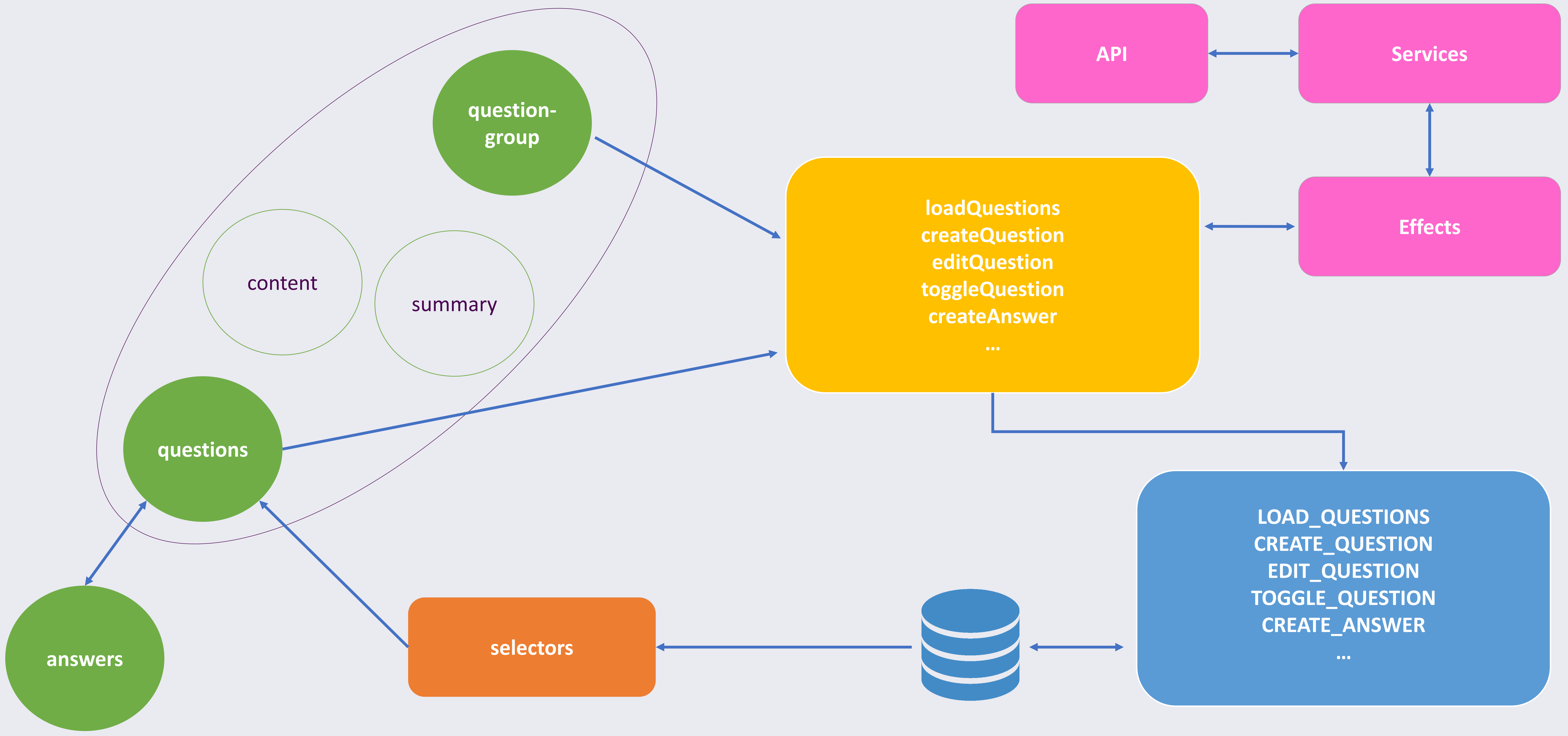
Implantación gradual del estado centralizado: Escenario inicial



