Angular state management

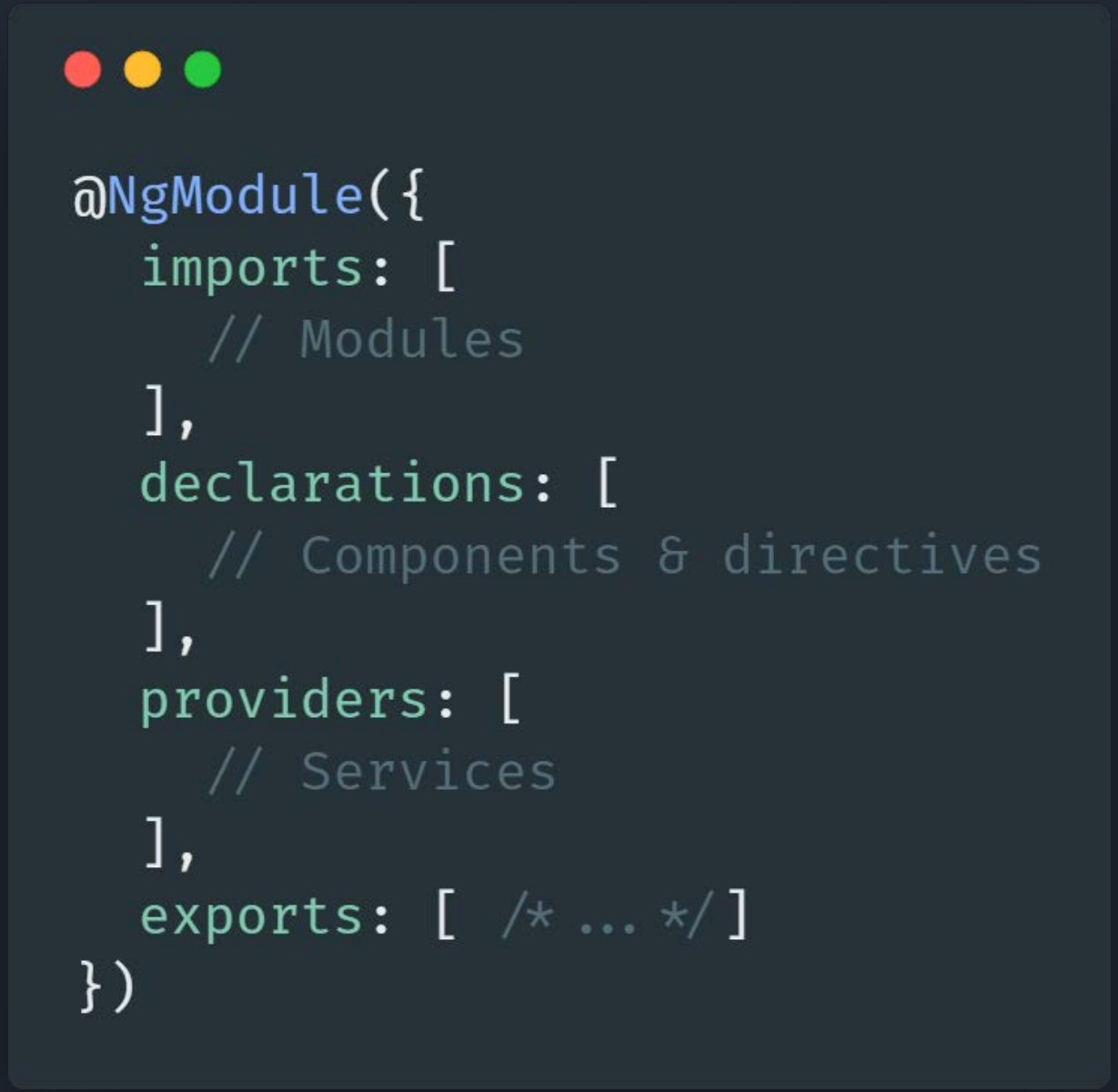
Store & Effect

@FabianGosebrink


Structure
@FabianGosebrink

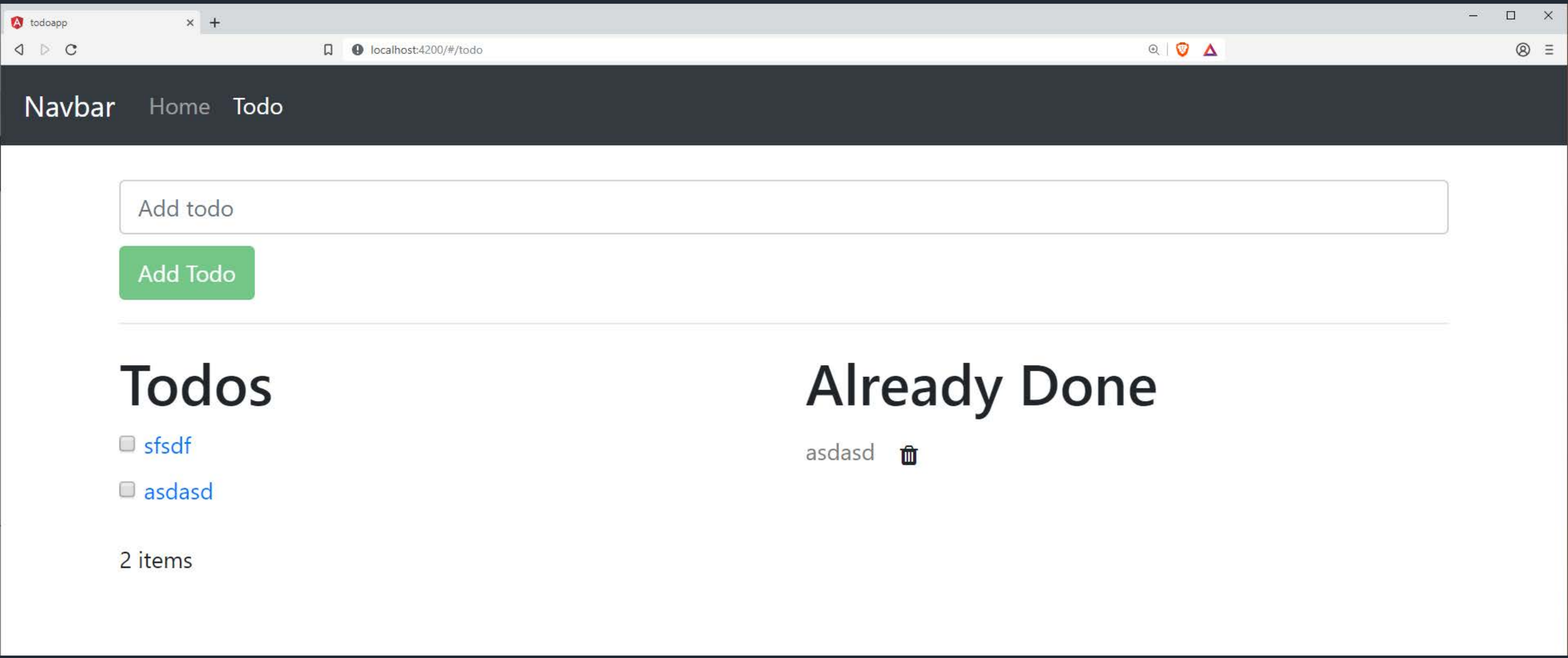
Modules

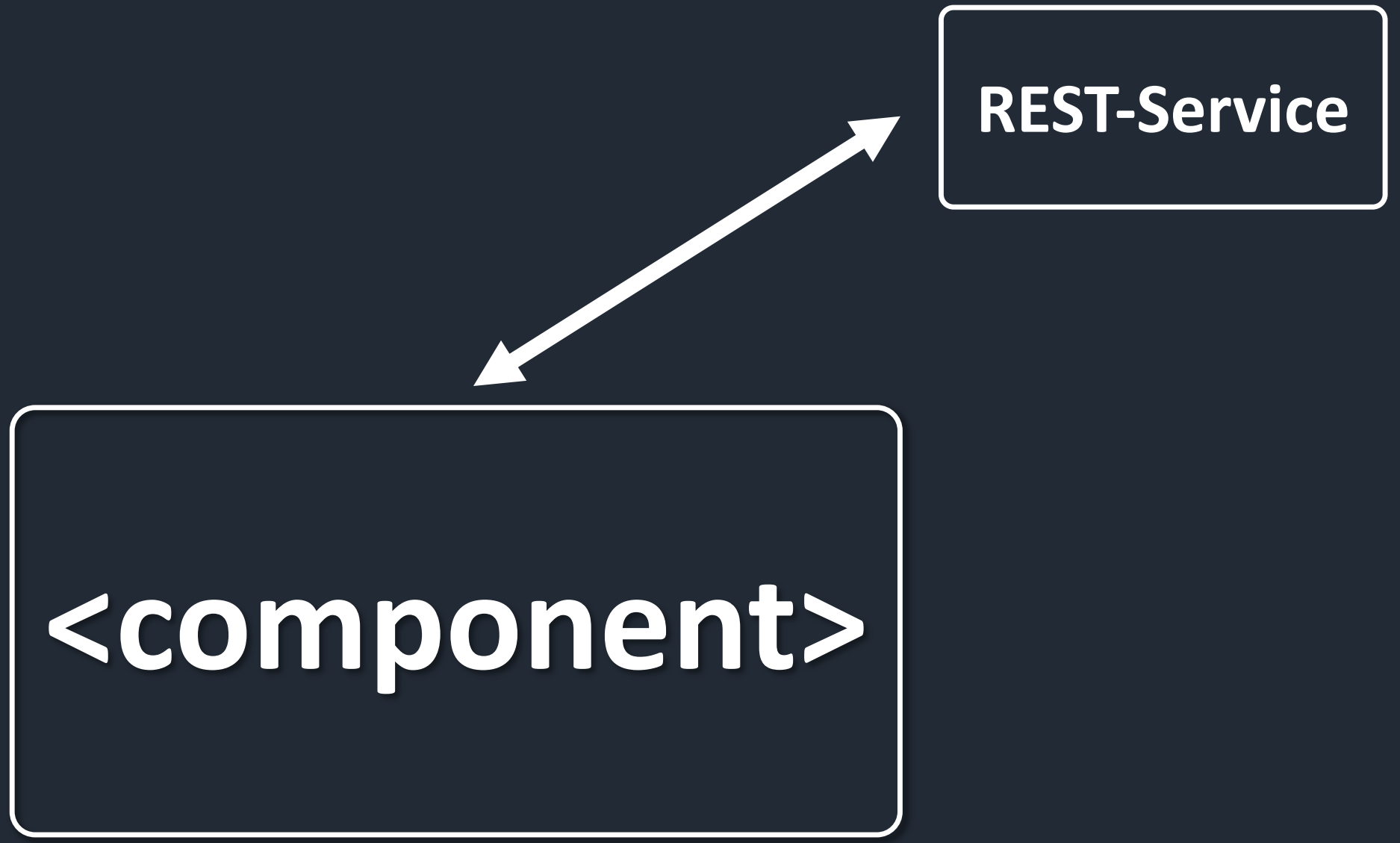
@FabianGosebrink

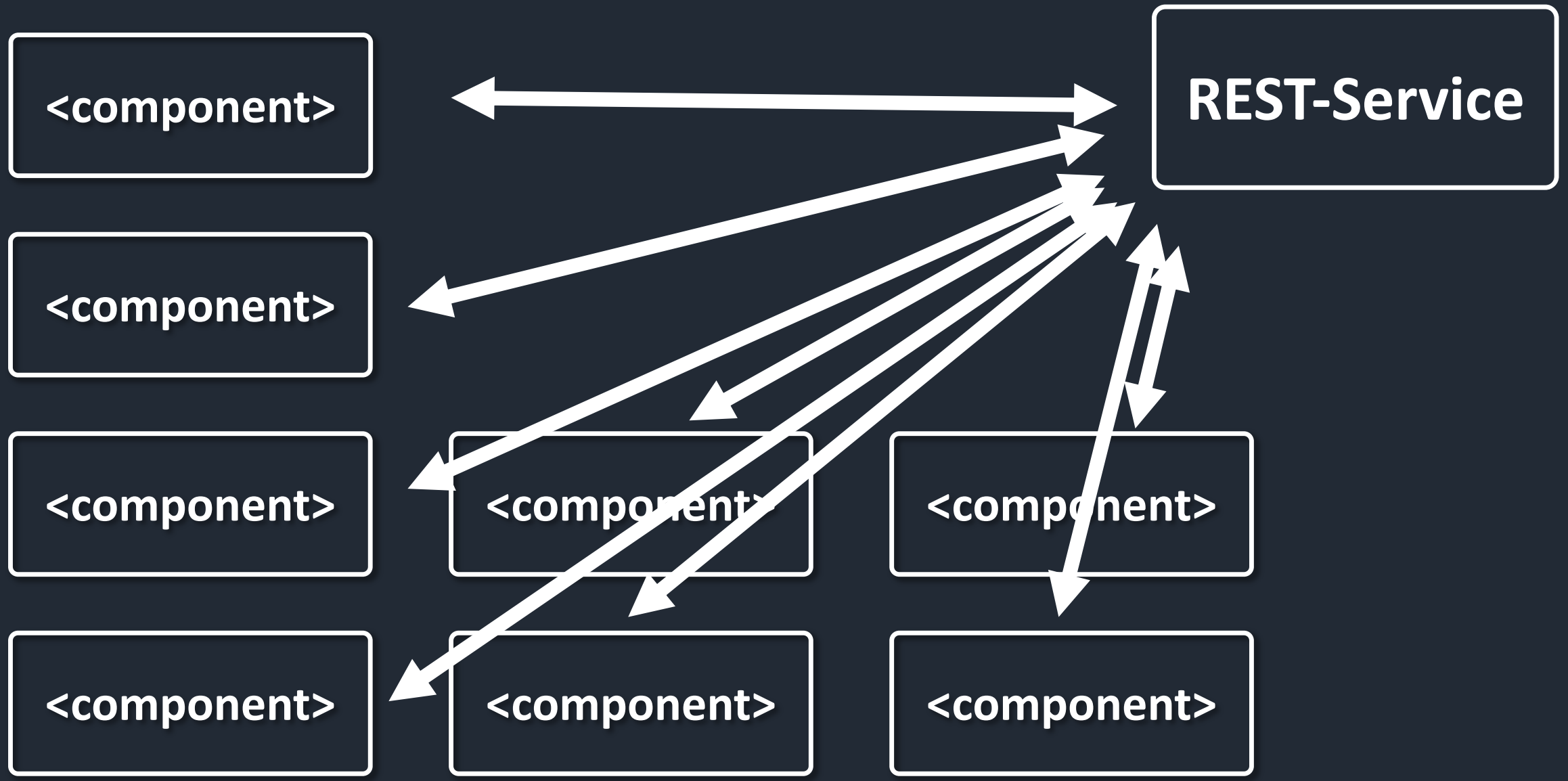


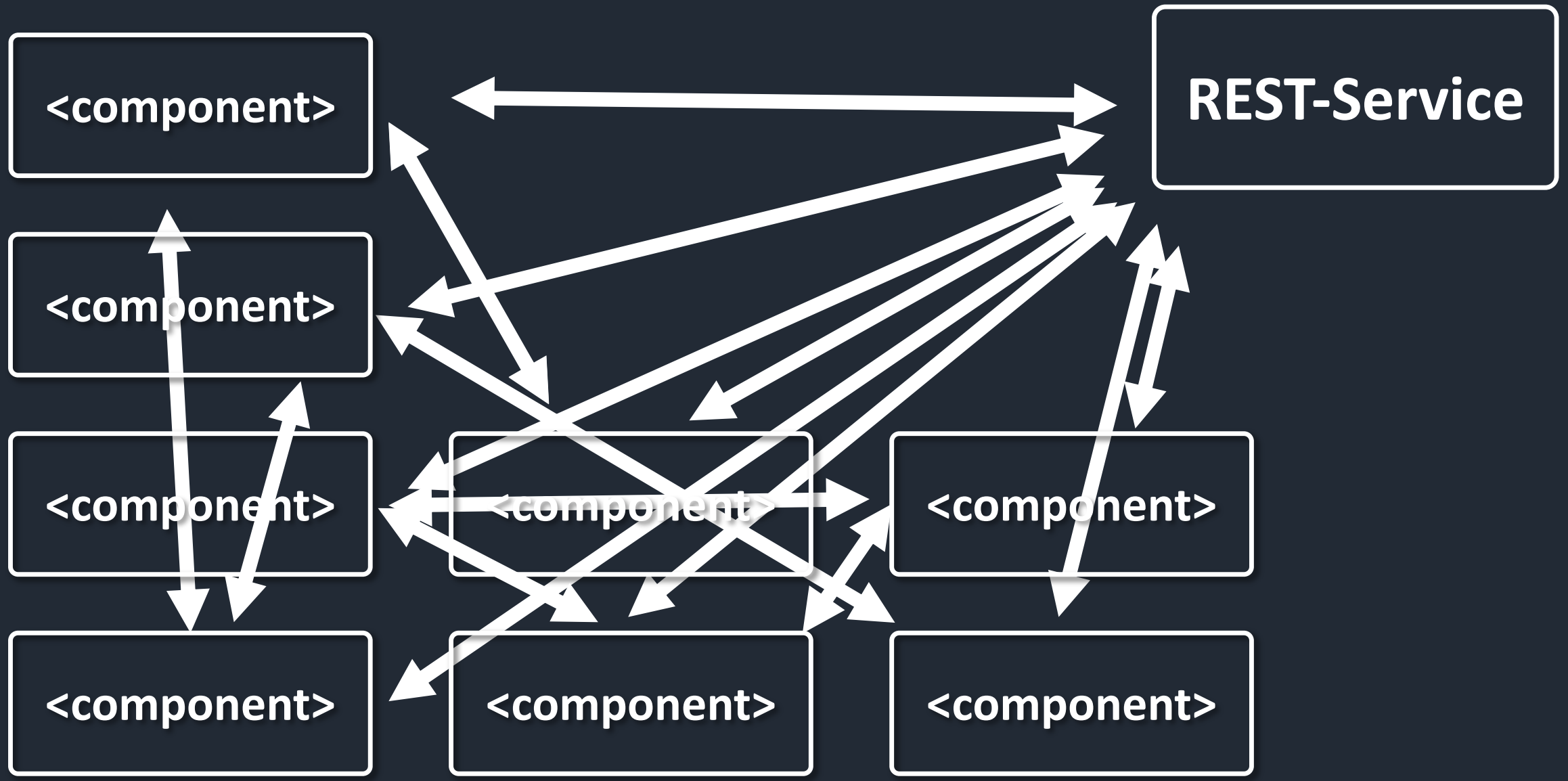
```
@NgModule({
  imports: [
    // Modules
  ],
  declarations: [
    // Components & directives
  ],
  providers: [
    // Services
  ],
  exports: [ /* ... */]
})
```

