ANGULAR 2 WORKSHOP

JOHAN KUSTERMANS

Who Am I?

- Technical Consultant RealDolmen (10 years)
  - Java developer / architect : frontend to backend
  - Angular 2
- Stationed at DHL
- Email: johan.kustermans@realdolmen.com

REALDOLMEN

to get there, together

PLANNING

- Overview
- Components
- Dependency Injection
- Observables
- Forms
- Routing

Startup

- git clone https://github.com/RDAngular2/UGentWorkshop.git (your work project)
- git clone https://github.com/RDAngular2/UGentWorkshopSolution.git (the solution)
- Run “npm install” in your project folder

What is Angular 2?

Framework to build GUI client applications in the modern Javascript and HTML ecosystem

… but it does not provide GUI components!

angular.io

Introduction

Features

- CROSS PLATFORM
  - Desktop / Mobile
  - Web / Native

- BUILT FOR SPEED
  - Code generation optimization
  - Code splitting & lazy loading
  - Change detection optimizations
  - Ahead Of Time compilation : Template compilation during build
  - Universal : initial page generation on server

Features

- PRODUCTIVITY & MAINTAINABILITY
  - Structured Language : Typescript (or ES 6 / 2015)
  - HTML Template Language
  - Easy Extendability of HTML
  - IDE support : Webstorm, Visual Studio Code, Visual Studio
  - GUI Libraries : Angular Material, Prime NG, Kendo, Wijmo / Mobile: Ionic