Implantación gradual del estado centralizado: Primeras adaptaciones



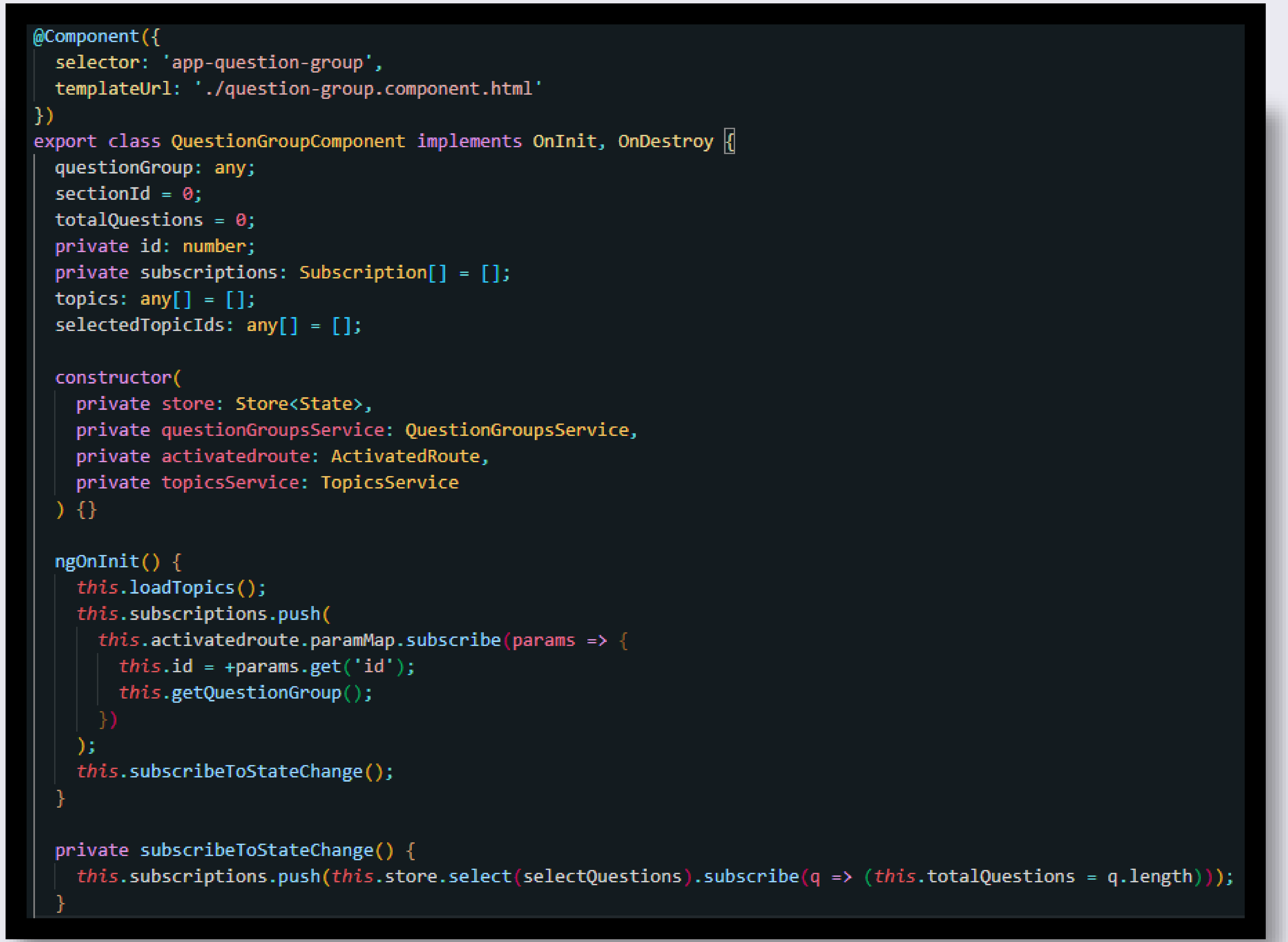
Implantación gradual del estado centralizado: Parte Core