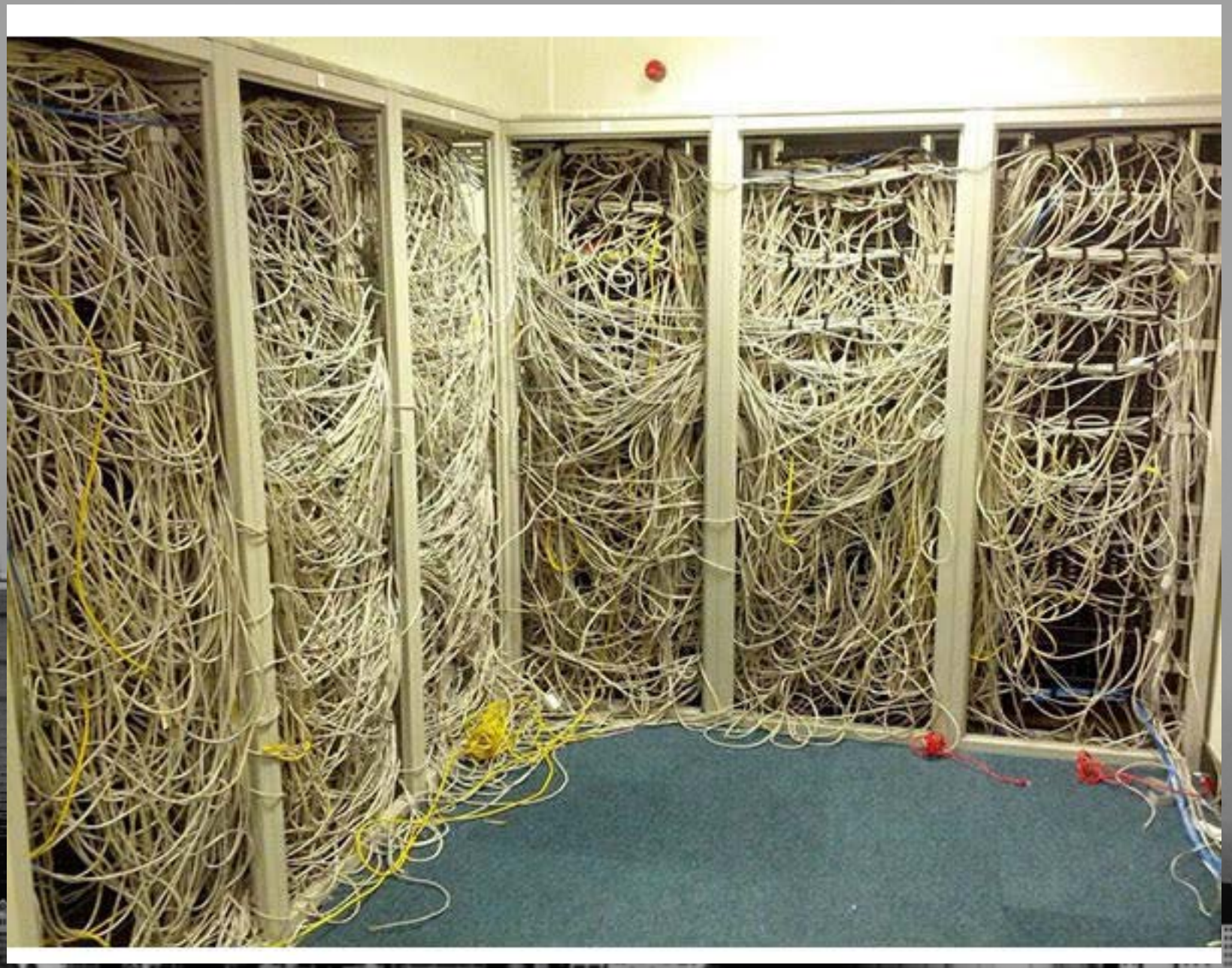

Compone
@FabianGosebrink


todoapp
localhost:4200/#/todo
Navbar
Home
Todo
Add todo
Add Todo
Todos
sfsdf
asdasd
2 items
Already Done
asdasd

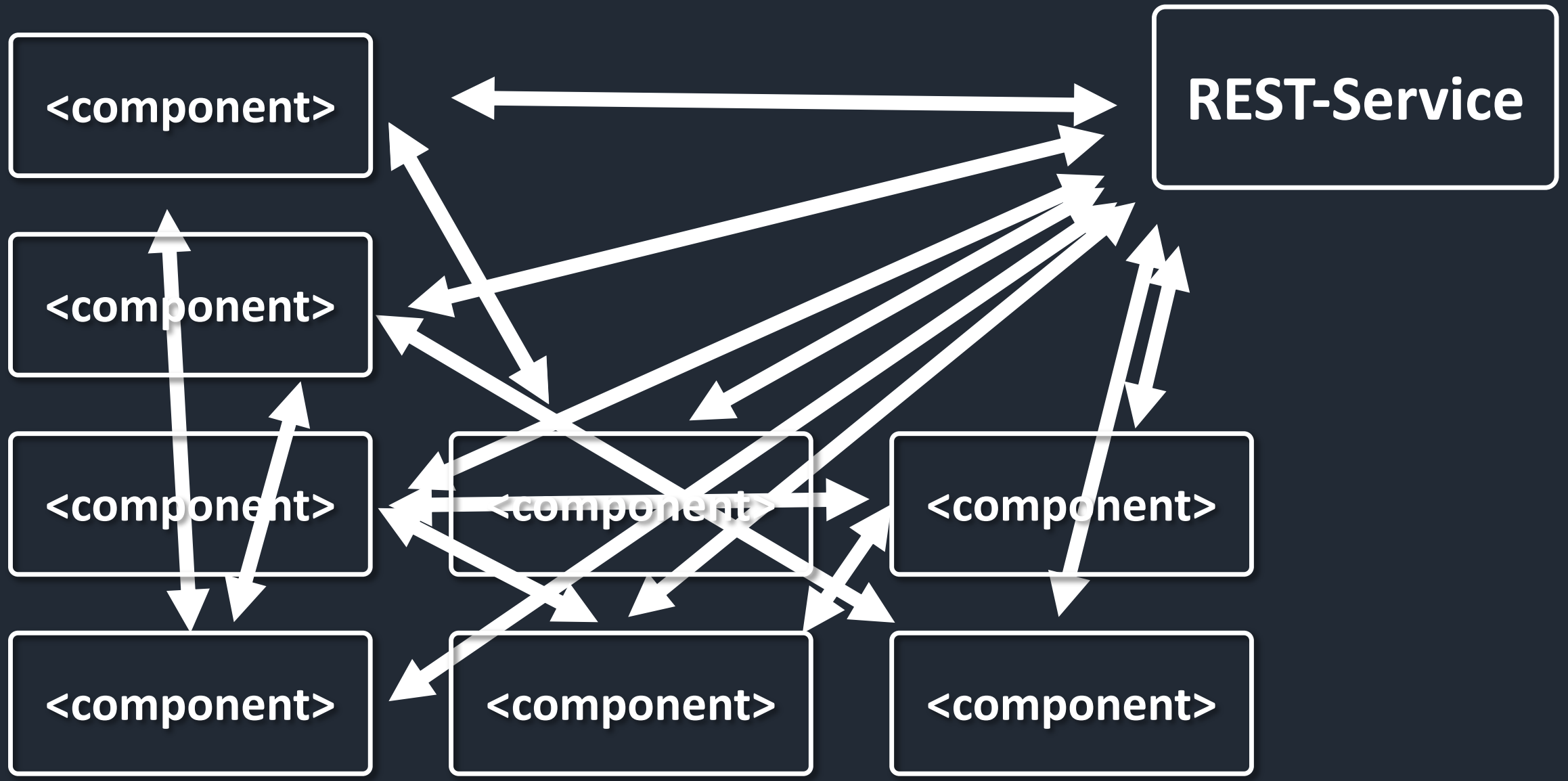

REST-Service
<component>


<component>
<component>
<component>
<component>
<component>
<component>
<component>
REST-Service

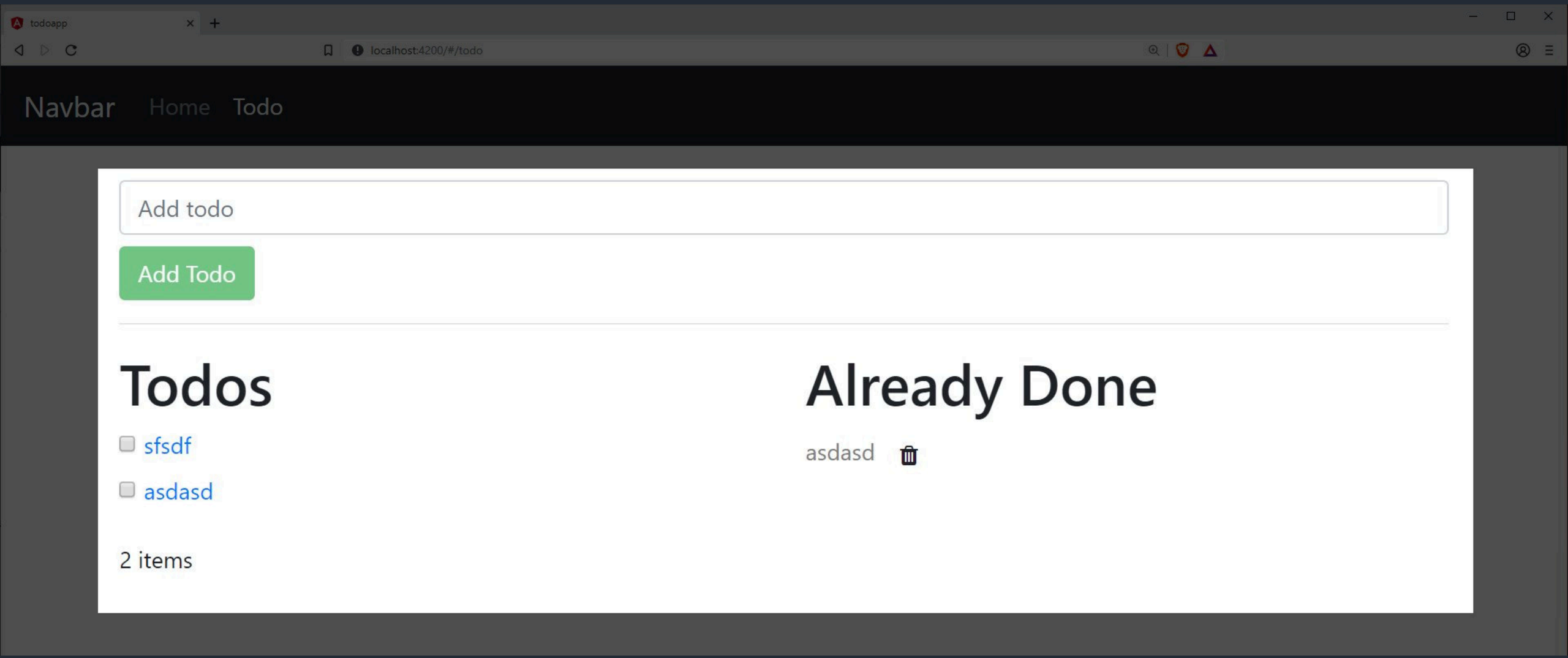

<component>
REST-Service
<component>
<component>
<component>
<component>
<component>
<component>
<component>