- FULL DEVELOPMENT CYCLE
  - Modern Build Tools
  - Testing

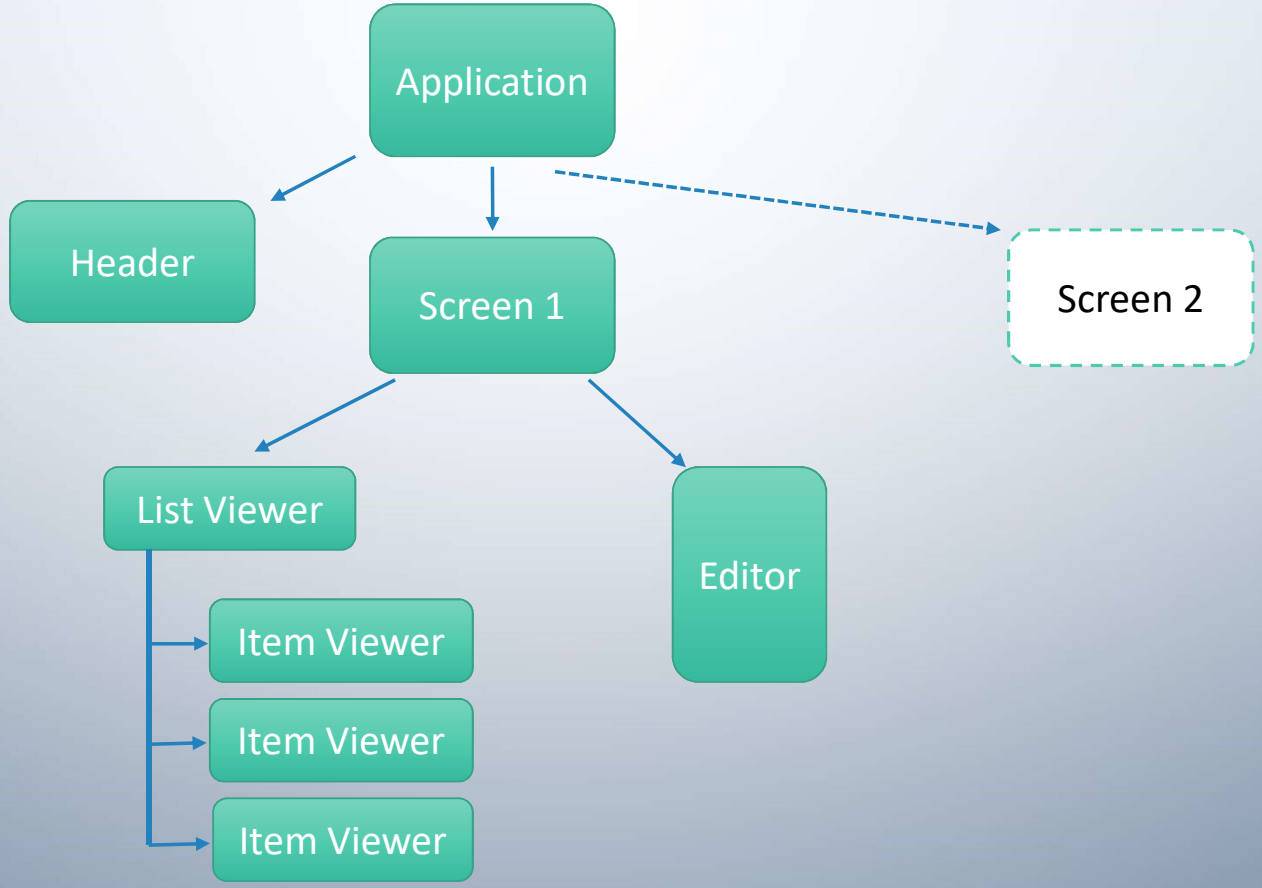
Key Aspects

- Typescript: Modules, Classes, TYPES & …
- Components & Directives
- HTML Template Language
- Wiring via Dependency Injection
- Data & Event binding
- Observables : event streams
- HTTP
- Navigation API : Routing
- Forms & Validation

Typescript

Components

Component Tree



Component

Component : self descriptive unit

- State and behaviour defined in class
- View via HTML Template
- Defines Input & Output API
- Wiring via dependency injection
- Well defined Lifecycle & callbacks: onInit, onDestroy, …

Anatomy of a Component

application-header.ts

```
@Component({
    selector: "application-header"
    templateUrl: "application-header.html"
})
class ApplicationHeaderComponent {

    applicationName : string;

    search() : void {
        … perform search …
    }
}
```

application-header.html

```
<span>{{applicationName}}<span>

<button (click)="search()">Search</button>
```

application.html

```
<application-header></application-header>
<div>
                … application content …
</div>
```

Exercise

1. Start server & compile code: “npm run start”

2. Make sure the application runs succesfully and

   - The title shows “ !!! TODO: show application name !!! ”
   - If the Search Icon is clicked, an alert is show via the component class.

   The entry point of the application is “application.html”.

Components

application-header.ts

```
@Component({
   selector: "application-header"
   templateUrl: "application-header.html"
})
class ApplicationHeaderComponent {

   @Input()
   applicationName : string;

}
```

application.ts

```
@Component({
   selector: "application"
   templateUrl: "application.html"
})
class ApplicationComponent {

  name : string = "Angular 2 Workshop";


}
```

application.html

```
<application-header [applicationName]="name" />
<div>
                … application content …
</div>
```

Exercise

Pass the name from the application component to the application header component so that the title “Angular 2 Workshop” is shown.

Structural Directives

- Repeaters

```
@Component({…})
class NameListViewerComponent
{

   names : string[];

}
```

```
<ul>
  <li *ngFor="let name of names">
       {{name}}
  </li>
</ul>
```

- NgIf and NgSwitch directives:

  *ngIf, *ngSwitch/*ngSwitchCase (see angular.io)

Exercise

1. Show the contact-viewer component in your application

2. Make sure it displays the right properties (see contact-viewer.component.html)

3. Show the contact-list-viewer-initial component in your application. Use the contact-viewer component to show the list of contacts.

4. If the user clicks on a contact in the list, it should be selected in the list viewer.

Dependency Injection

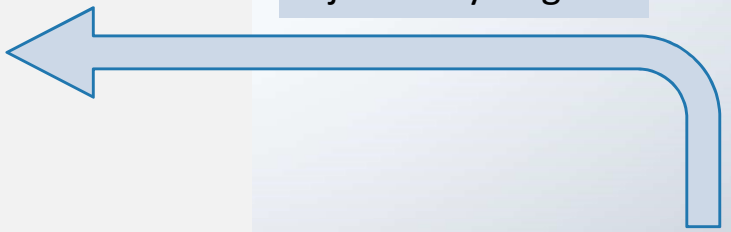
Dependency Injection

```
@Component({…})
class ContactListViewerComponent {

   contactService : ContactService;

   contacts : Contact[];

   refresh() : void {
       this.contacts =
          this.contactService.getContacts();
   }
}
```