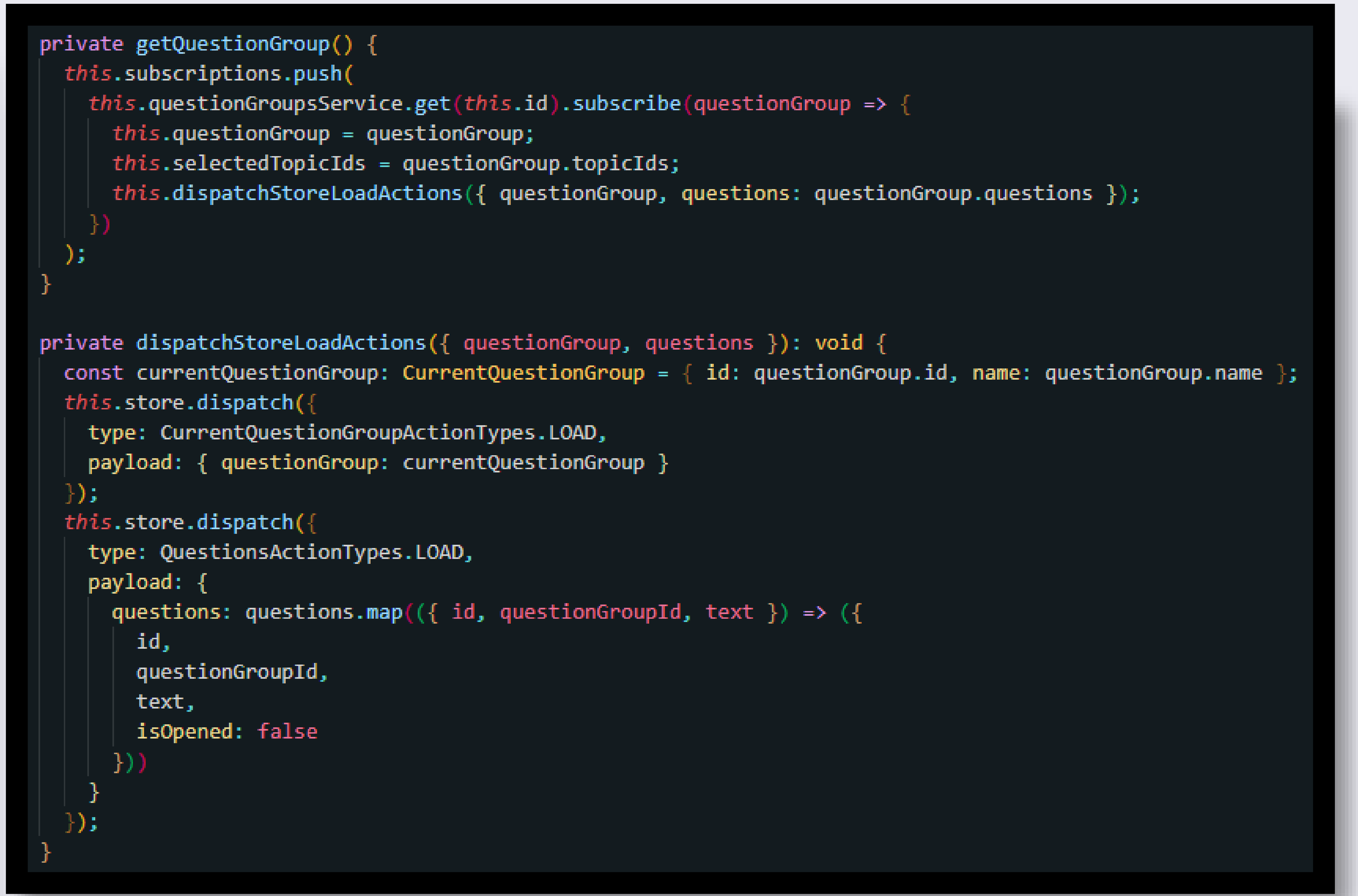
DotNet2020

#DotNet2020

Carga inicial de datos desde el componente (QuestionGroup ViewModel)



```typescript
@Component({
  selector: 'app-question-group',
  templateUrl: './question-group.component.html'
})
export class QuestionGroupComponent implements OnInit, OnDestroy {
  questionGroup: any;
  sectionId = 0;
  totalQuestions = 0;
  private id: number;
  private subscriptions: Subscription[] = [];
  topics: any[] = [];
  selectedTopicIds: any[] = [];

  constructor(
    private store: Store<State>,
    private questionGroupsService: QuestionGroupsService,
    private activatedroute: ActivatedRoute,
    private topicsService: TopicsService
  ) {}

  ngOnInit() {
    this.loadTopics();
    this.subscriptions.push(
      this.activatedroute.paramMap.subscribe(params => {
        this.id = +params.get('id');
        this.getQuestionGroup();
      })
    );
    this.subscribeToStateChange();
  }

  private subscribeToStateChange() {
    this.subscriptions.push(this.store.select(selectQuestions).subscribe(q => (this.totalQuestions = q.length)));
  }
```



```typescript
private getQuestionGroup() {
  this.subscriptions.push(
    this.questionGroupsService.get(this.id).subscribe(questionGroup => {
      this.questionGroup = questionGroup;
      this.selectedTopicIds = questionGroup.topicIds;
      this.dispatchStoreLoadActions({ questionGroup, questions: questionGroup.questions });
    })
  );
}

private dispatchStoreLoadActions({ questionGroup, questions }): void {
  const currentQuestionGroup: CurrentQuestionGroup = { id: questionGroup.id, name: questionGroup.name };
  this.store.dispatch({
    type: CurrentQuestionGroupActionTypes.LOAD,
    payload: { questionGroup: currentQuestionGroup }
  });
  this.store.dispatch({
    type: QuestionsActionTypes.LOAD,
    payload: {
      questions: questions.map(({ id, questionGroupId, text }) => ({
        id,
        questionGroupId,
        text,
        isOpened: false
      }))
    }
  });
}
```

Suscripción y despacho de acciones (Questions ViewModel)

```
// projects\with-ngrx\src\app\components\question-group\content\questions\questions.component.ts
@Component({
  selector: 'app-questions',
  templateUrl: './questions.component.html',
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class QuestionsComponent {
  @Output() messageEventFromQuestions = new EventEmitter<{ action: string; value: any }>();

  // TODO: Pass the store dependency to the upper level components and transform questions and answers into
  // input properties.
  questions$: Observable<Question[]> = this.store.select(selectQuestions);
  answers$: Observable<Answer[]> = this.store.select(selectAnswers);

  questionToAddText = '';

  constructor(private store: Store<State>) {}

  toggleQuestion(questionId: number) {
    this.store.dispatch({ type: QuestionsActionTypes.TOGGLE, payload: { questionId } });
    this.messageEventFromQuestions.emit({ action: 'question-toggle', value: { questionId } });
  }
}
```