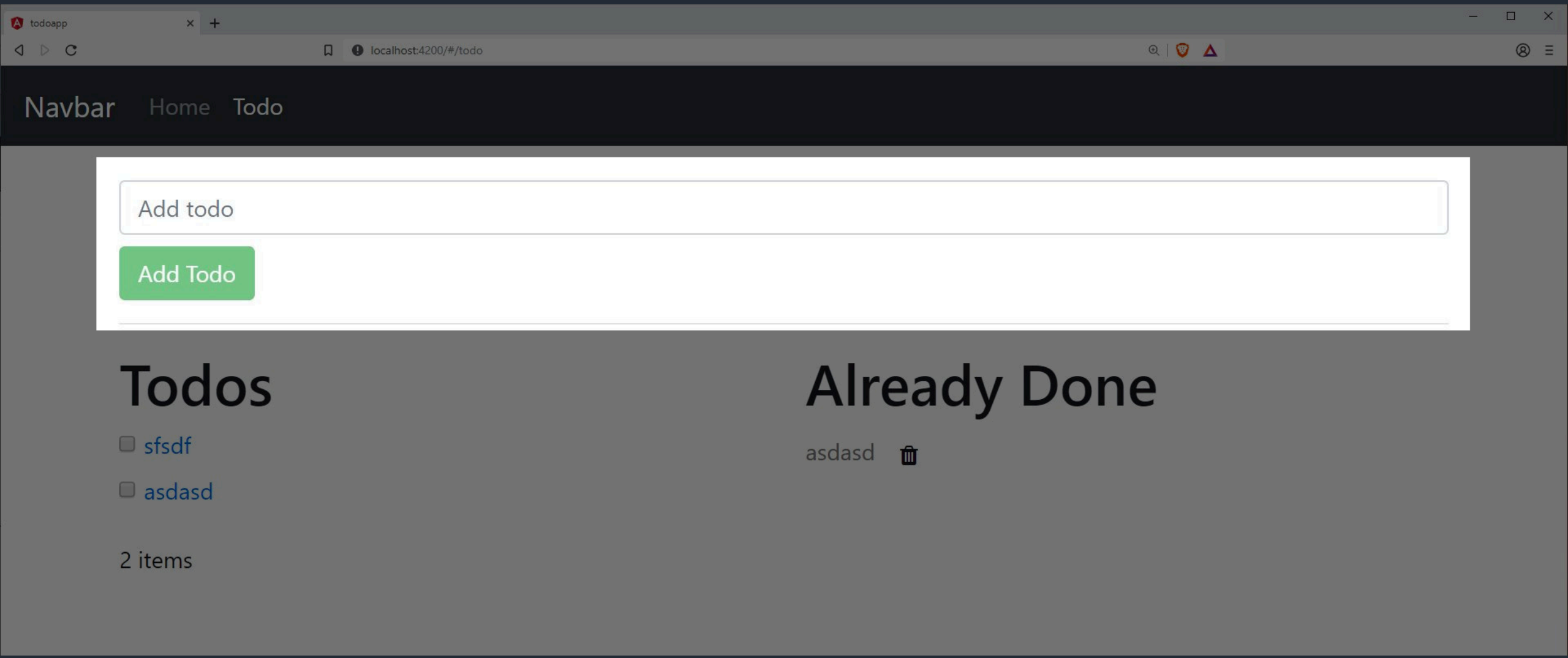


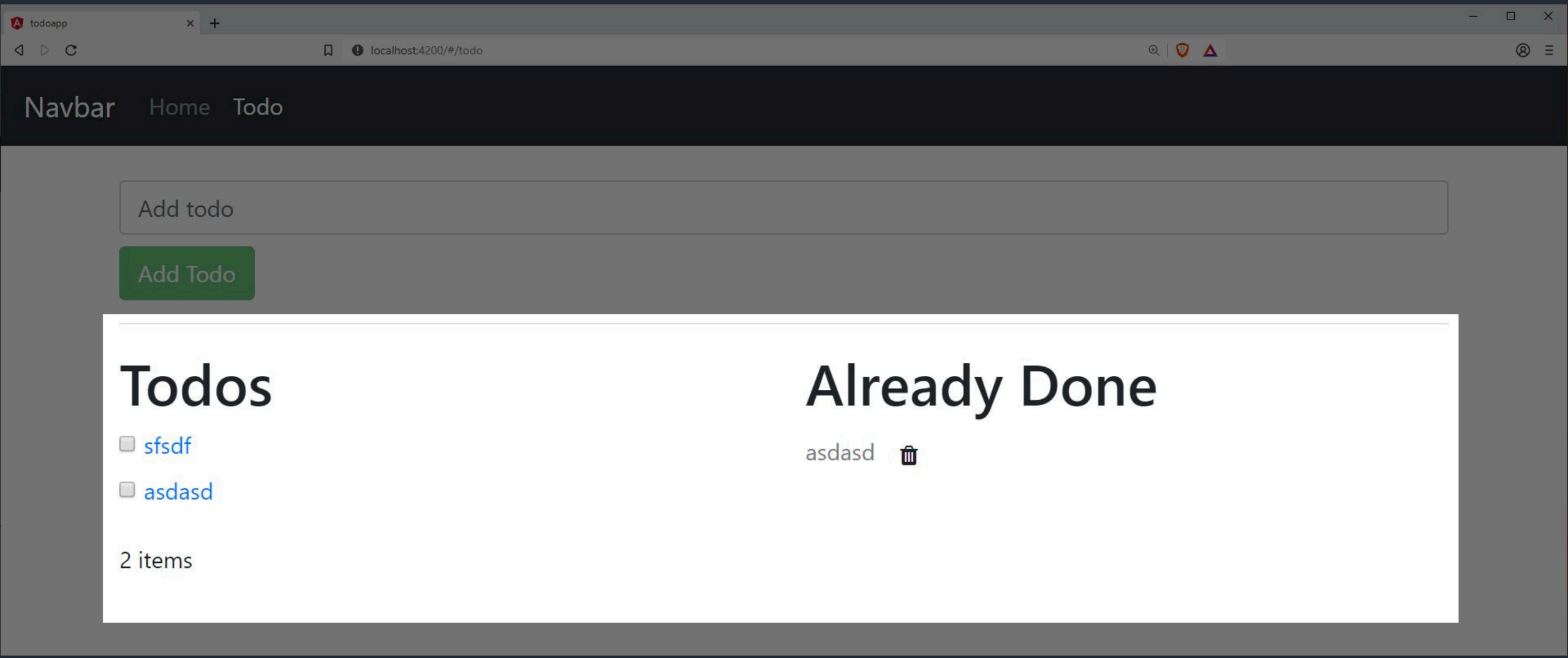
@FabianGosebrink

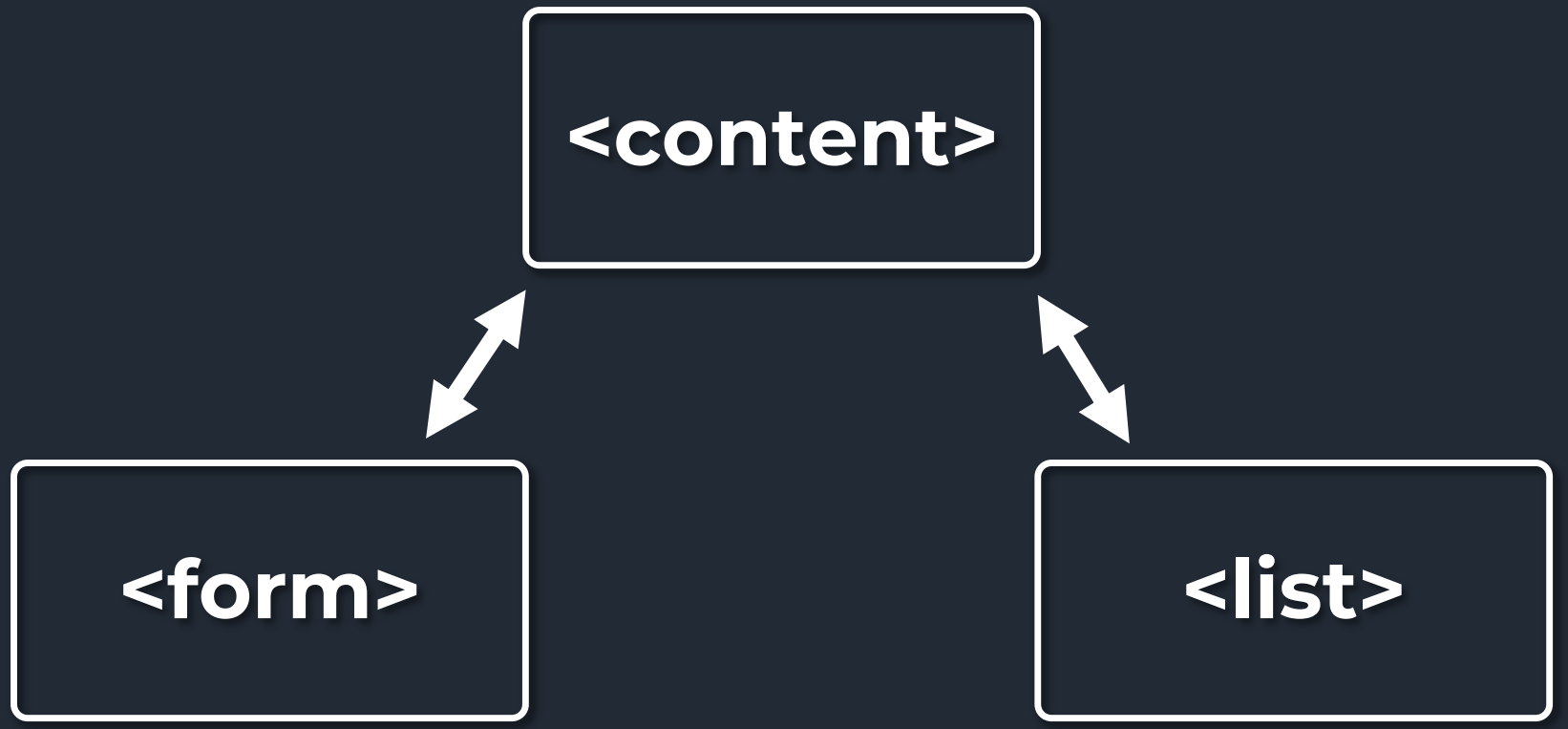

<component>
REST-Service
<component>
<component>
<component>
<component>
<component>
<component>
<component>

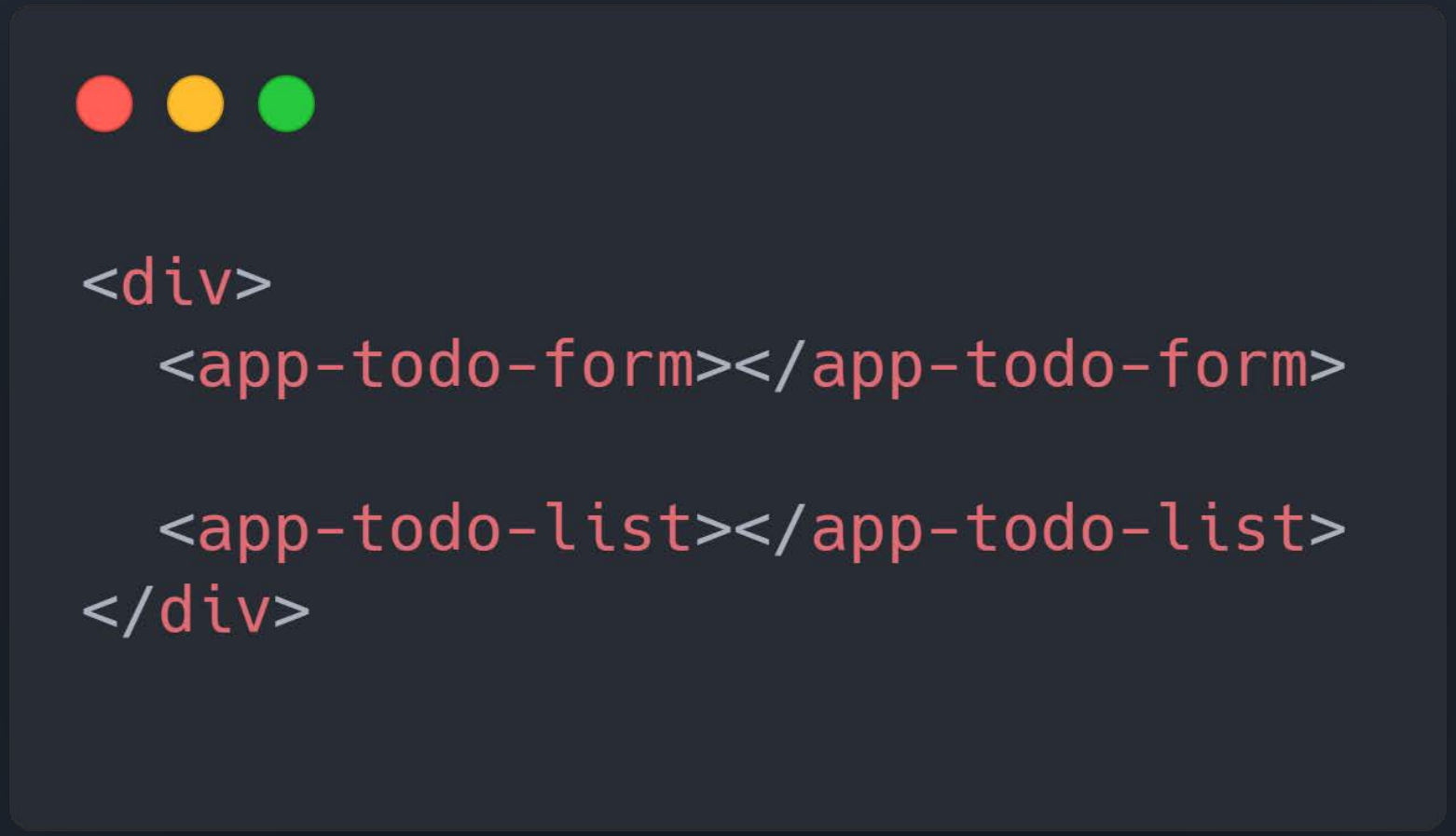

One Way Dataflow
@FabianGosebrink


todoapp
localhost:4200/#/todo
Navbar
Home
Todo
Add todo
Add Todo
Todos
sfsdf
asdasd
2 items
Already Done
asdasd

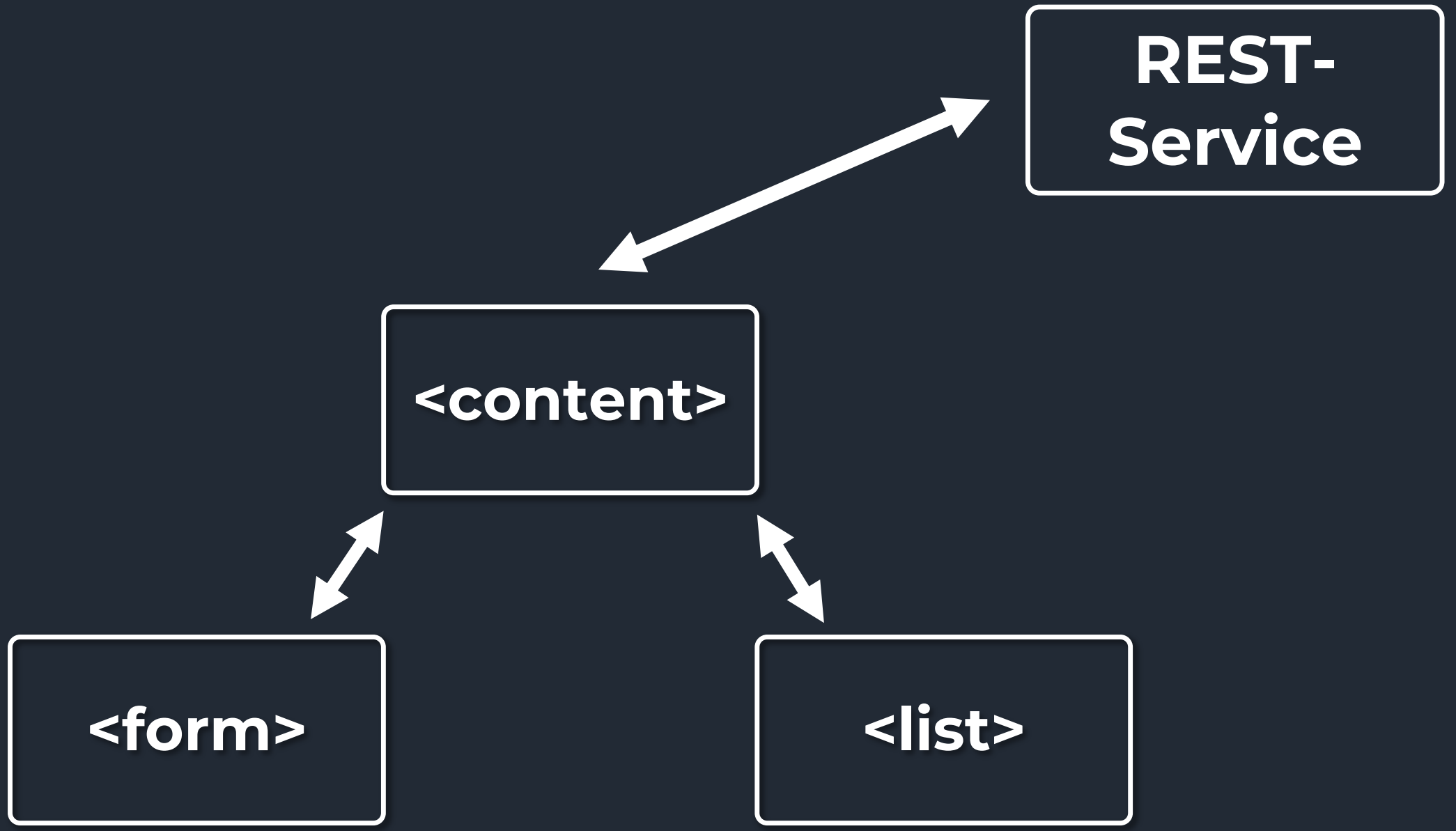

todoapp
localhost:4200/#/todo
Navbar
Home
Todo
Add todo
Add Todo
Todos
sfsdf
asdasd
2 items
Already Done
asdasd

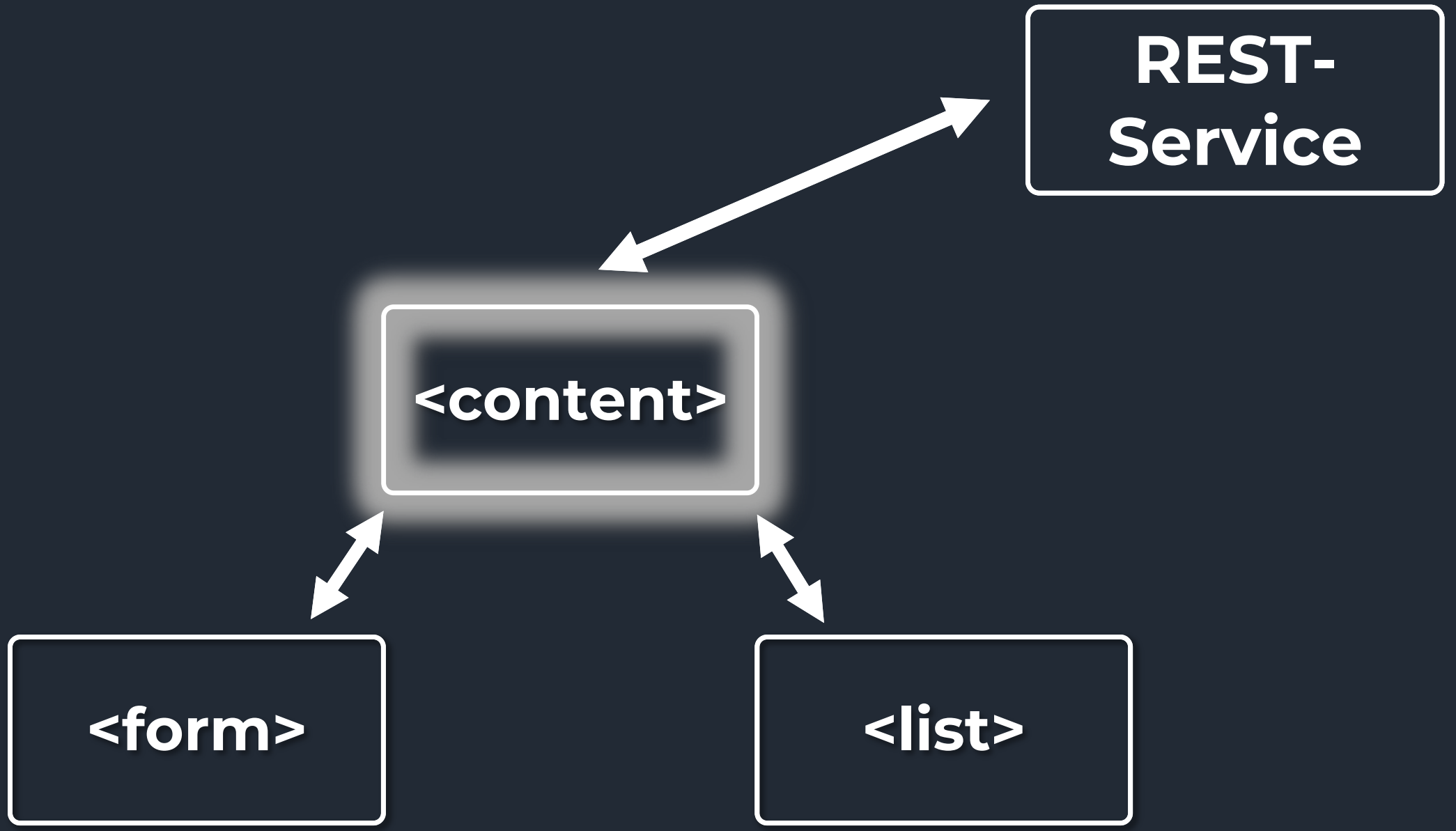

todoapp
localhost:4200/#/todo
Navbar
Home
Todo
Add todo
Add Todo
Todos
sfsdf
asdasd
2 items
Already Done
asdasd