Suscripción (Questions View)

```
// projects\with-ngrx\src\app\components\question-group\content\questions\questions.component.html
<div class="flex-wrap mb-4 items-center lg:w-3/4 lg:max-w-3xl" *ngFor="let item of questions$ | async">
  <div class="flex">
    <input
      type="text"
      value="{{ item.text }}"
      id="question-{{ item.id }}"
      class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight"
      (focusout)="updateQuestion(13, item, $event.target.value)"
      (keyup)="updateQuestion($event.keyCode, item, $event.target.value)"
    />
    <button
      id="toggle-question-{{ item.id }}"
      class="flex-no-shrink p-2 ml-2 border-2 rounded border-teal-200 hover:text-white hover:bg-teal-500"
      (click)="toggleQuestion(item.id)"
    >
      {{ item.isOpened ? 'Collapse' : 'Expand' }}
    </button>
  </div>
</div>
```

Descarga de lógica y responsabilidades (QuestionGroup ViewModel)

```
// projects\with-ngrx\src\app\components\question-group\question-group.component.ts
receiveMessageFromContent(event: { action: string; value: any }) {
  // TODO: Once the refactor to ngrx will be finished, this component will manage the events sent by the
  // child components and invoke the store actions.
  // The difference with the original code is all the logic will be encapsulated in the store,
  // having softer components and more testable code.
  switch (event.action) {
    case 'content-question-add':
      // this.store.dispatch({ type: QuestionsActionTypes.CREATE, payload: { text: event.value.text } });
      break;
    case 'content-question-update':
      // this.store.dispatch({ type: QuestionsActionTypes.EDIT, payload: { question: event.value.question } });
      break;
    case 'content-question-delete':
      // this.store.dispatch({ type: QuestionsActionTypes.DELETE, payload: { questionId: event.value.questionId } });
      break;
    case 'content-question-toggle':
      // this.store.dispatch({ type: QuestionsActionTypes.TOGGLE, payload: { questionId: event.value.questionId } });
      break;
    case 'content-answer-add':
      // this.store.dispatch({ type: AnswersActionTypes.CREATE, payload: { answer: event.value.answer } });
      break;
    case 'content-answer-update':
      // this.store.dispatch({ type: AnswersActionTypes.EDIT, payload: { answer: event.value.answer } });
      break;
```

QuestionGroup ViewModel (antes)

```
// projects\original\src\app\components\question-group\question-group.component.ts
export class QuestionGroupComponent implements OnChanges, OnDestroy {
  receiveMessageFromContent(data: any) {
    const { action, value } = data;
    switch (action) {
      // Hace de todo, lo mismo te añade una pregunta que te frie un huevo
      case "content-question-add":
        // [TL;DR]
        break;
      case "content-question-update":
       // [TL;DR]
        break;
      case "content-question-toggle":
        // [TL;DR]
        break;
      case "content-answer-add":
        // [TL;DR]
        break;
      case "content-topic-select":
        // [TL;DR]
        break;
    }
  }
  // Referencias a todos los servicios de la aplicación
  // Más de 600 líneas de código
```

Manejo de side effects

```
// projects\with-ngrx\src\app\state\questions\effects\questions-effects.service.ts
toggle$ = createEffect(() =>
  this.actions$.pipe(
    ofType(toggle),
    concatMap(action =>
      of(action).pipe(
        withLatestFrom(this.store.select(fromQuestions.selectQuestionById, { id: action.payload.questionId }))
      )
    ),
    switchMap(([action, existingQuestion]) => {
      if (existingQuestion.isOpened) {
        this.store.dispatch({ type: LoaderActionTypes.DEACTIVATE });
        return of(toggleSuccess({ payload: { questionId: action.payload.questionId } }));
      }

      return forkJoin([this.answersService.getAll(existingQuestion.questionGroupId, action.payload.questionId)]).pipe(
        switchMap(([answers]) => {
          return [
            loadAnswers({
              payload: {
                answers: answers.map(({ id, questionGroupId, questionId, text }) => ({
                  id, questionGroupId, questionId, text
                }))
              }
            }),  toggleSuccess({ payload: { questionId: action.payload.questionId } })
          ];
        })
      );
```

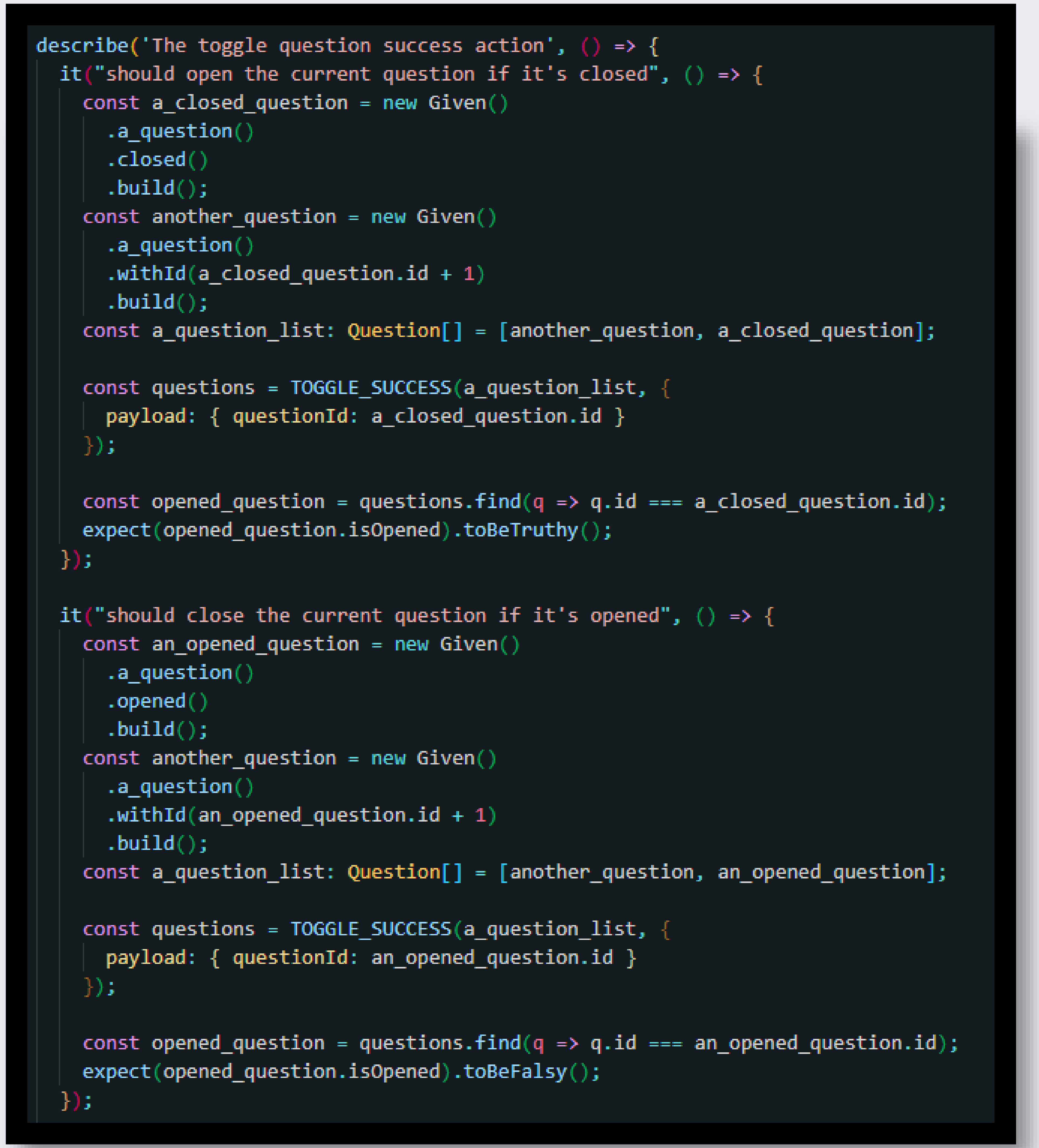
Lógica mezclada entre componentes

```
// projects\original\src\app\components\question-group\question-group.component.ts
case "content-question-toggle":
        if (value === this.openedQuestionId) {
          this.openedQuestionId = null;
        } else {
          this.openedQuestionId = value;
        }
        this.getAnswers(value).then(() => this.getQuestionGroup());
        break;

private getQuestionGroup() {
    return this.questionGroupsService
      .get(+this.id)
      .toPromise()
      .then(data => {
        this.questionGroup = data;
        this.questions = this.questionGroup.questions.map(q => ({
          ...q,
          isOpened: q.id === +this.openedQuestionId
        }));
        this.selectedTopicIds = this.questionGroup.topicIds;
        this.numberOfQuestions = this.questionGroup.questions.length;
        this.ref.detectChanges();
      });
  }
```

Lógica encapsulada en reducers (testable)

```
// projects\with-ngrx\src\app\state\questions\reducers\toggle\toggle.ts
export function TOGGLE(state: Question[]): Question[] {
  return [...state];
}

export function TOGGLE_SUCCESS(state: Question[], action: { payload: { questionId: number } }): Question[] {
  const newState = closeOtherQuestions([...state], action.payload.questionId);
  const questionToOpenIndex = newState.findIndex(q => q.id === action.payload.questionId);
  const questionToOpen = newState[questionToOpenIndex];
  const questionOpened = {
    ...questionToOpen,
    isOpened: !questionToOpen.isOpened
  };
  newState[questionToOpenIndex] = questionOpened;
  return newState;
}

function closeOtherQuestions(questions: Question[], questionId: number): Question[] {
  return questions.map(q => {
    if (q.id === questionId) {
      return { ...q };
    }
    return { ...q, isOpened: false };
  });
}

export function TOGGLE_ERROR(state: Question[]): Question[] {
  return [...state];
```