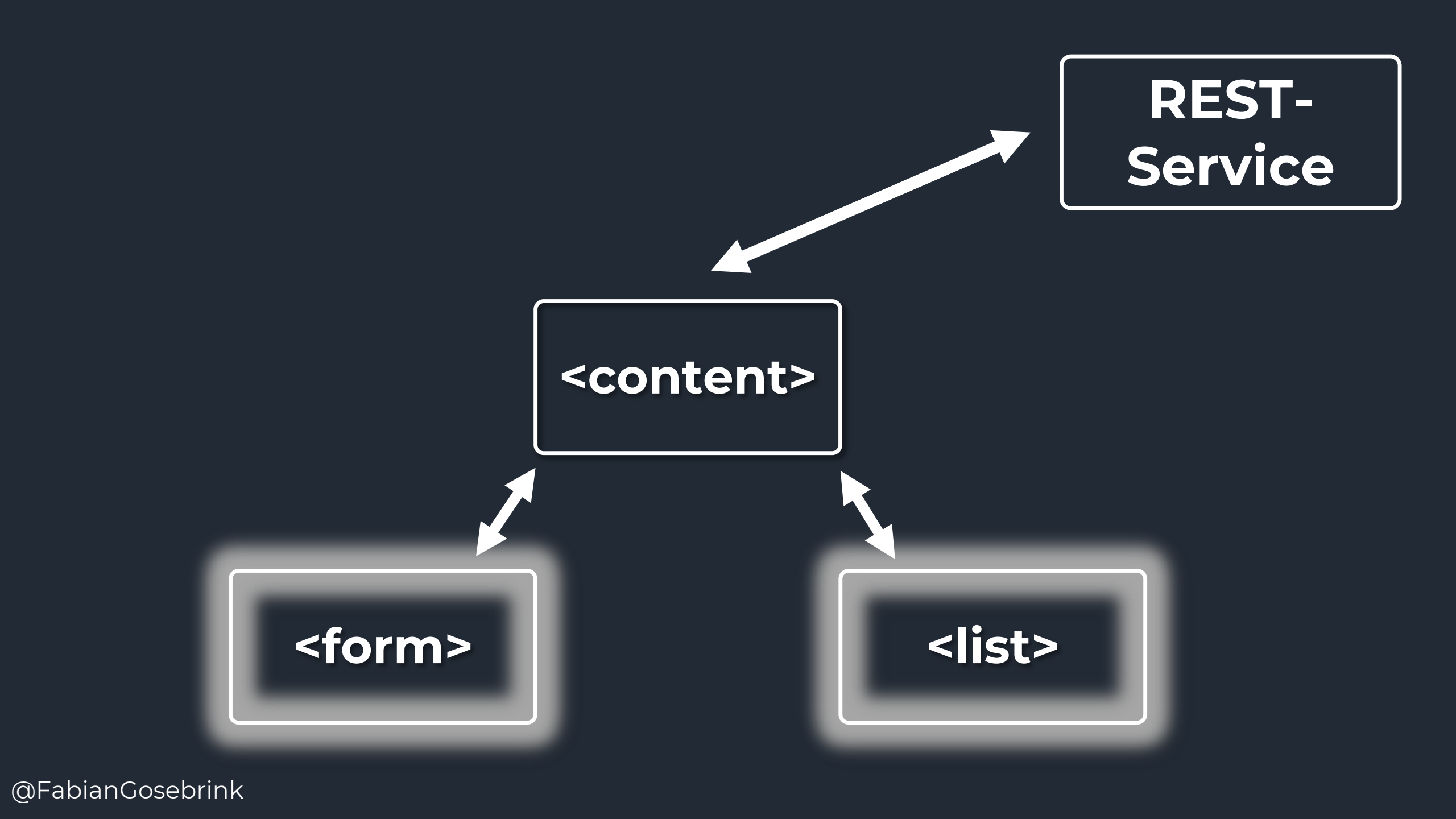

<content>
<form>
<list>

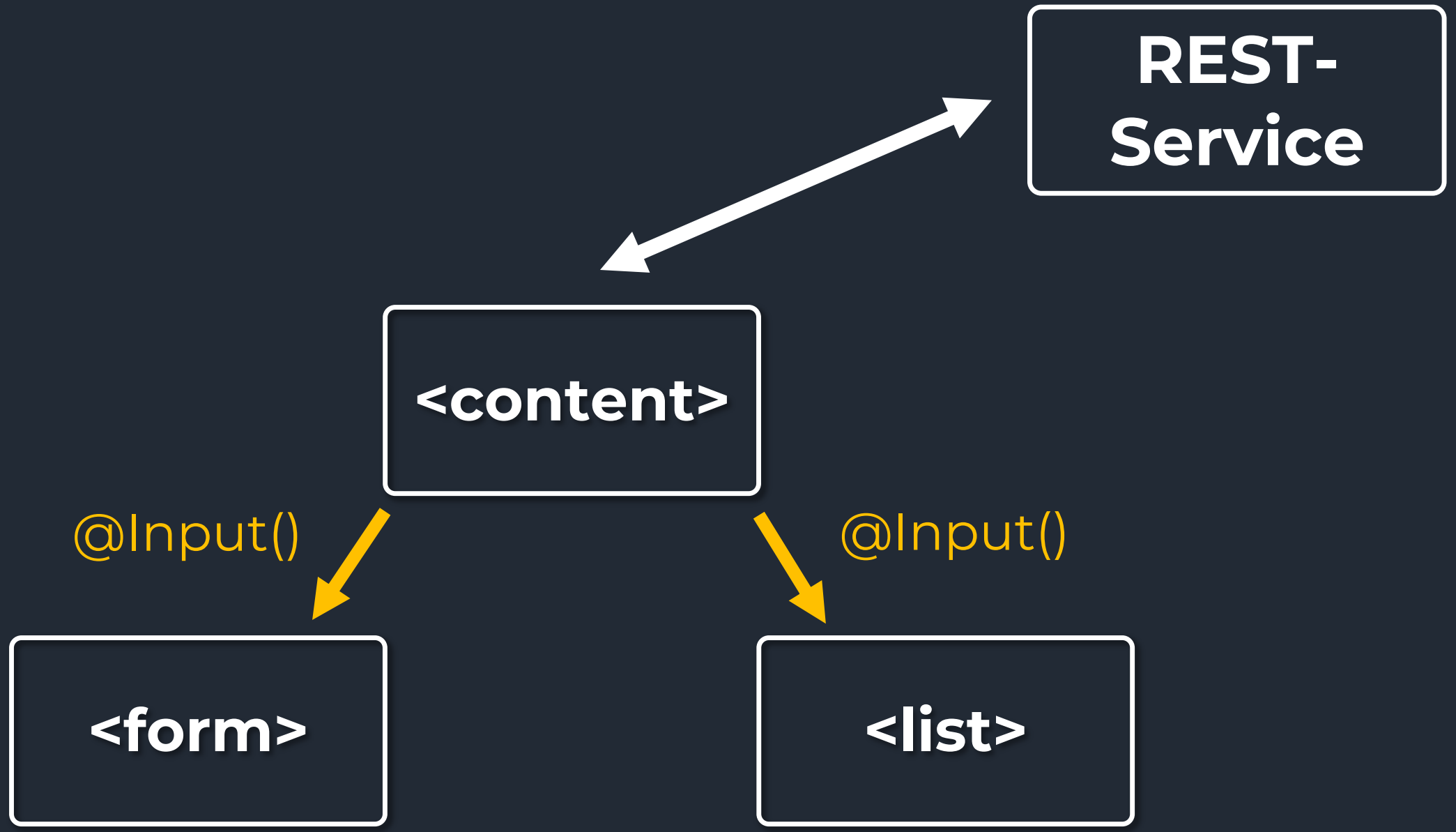


<app-todo-form></app-todo-form>
<app-todo-list></app-todo-list>

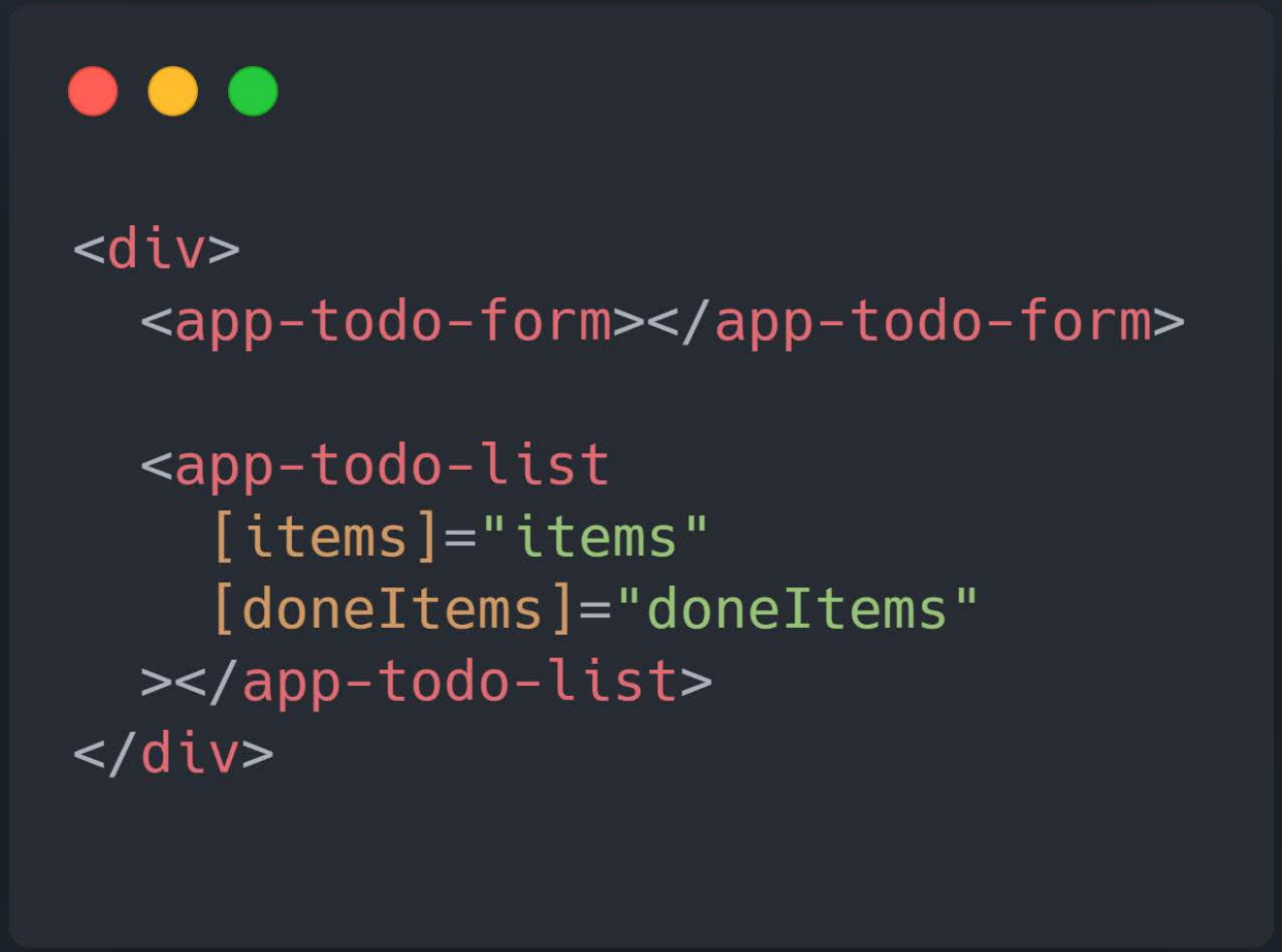

REST-
Service
<content>
<form>
<list>


REST-
Service
<content>
<form>
<list>

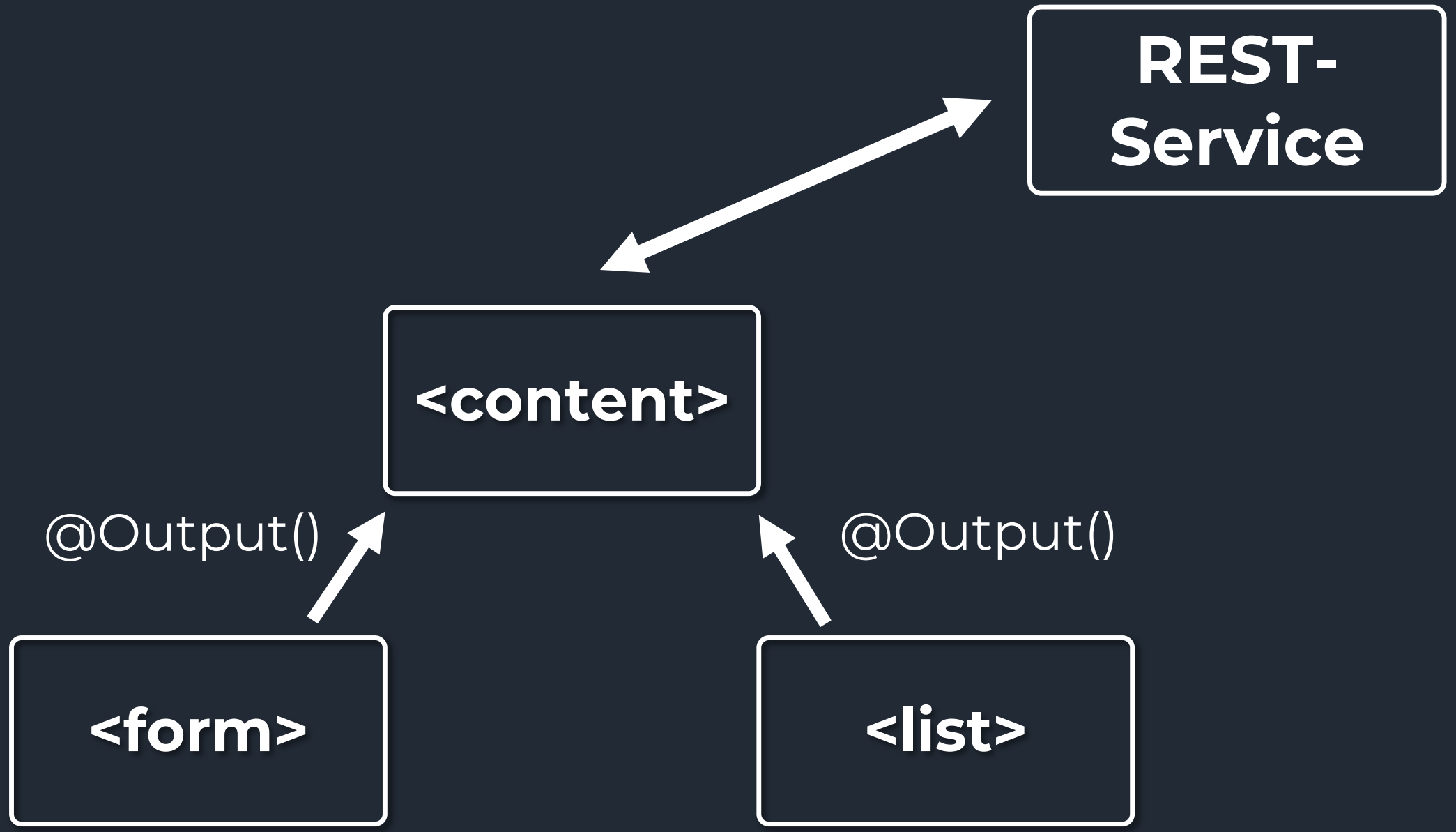

REST-
Service
<content>
<form>
<list>

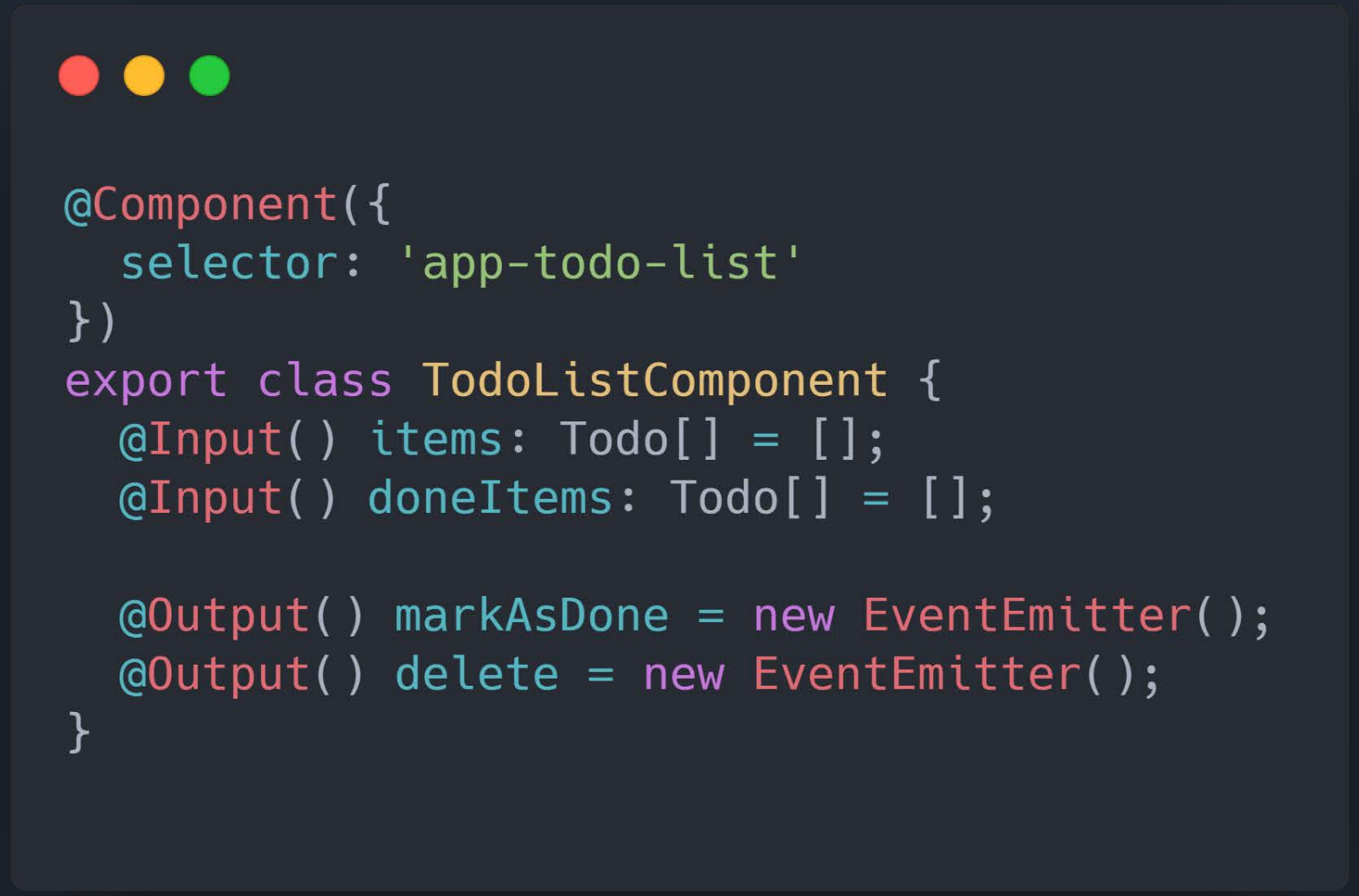

REST-
Service
<content>
@Input()
@Input()
<form>
<list>

```
@Component({
  selector: 'app-todo-list'
})
export class TodoListComponent {
  @Input() items: Todo[] = [];
  @Input() doneItems: Todo[] = [];
}
```



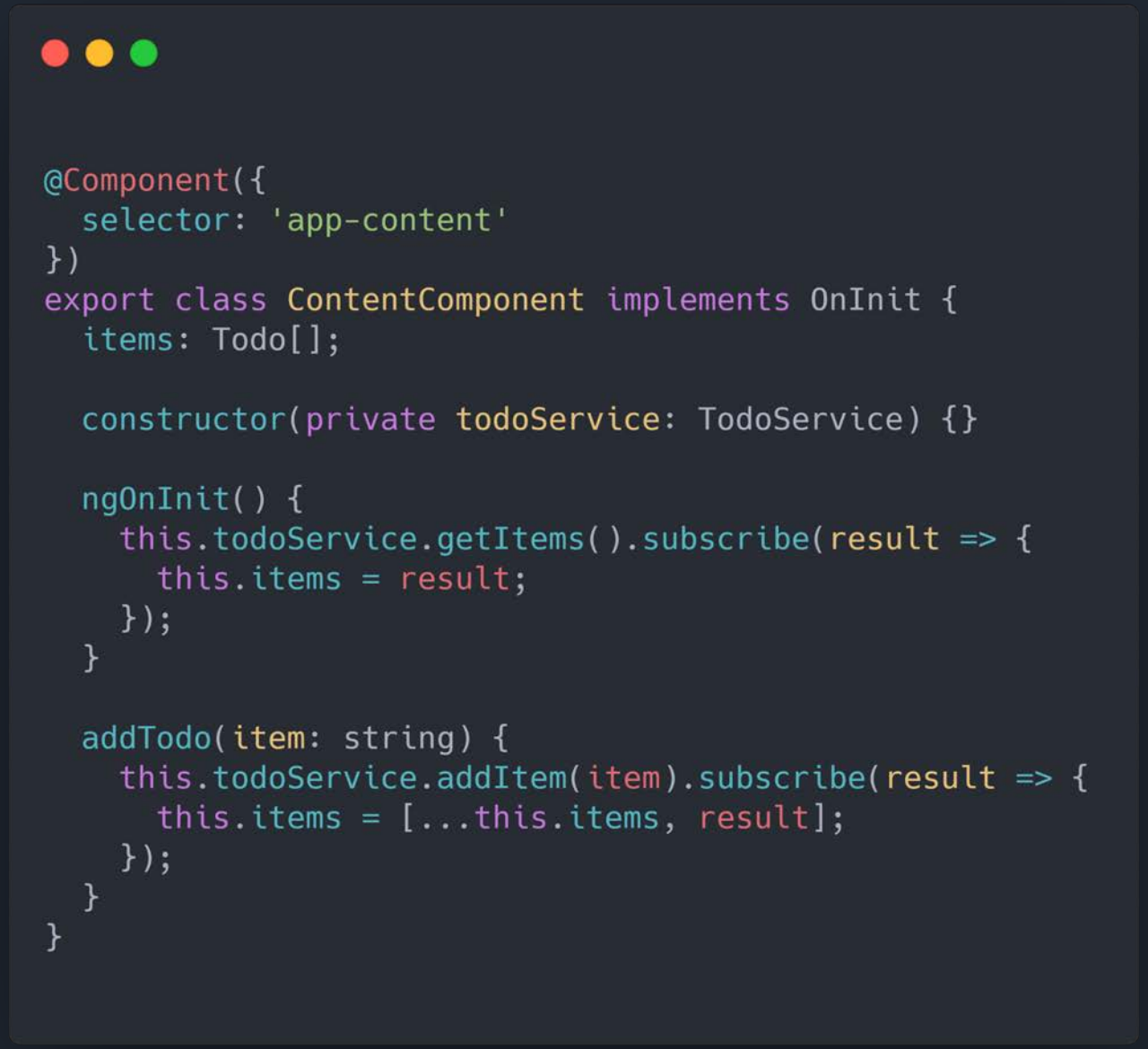
```
<div>
  <app-todo-form></app-todo-form>

  <app-todo-list
    [items]="items"
    [doneItems]="doneItems"
  ></app-todo-list>
</div>
```


REST-
Service
<content>
@Output()
@Output()
<form>
<list>

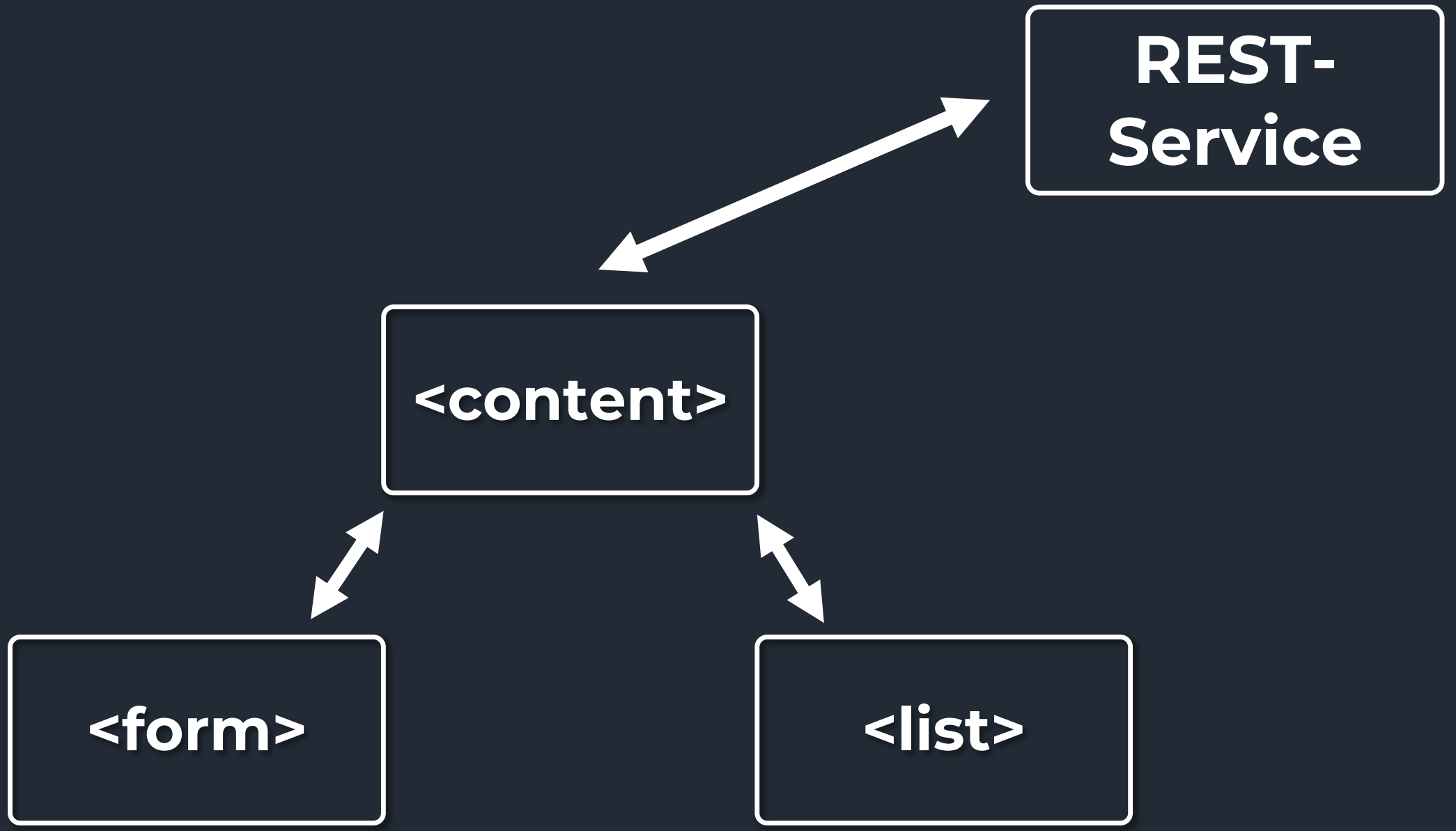


```
@Component({
  selector: 'app-todo-list'
})
export class TodoListComponent {
  @Input() items: Todo[] = [];
  @Input() doneItems: Todo[] = [];

  @Output() markAsDone = new EventEmitter();
  @Output() delete = new EventEmitter();
}
```

```
<div>
  <app-todo-form (todoAdded)="addTodo($event)"></app-todo-form>

  <app-todo-list
    [items]="items"