Definiendo la estrategia de testing: Reducers

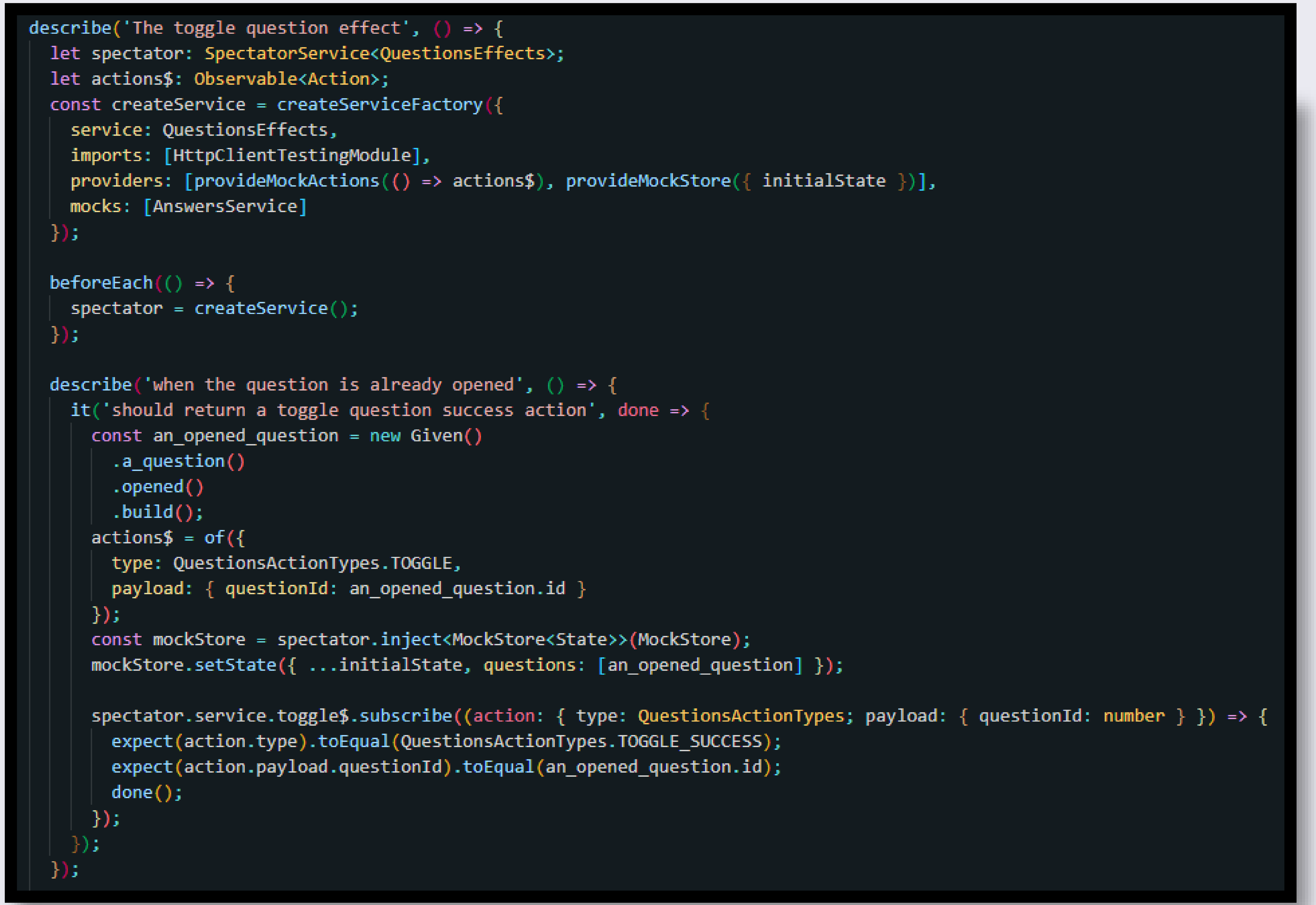


```
describe('The toggle question success action', () => {
  it("should open the current question if it's closed", () => {
    const a_closed_question = new Given()
      .a_question()
      .closed()
      .build();
    const another_question = new Given()
      .a_question()
      .withId(a_closed_question.id + 1)
      .build();
    const a_question_list: Question[] = [another_question, a_closed_question];

    const questions = TOGGLE_SUCCESS(a_question_list, {
      payload: { questionId: a_closed_question.id }
    });

    const opened_question = questions.find(q => q.id === a_closed_question.id);
    expect(opened_question.isOpened).toBeTruthy();
  });

  it("should close the current question if it's opened", () => {
    const an_opened_question = new Given()
      .a_question()
      .opened()
      .build();
    const another_question = new Given()
      .a_question()
      .withId(an_opened_question.id + 1)
      .build();
    const a_question_list: Question[] = [another_question, an_opened_question];

    const questions = TOGGLE_SUCCESS(a_question_list, {
      payload: { questionId: an_opened_question.id }
    });

    const opened_question = questions.find(q => q.id === an_opened_question.id);
    expect(opened_question.isOpened).toBeFalsy();
  });
```

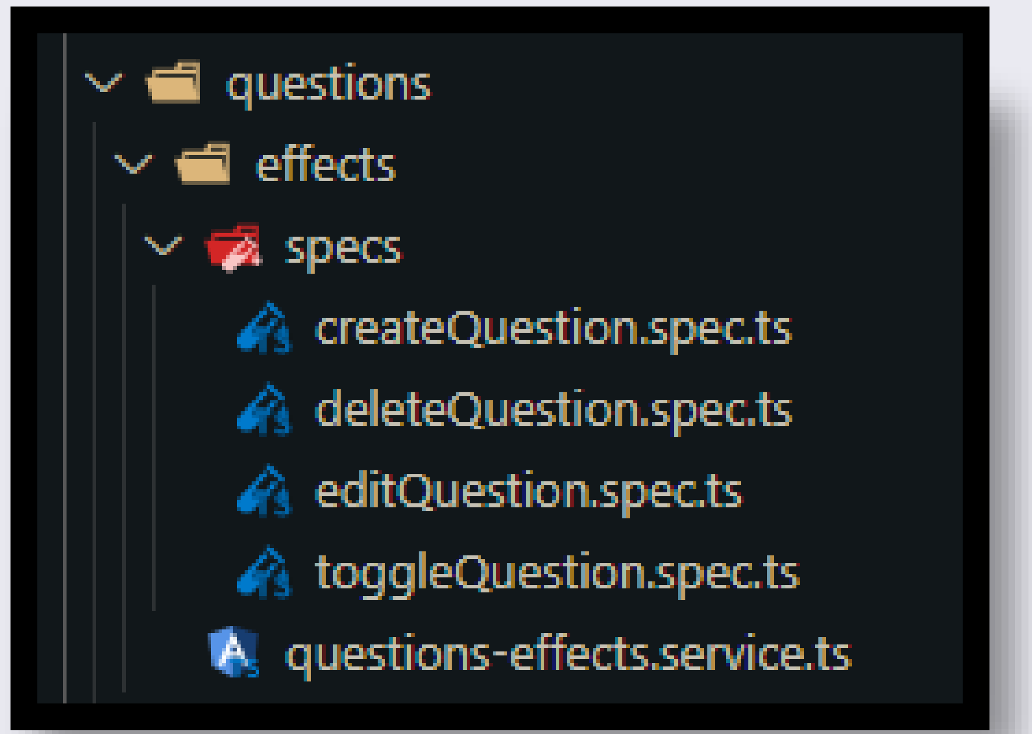
DotNet2020

#DotNet2020

Definiendo la estrategia de testing: Effects

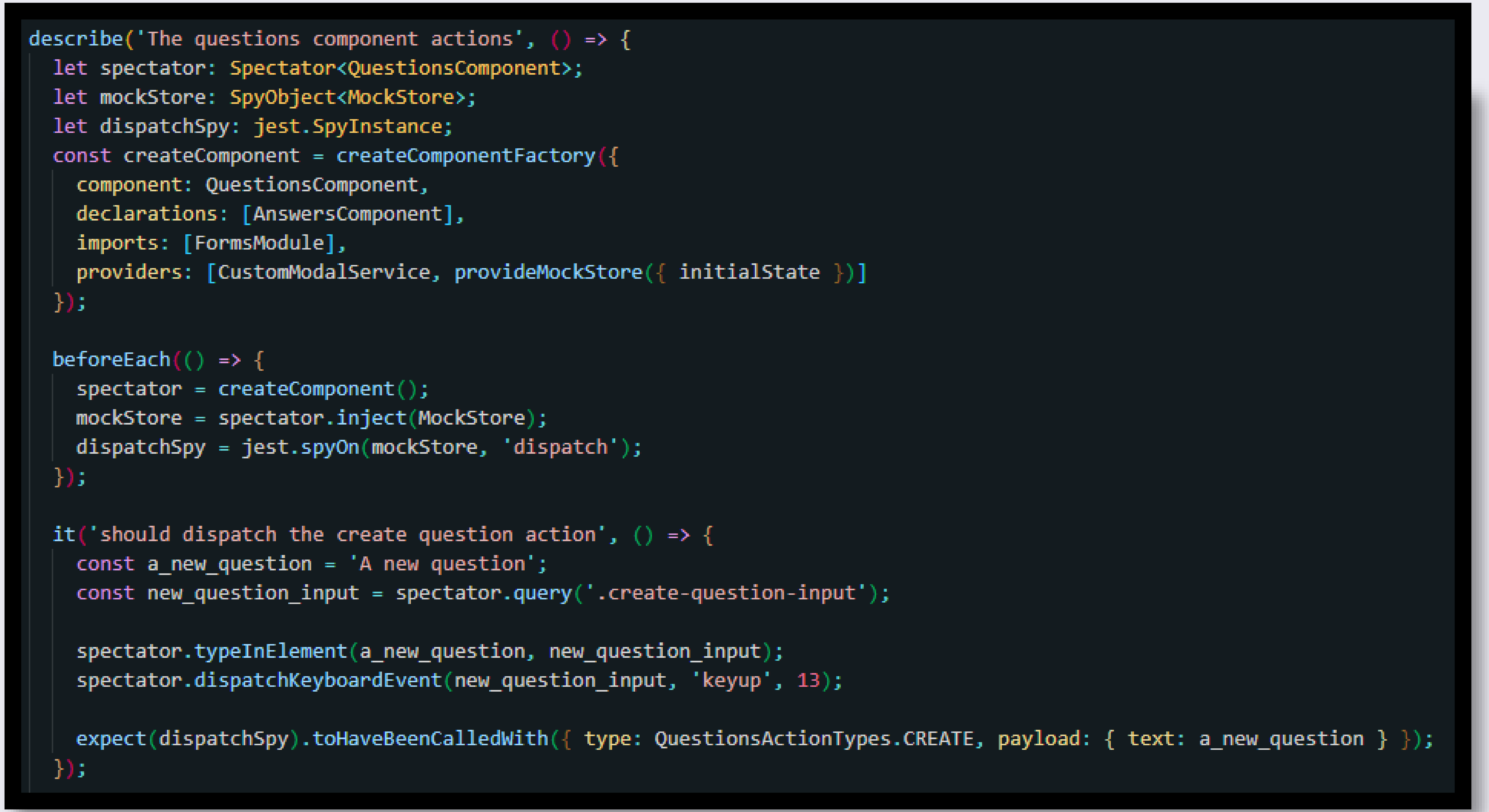


```
describe('The toggle question effect', () => {
  let spectator: SpectatorService<QuestionsEffects>;
  let actions$: Observable<Action>;
  const createService = createServiceFactory({
    service: QuestionsEffects,
    imports: [HttpClientTestingModule],
    providers: [provideMockActions(() => actions$), provideMockStore({ initialState })],
    mocks: [AnswersService]
  });

  beforeEach(() => {
    spectator = createService();
  });

  describe('when the question is already opened', () => {
    it('should return a toggle question success action', done => {
      const an_opened_question = new Given()
        .a_question()
        .opened()
        .build();
      actions$ = of({
        type: QuestionsActionTypes.TOGGLE,
        payload: { questionId: an_opened_question.id }
      });
      const mockStore = spectator.inject<MockStore<State>>(MockStore);
      mockStore.setState({ ...initialState, questions: [an_opened_question] });

      spectator.service.toggle$.subscribe((action: { type: QuestionsActionTypes; payload: { questionId: number } }) => {
        expect(action.type).toEqual(QuestionsActionTypes.TOGGLE_SUCCESS);
        expect(action.payload.questionId).toEqual(an_opened_question.id);
        done();
      });
    });
  });
});
```



```
questions
  effects
    specs
      createQuestion.spec.ts
      deleteQuestion.spec.ts
      editQuestion.spec.ts
      toggleQuestion.spec.ts
    questions-effects.service.ts
```