    [doneItems]="doneItems"
    (markAsDone)="markAsDone($event)"
    (delete)="deleteItem($event)"
  ></app-todo-list>
</div>
```



```
@Component({
  selector: 'app-content'
})
export class ContentComponent implements OnInit {
  items: Todo[];

  constructor(private todoService: TodoService) {}

  ngOnInit() {
    this.todoService.getItems().subscribe(result => {
      this.items = result;
    });
  }

  addTodo(item: string) {
    this.todoService.addItem(item).subscribe(result => {
      this.items = [...this.items, result];
    });
  }
}
```


REST-
Service
<content>
<form>
<list>


Data Flows Down
@FabianGosebrink


Data Flows Up
@FabianGosebrink


One Way
@FabianGosebrink


State
@FabianGosebrink

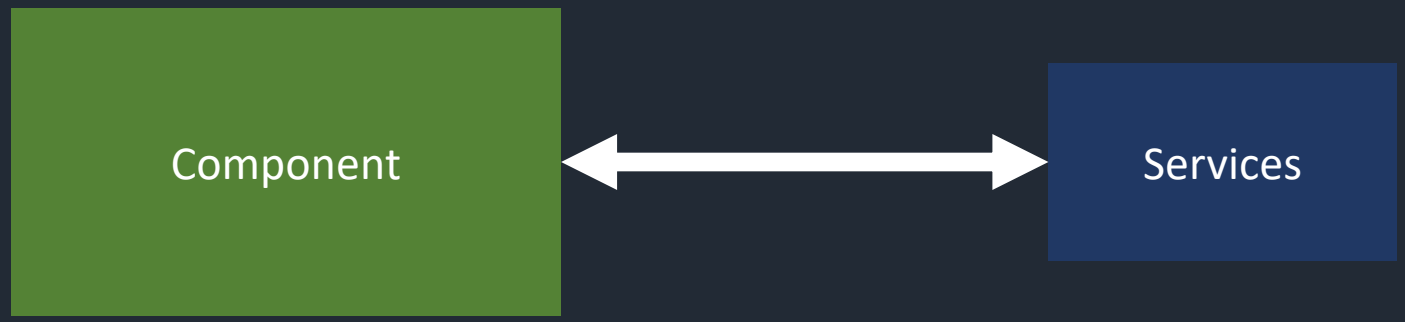
Where

@FabianGosebrink


Where
Change
@FabianGosebrink

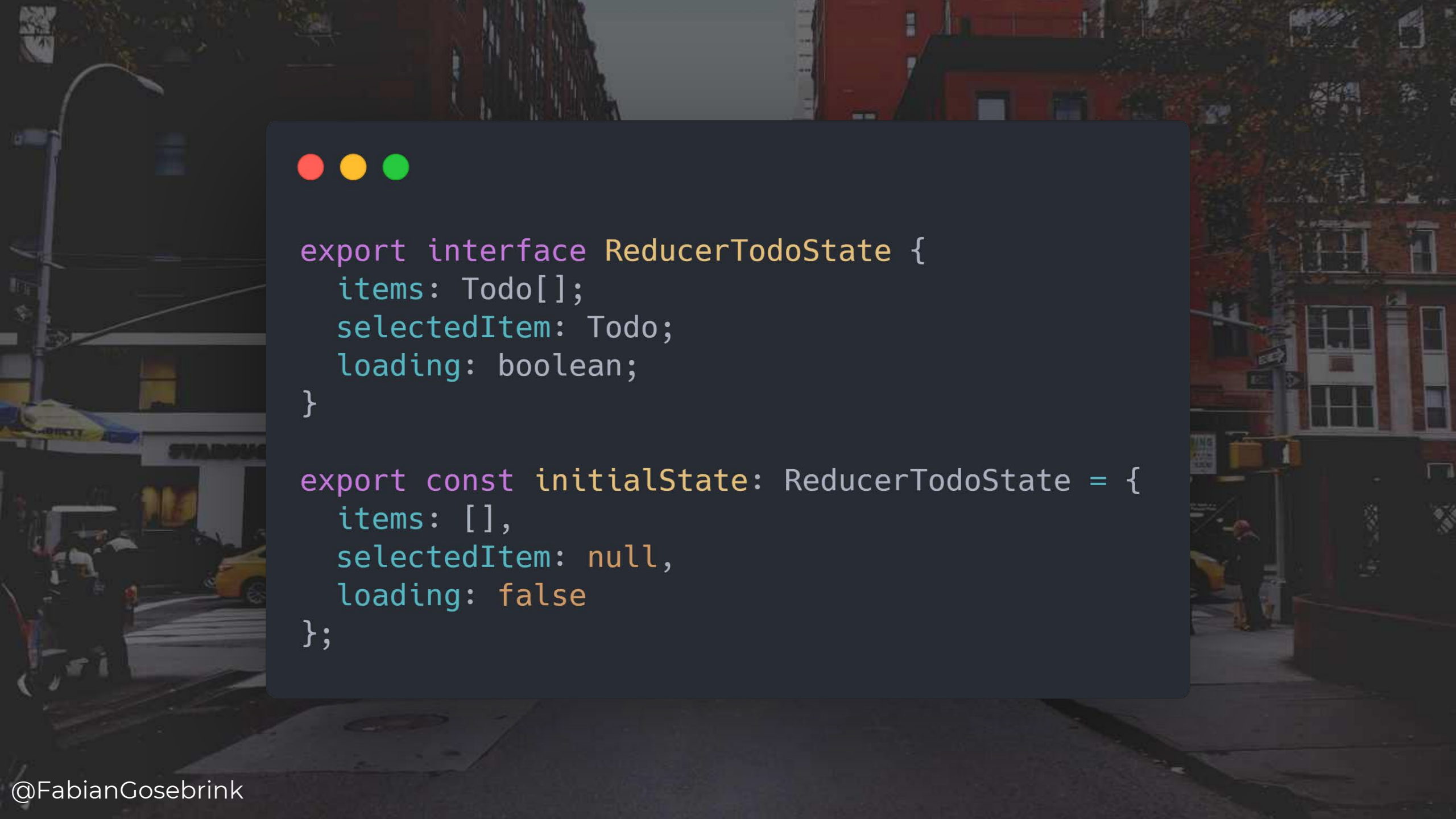

Where
Change
Effects
@FabianGosebrink

@FabianGosebrink


Component
Services

State

```ts
export interface ReducerTodoState {
  items: Todo[];
  selectedItem: Todo;
  loading: boolean;
}
```



```ts
export interface ReducerTodoState {
  items: Todo[];
  selectedItem: Todo;
  loading: boolean;
}

export const initialState: ReducerTodoState = {
  items: [],
  selectedItem: null,
  loading: false
};
```