Definiendo la estrategia de testing: Componentes



```
describe('The questions component actions', () => {
  let spectator: Spectator<QuestionsComponent>;
  let mockStore: SpyObject<MockStore>;
  let dispatchSpy: jest.SpyInstance;
  const createComponent = createComponentFactory({
    component: QuestionsComponent,
    declarations: [AnswersComponent],
    imports: [FormsModule],
    providers: [CustomModalService, provideMockStore({ initialState })]
  });

  beforeEach(() => {
    spectator = createComponent();
    mockStore = spectator.inject(MockStore);
    dispatchSpy = jest.spyOn(mockStore, 'dispatch');
  });

  it('should dispatch the create question action', () => {
    const a_new_question = 'A new question';
    const new_question_input = spectator.query('.create-question-input');

    spectator.typeInElement(a_new_question, new_question_input);
    spectator.dispatchKeyboardEvent(new_question_input, 'keyup', 13);

    expect(dispatchSpy).toHaveBeenCalledWith({ type: QuestionsActionTypes.CREATE, payload: { text: a_new_question } });
  });
```

Tomando el control: Últimos cambios

- Loading intrusivo.
- Actualización a Angular 9.

Próximos pasos

- Continuar la refactorización.
- + Tipos.
- Features y lazy loading.
- Loadings menos intrusivos.
- @ngrx/entity y @ngrx/data.

Lecciones aprendidas

- Reactividad/RxJS.
- Jest con JSDOM.
- Commitizen, Husky y Git Hooks.
- Prettier.
- Pair programming.
- Deuda técnica.

Referencias

- Questions Manager repository
- State Management: don't shoot yourself in the foot before you start an Angular application
- Angular Architecture - Smart Components vs Presentational Components
- The Last Guide For Angular Change Detection You'll Ever Need
- NgRx Docs
- Jest Docs
- Spectator Docs
- Clean Code: A Handbook of Agile Software Craftsmanship (Robert C. Martin)
- Refactoring: Improving the Design of Existing Code (Martin Fowler)
- Clean JavaScript Book (Miguel A. Gómez)
- Diseño Ágil con TDD (Carlos Blé)

DotNet2020

#DotNet2020

Questions & Answers

DotNet2020

#DotNet2020

Thanks and ...
See you soon!

Thanks also to the sponsors.
Without whom this would not have been posible.

plain concepts

Microsoft

axazure

plasticscm

CEU | Universidad San Pablo