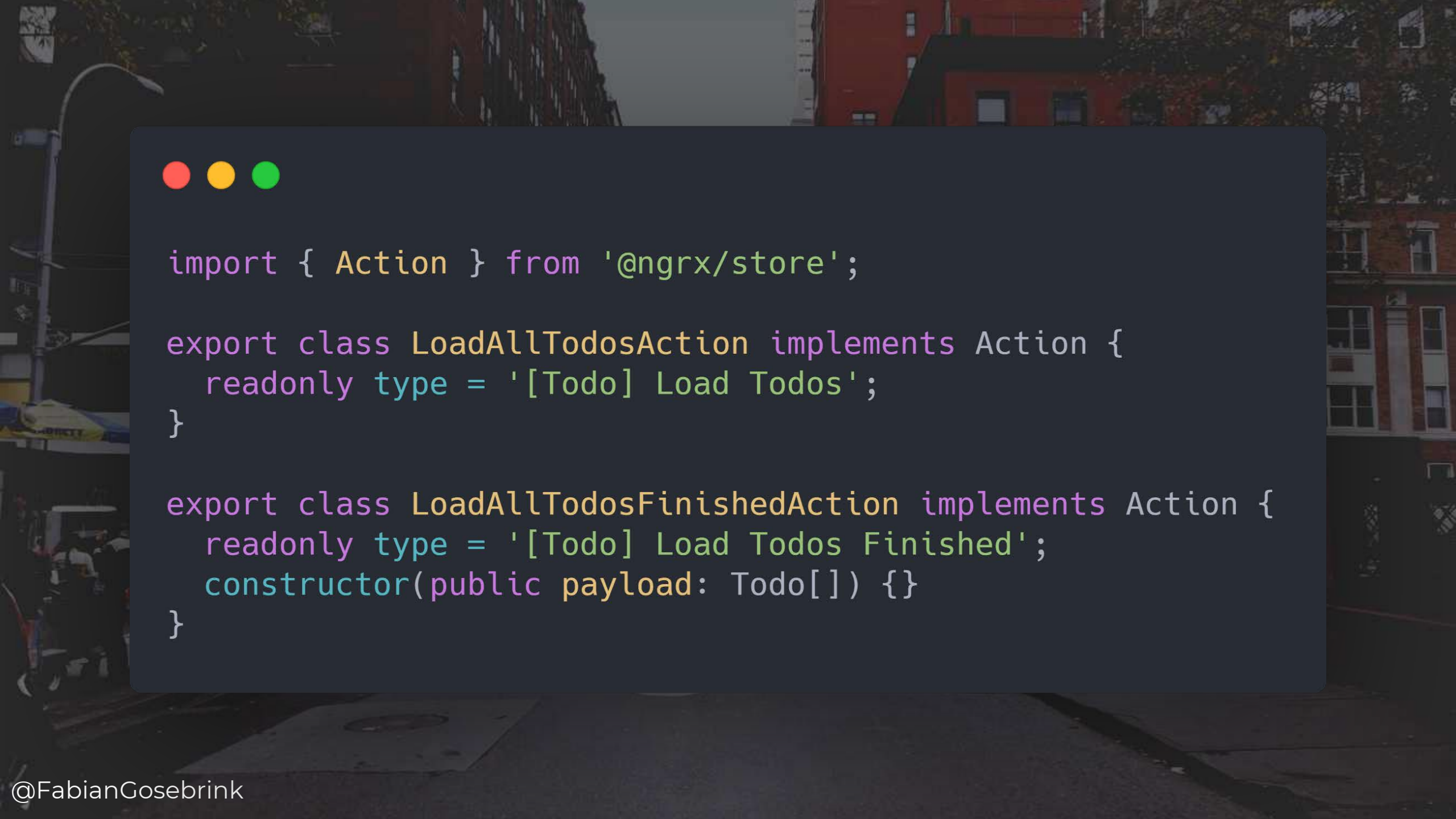
Component

@FabianGosebrink


{
type: '[Todo] Load Todos'
}

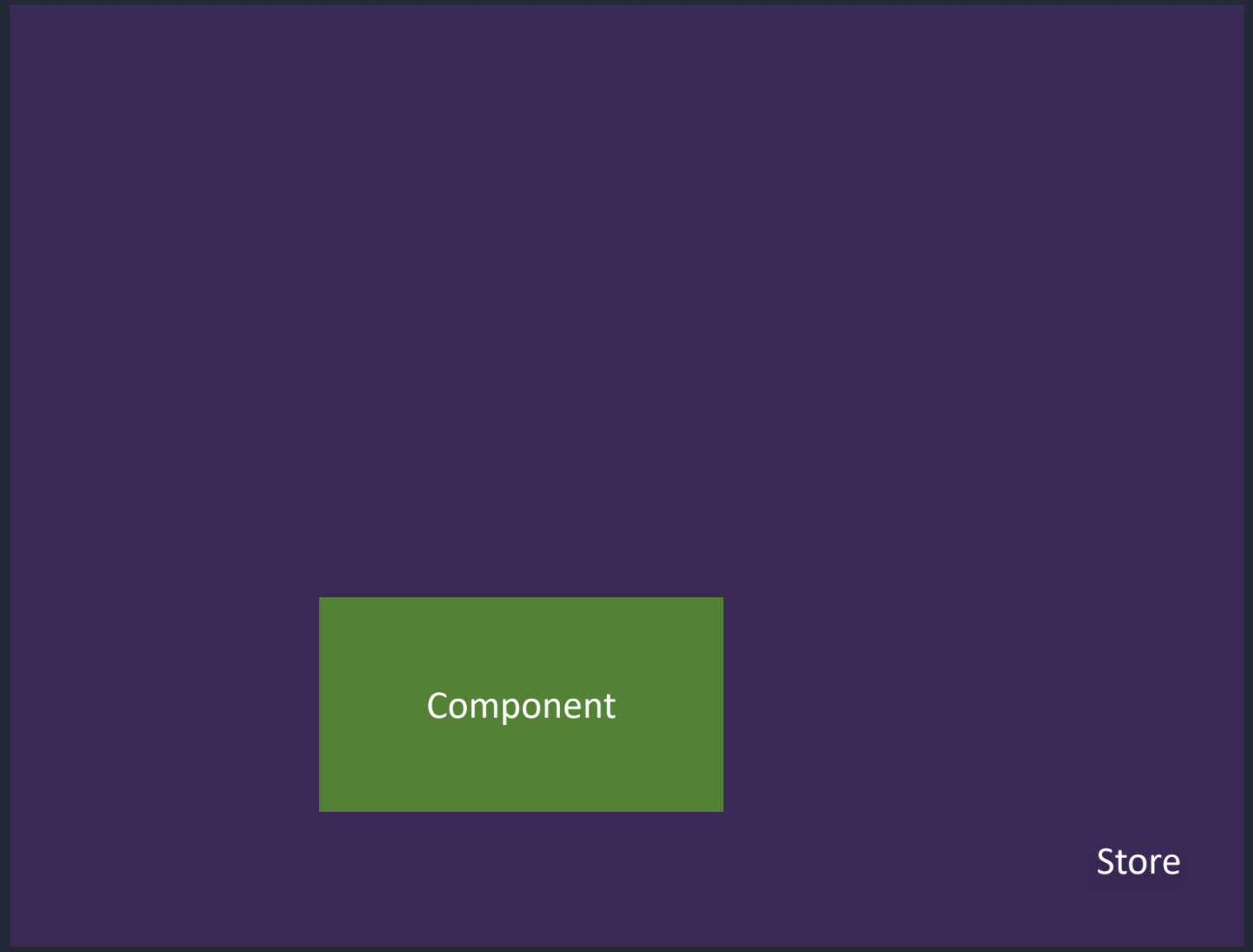

{
type: '[Todo] Load Todos',
payload?: ...
}

```
import { Action } from '@ngrx/store';

export class LoadAllTodosAction implements Action {
  readonly type = '[Todo] Load Todos';
}
```

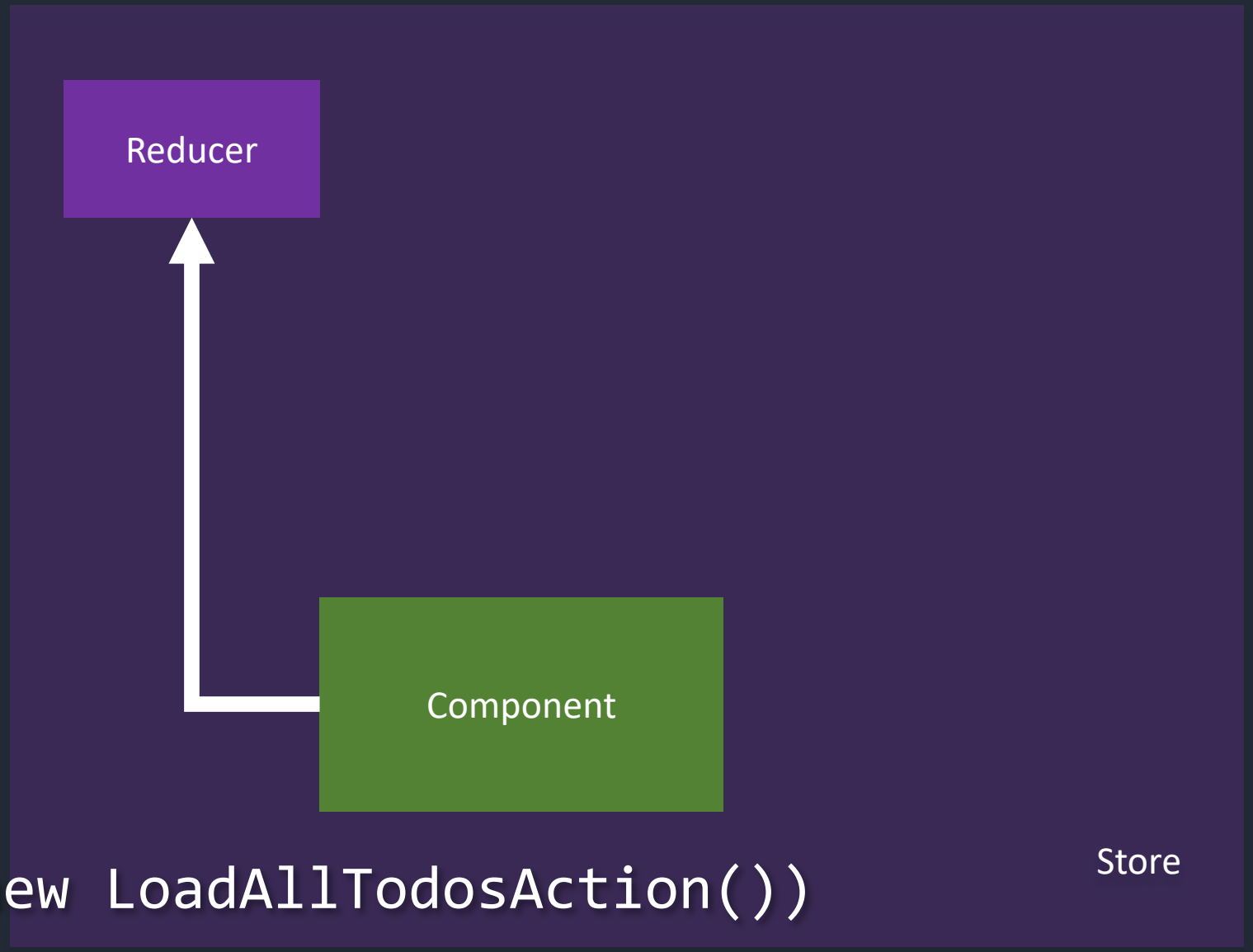


```ts
import { Action } from '@ngrx/store';

export class LoadAllTodosAction implements Action {
  readonly type = '[Todo] Load Todos';
}

export class LoadAllTodosFinishedAction implements Action {
  readonly type = '[Todo] Load Todos Finished';
  constructor(public payload: Todo[]) {}
}
```

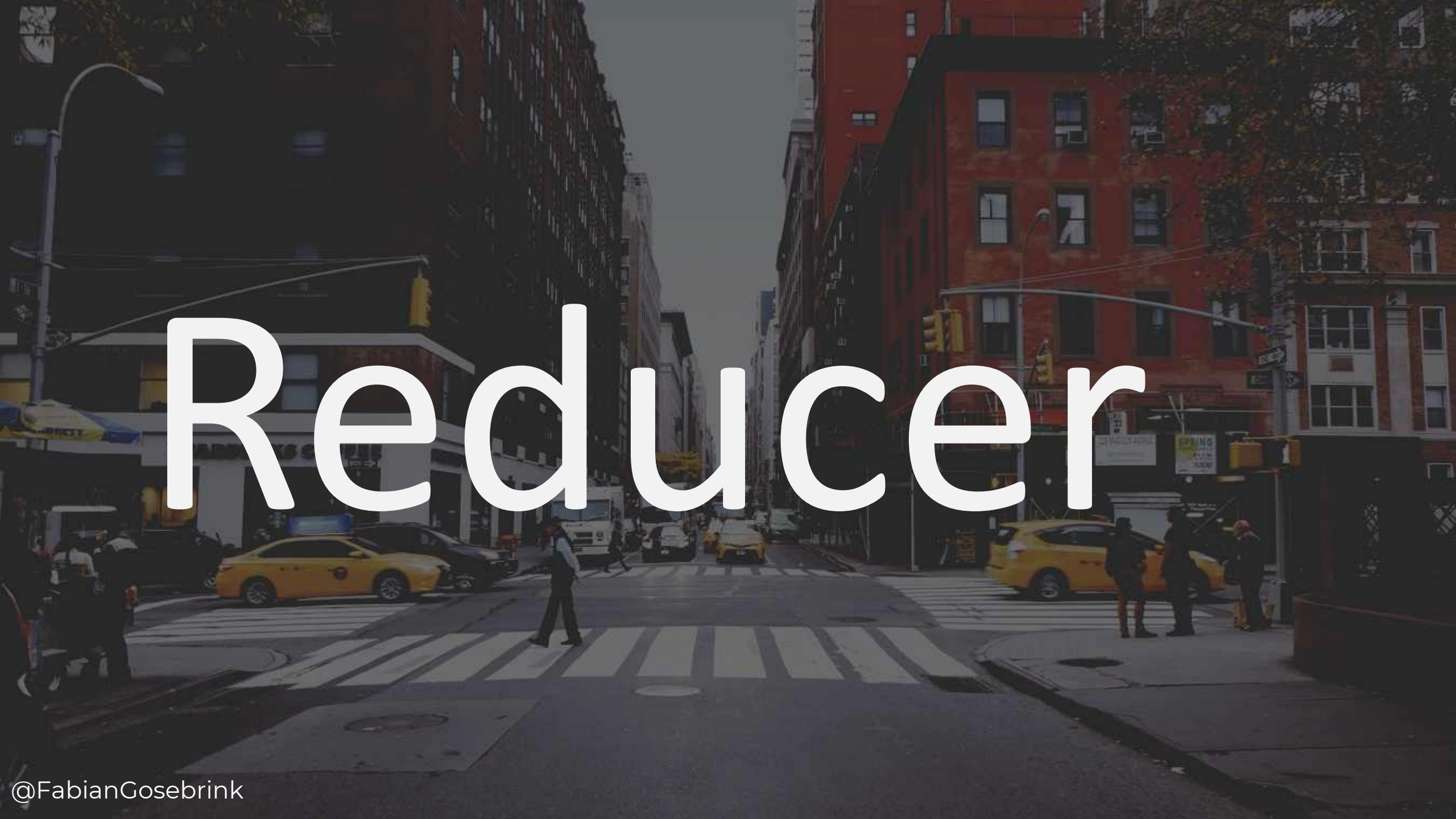
Component


Component
Store


```
store.dispatch(new LoadAllTodosAction())
```



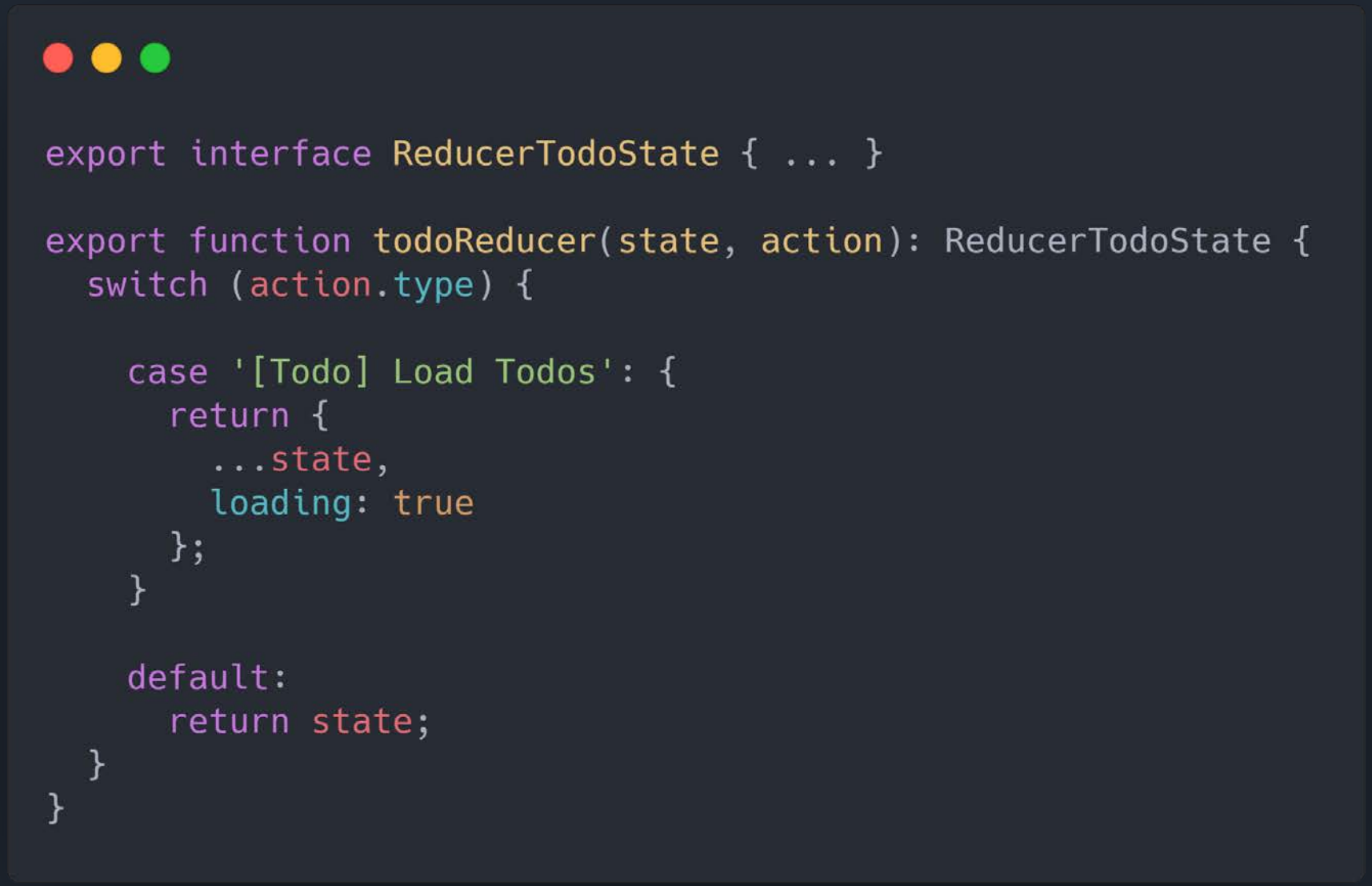

```
store.dispatch(new LoadAllTodosAction())
```



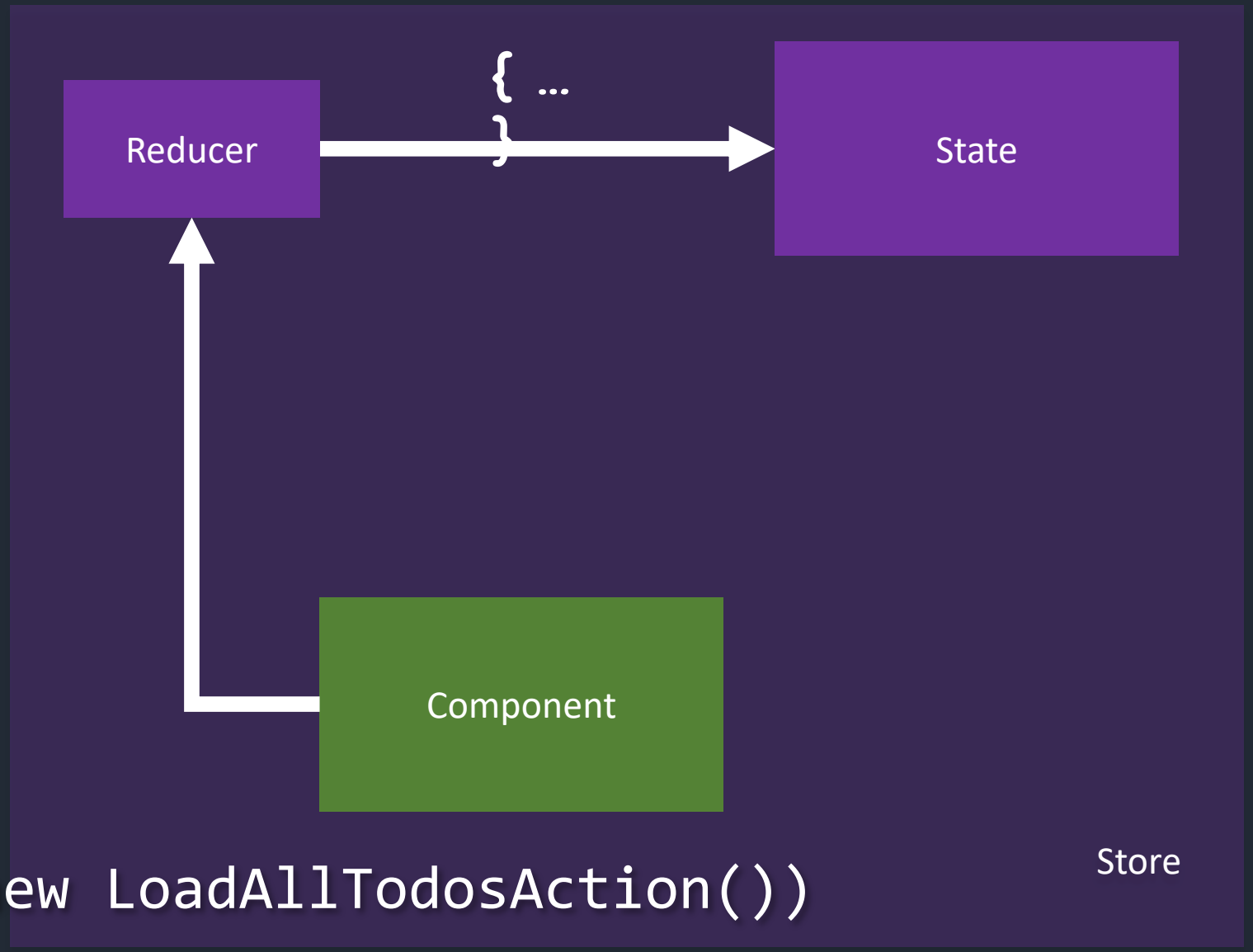
@FabianGosebrink

```
(oldState, action) => newState
```

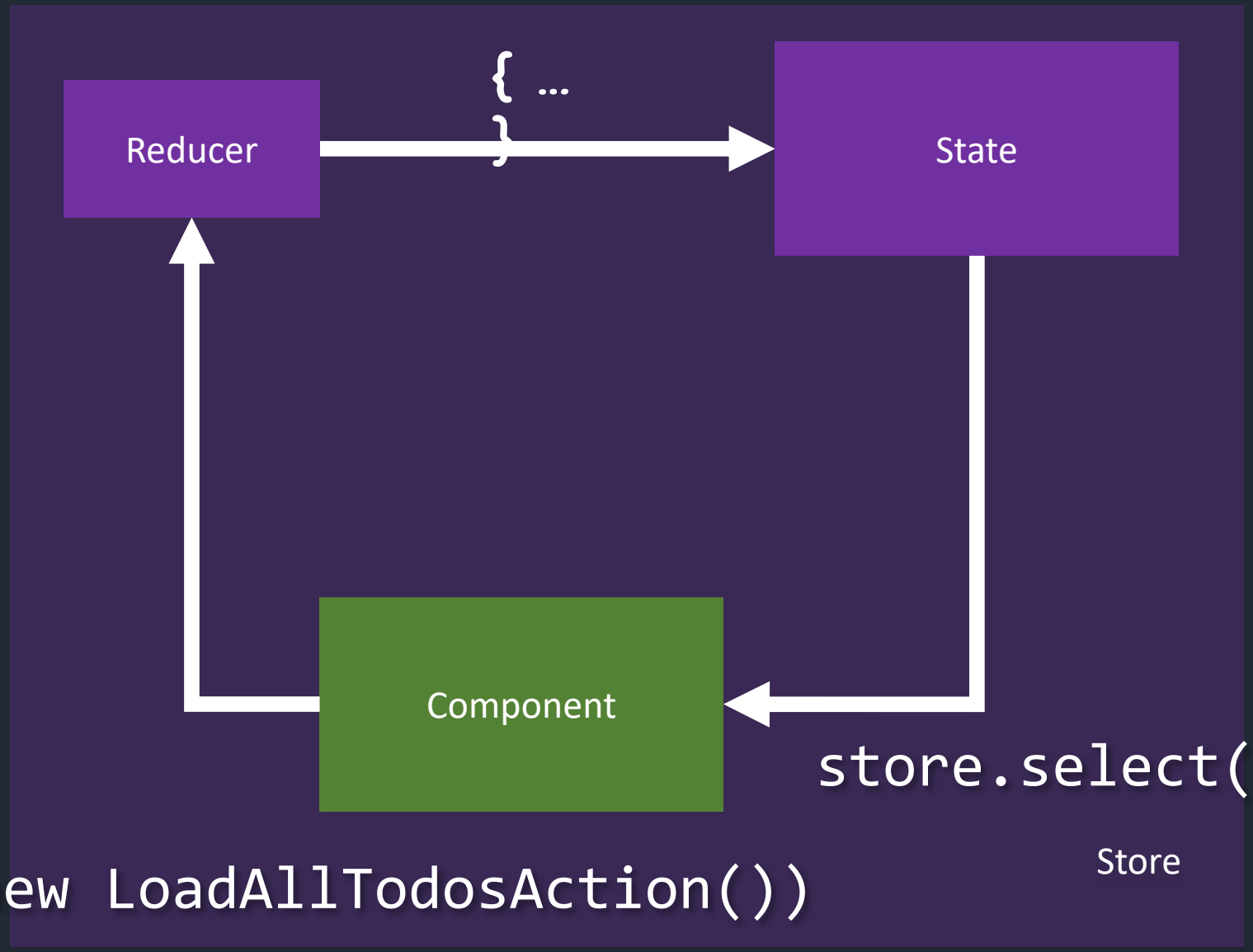

export interface ReducerTodoState { ... }
export function todoReducer(state, action): ReducerTodoState {
switch (action.type) {
default:
return state;
}
}



```
export interface ReducerTodoState { ... }

export function todoReducer(state, action): ReducerTodoState {
  switch (action.type) {

    case '[Todo] Load Todos': {
      return {
        ...state,
        loading: true
      };
    }

    default:
      return state;
  }
}
```




```
store.dispatch(new LoadAllTodosAction())
```

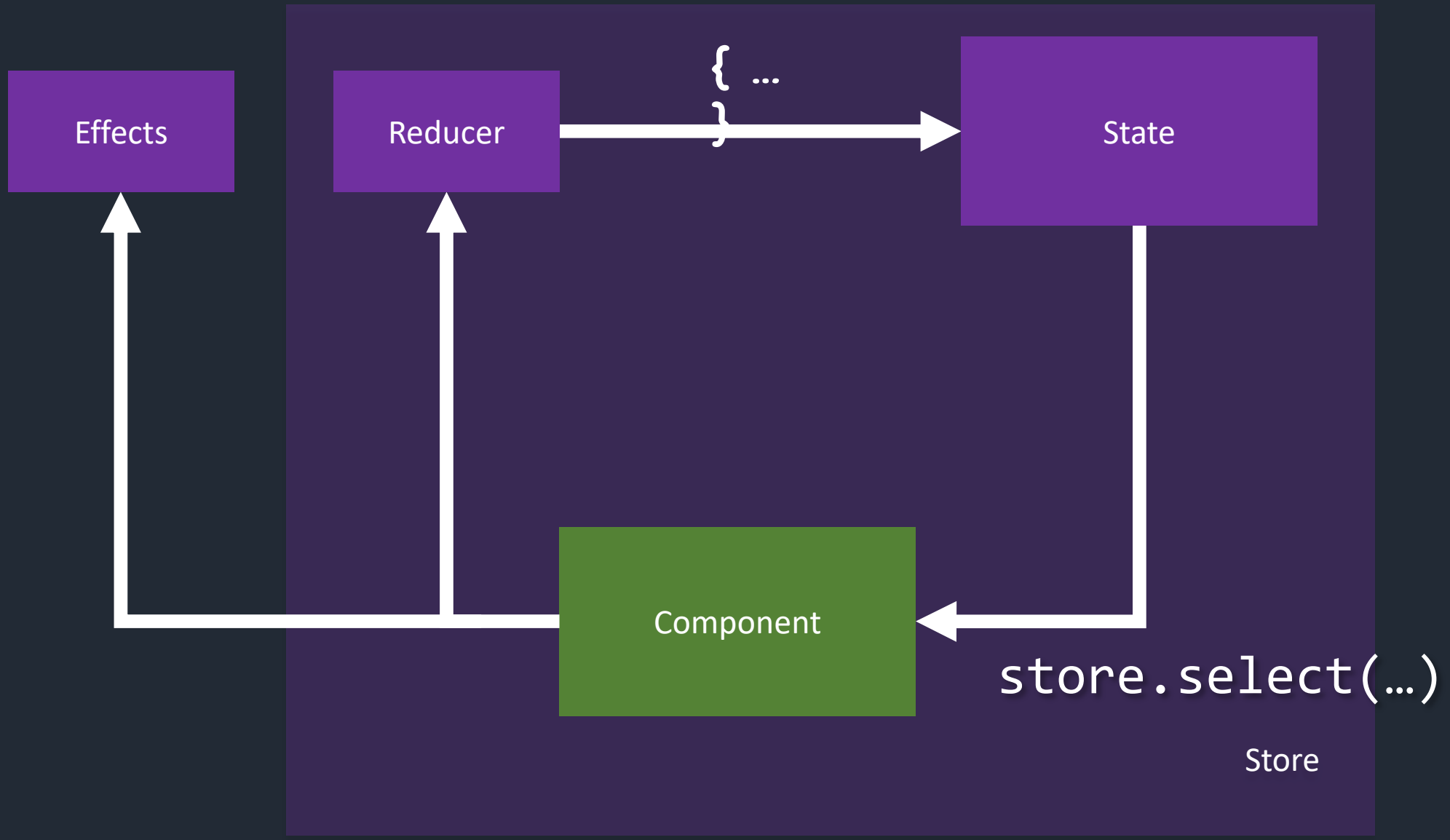



```
store.dispatch(new LoadAllTodosAction())
```

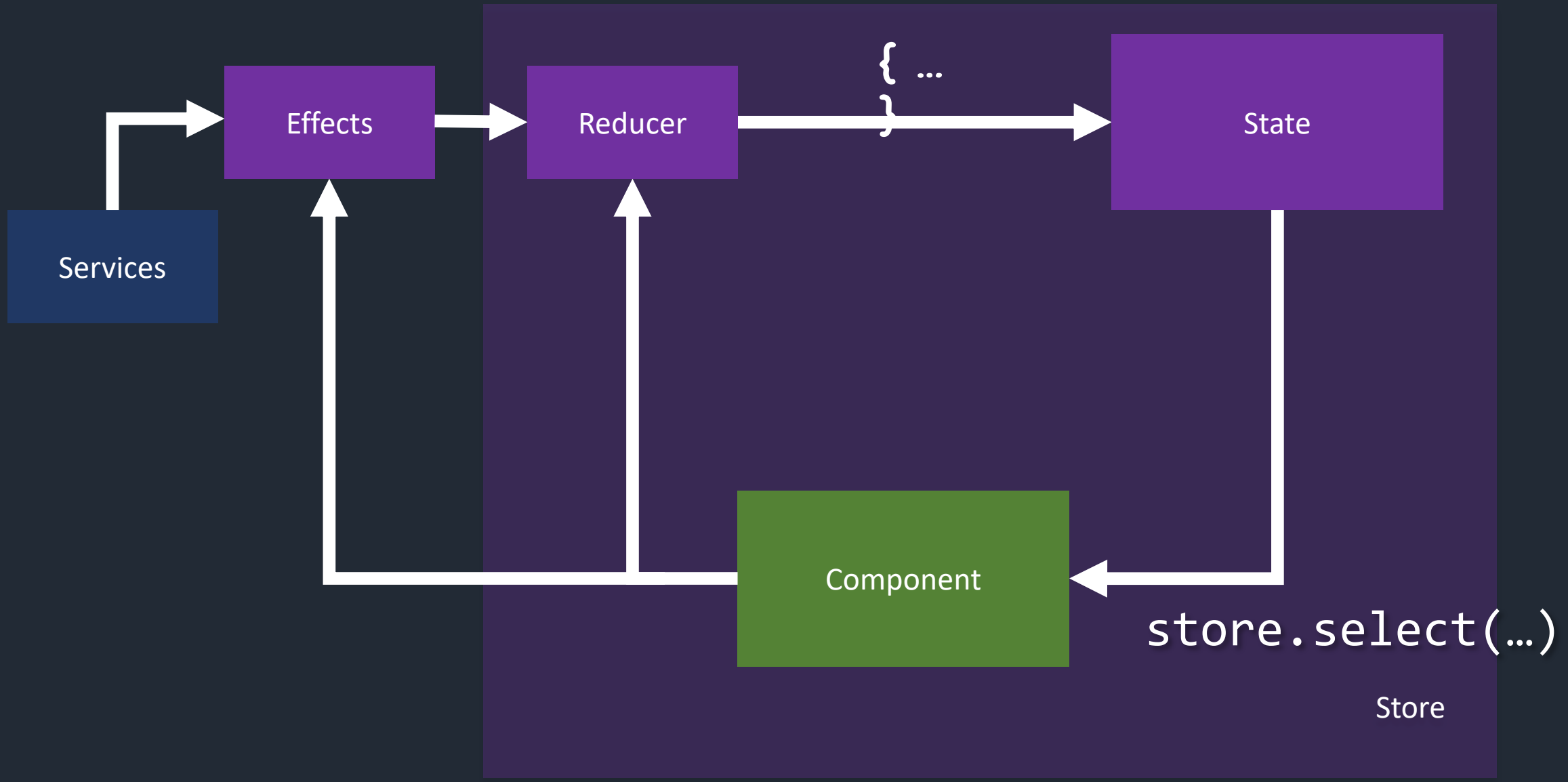
Container component	Presentational component
store.select(…)	@Input(…)
store.dispatch(…)	@Output(…)

@FabianGosebrink

@FabianGosebrink

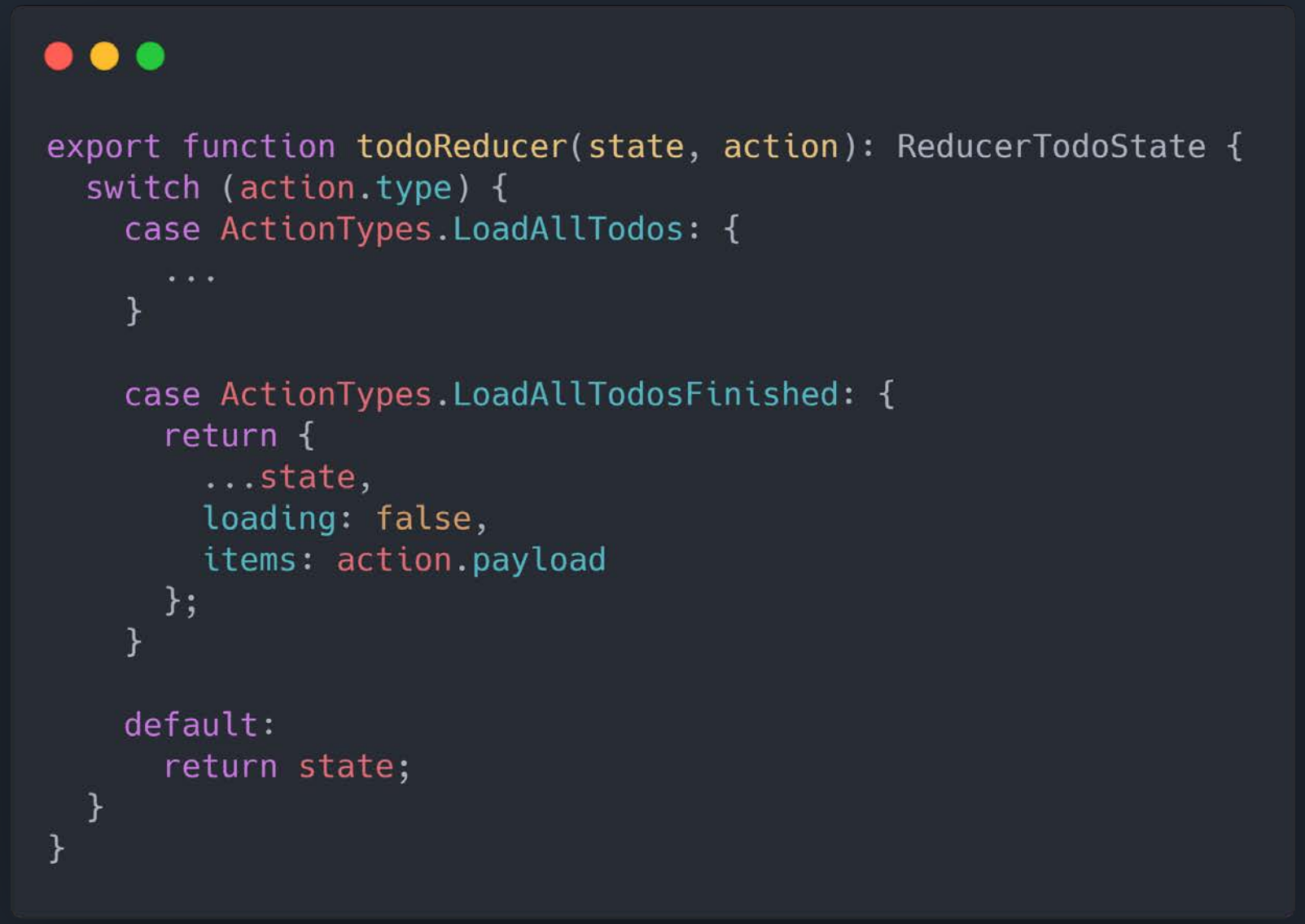



```
store.dispatch(new LoadAllTodosAction())
```

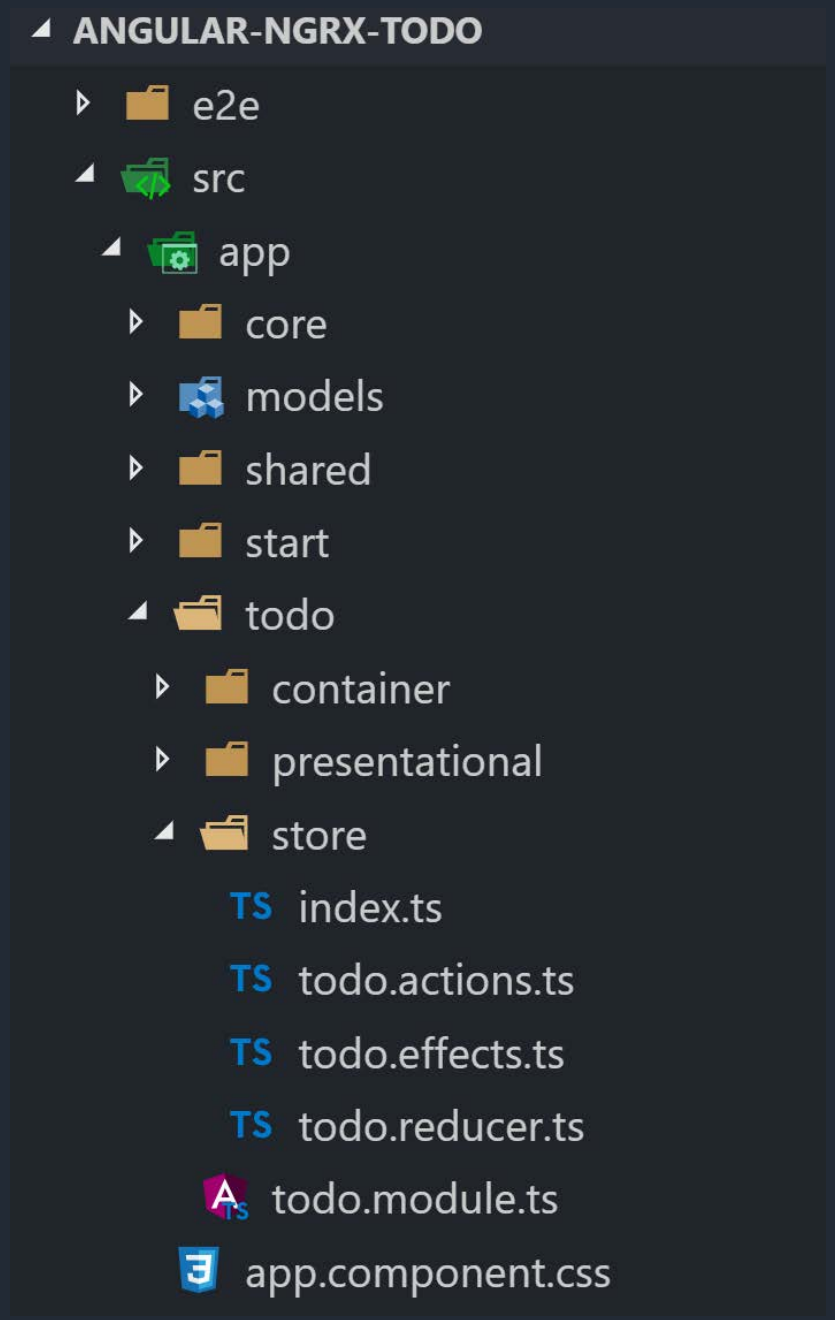



```
store.dispatch(new LoadAllTodosAction())
```

```
@Injectable()
export class TodoEffects {
  constructor(
      private actions$: Actions,
      private todoService: TodoService) {}

  @Effect()
  loadTodos$ = this.actions$.pipe(
    ofType(ActionTypes.LoadAllTodos),
    switchMap(() =>
      this.todoService.getItems()
        .pipe(
          map(
            (todos: Todo[]) => new LoadAllTodosFinishedAction(todos)
          )
        )
    )
  );
}
```



```
export function todoReducer(state, action): ReducerTodoState {
  switch (action.type) {
    case ActionTypes.LoadAllTodos: {
      ...
    }

    case ActionTypes.LoadAllTodosFinished: {
      return {
        ...state,
        loading: false,
        items: action.payload
      };
    }

    default:
      return state;
  }
}
```


ANGULAR-NGRX-TODO
e2e
src
app
core
models
shared
start
todo
container
presentational
store
index.ts
todo.actions.ts
todo.effects.ts
todo.reducer.ts
todo.module.ts
app.component.css

```
import { StoreModule } from '@ngrx/store';

@NgModule({
  imports: [
    StoreModule.forRoot(...)
  ],
})
export class AppModule {}
```

```
import { StoreModule } from '@ngrx/store';

@NgModule({
  imports: [
    StoreModule.forRoot({}),
  ],
})
export class AppModule {}
```

```
import { StoreModule } from '@ngrx/store';

@NgModule({
  imports: [
    StoreModule.forFeature(...),
  ],
})
export class TodoModule {}
```

```
import { StoreModule } from '@ngrx/store';
import { todoReducer } from './store/todo.reducer';

@NgModule({
  imports: [
    StoreModule.forFeature("todoFeature", todoReducer),
  ],
})
export class TodoModule {}
```

```ts
import { StoreModule } from '@ngrx/store';
import { todoReducer } from './store/todo.reducer';

@NgModule({
  imports: [
    StoreModule.forFeature("todoFeature", todoReducer),
  ],
})
export class TodoModule {}
```

```
{
  todoFeature: {
    // …
  }
}
```

```ts
import { StoreModule } from '@ngrx/store';
import { todoReducer } from './store/todo.reducer';

@NgModule({
  imports: [
    StoreModule.forFeature("todoFeature", todoReducer),
  ],
})
export class TodoModule {}
```

```
{
  todoFeature: {
    items: [],
    selectedItem:
null,
    loading: false
  }
}
```


Ben Lesh
@BenLesh
Following
Oh.. and on the anti-redux-because-boilerplate front:
That "boilerplate" is directly the result of separation of concerns: You have to define many separate things. It's more than you're used to, because you weren't separating concerns before when your app was a big pile of code.
3:44 AM - 13 Feb 2018
26 Retweets 108 Likes
6
26
108
Tweet your reply
Ben Lesh @BenLesh · Feb 13
Tangling state and rendering together into one big, freaky component, or many tiny, cute, little, freaky components doesn't mean you've separated concerns.
2
3
25


@FabianGosebrink

```
export class ContentComponent implements OnInit {
  items: Todo[];
  doneItems: Todo[];

  constructor(private todoService: TodoService) {}

  ngOnInit() {
    this.getData();
  }

  addTodo(item: string) {
    this.todoService.addItem(item).subscribe(
      () => {
        this.getData();
      },
      error => console.log(error)
    );
  }

  markAsDone(item: Todo) {
    this.todoService.updateItem(item).subscribe(
      () => {
        this.getData();
      },
      error => console.log(error)
    );
  }

  private getData() {
    this.todoService.getItems().subscribe(
      items => {
        this.items = items.filter(x => !x.done);
        this.items = items.filter(x => x.done);
      },
      error => console.log(error)
    );
  }
}
```

```
export class ContentComponent implements OnInit {
  items$: Observable<Todo[]>;
  doneItems$: Observable<Todo[]>;

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

  ngOnInit() {
    this.items$ = this.store.pipe(select(/*...*/));
    this.doneItems$ = this.store.pipe(select(/*...*/));

    this.store.dispatch(new LoadAllTodosAction());
  }

  addTodo(item: string) {
    this.store.dispatch(new AddTodoAction(item));
  }

  markAsDone(item: Todo) {
    this.store.dispatch(new SetAsDoneAction(item));
  }
}
```

Demo

```
import { createAction, props } from '@ngrx/store';

export const addTodo = createAction(
  '[Todo] Add Todo',
  props<{ value: string }>()
);
```


@ngrx/data

```
import { EntityMetadataMap } from '@ngrx/data';

const entityMetadata: EntityMetadataMap = {
  Todo: {},
  // Add other items here
};

export const entityConfig = {
  entityMetadata
};
```

```
import { NgModule } from '@angular/core';
import { EffectsModule } from '@ngrx/effects';
import { StoreModule } from '@ngrx/store';
import { DefaultDataServiceConfig, EntityDataModule } from '@ngrx/data';
import { entityConfig } from './entity-metadata';

@NgModule({
  imports: [
    StoreModule.forRoot({}),
    EffectsModule.forRoot([]),
    EntityDataModule.forRoot(entityConfig)
  ]
})
export class AppModule {}
```

```ts
import { Injectable } from '@angular/core';
import {
  EntityCollectionServiceBase,
  EntityCollectionServiceElementsFactory
} from '@ngrx/data';
import { Todo } from '../todo.model';

@Injectable({ providedIn: 'root' })
export class TodoService extends EntityCollectionServiceBase<Todo> {
  constructor(serviceElementsFactory: EntityCollectionServiceElementsFactory) {
    super('Todo', serviceElementsFactory);
  }
}
```

```
export class TodoComponent implements OnInit {
  loading$: Observable<boolean>;
  todos$: Observable<Todo[]>;

  constructor(private todoService: TodoService) {
    this.todos$ = todoService.entities$;
    this.loading$ = todoService.loading$;
  }

  ngOnInit() {
    this.todoService.getAll();
  }

  add(todo: Todo) {
    this.todoService.add(todo);
  }

  delete(todo: Todo) {
    this.todoService.delete(todo.id);
  }
}
```

Fabian Gosebrink

http://offering.solutions
http://github.com/FabianGosebrink
http://fabian-gosebrink.com

@FabianGosebrink