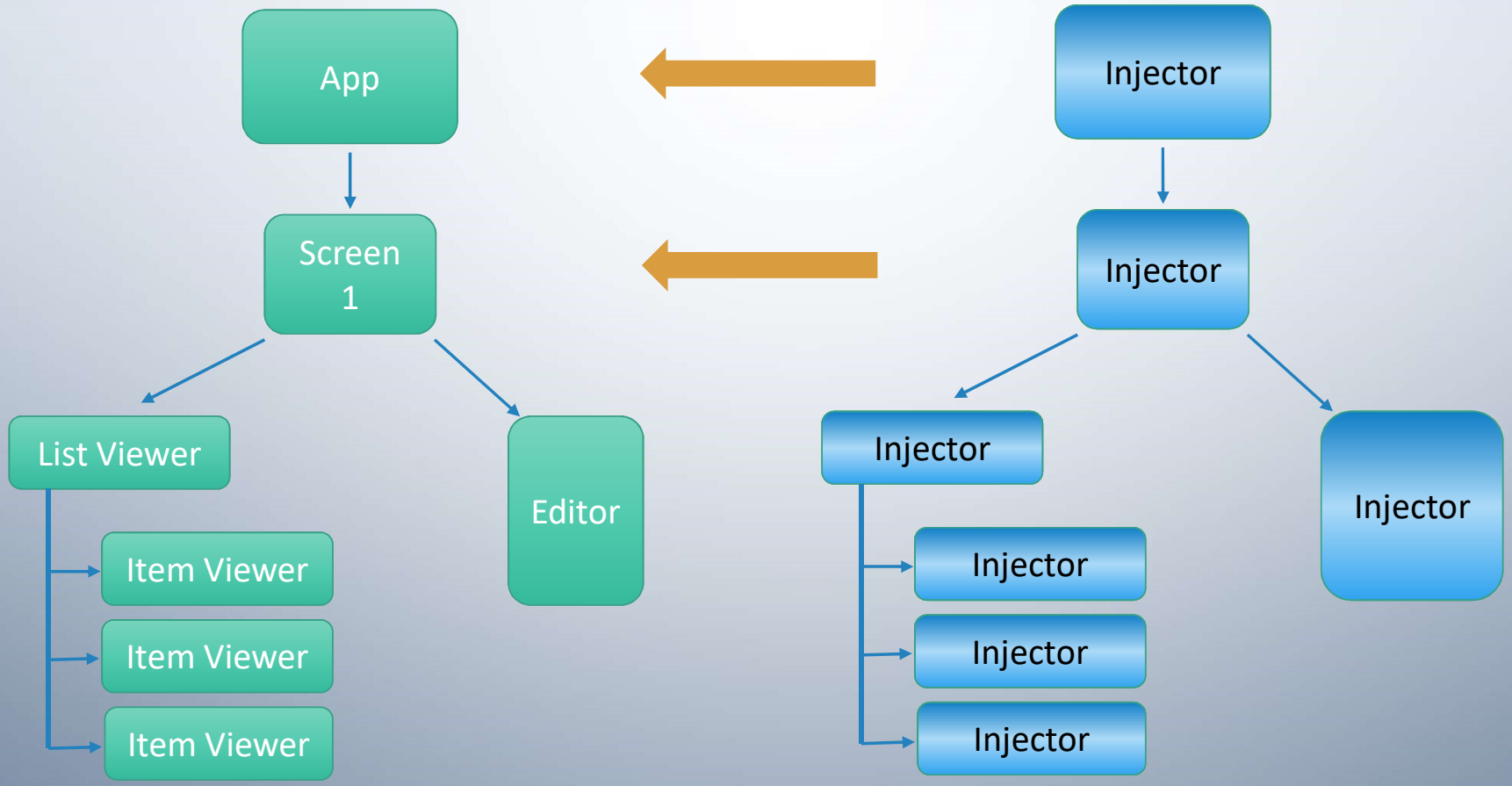



```
@Injectable()
class ContactService {

   getContacts() : Contact[] {
       … go to backend via http …
       return contacts;
   )

}
```

Dependency Injection



Dependency Injection

What can be injected?

- Anything that you provide on the component or ancestors
- Platform services: http, router
- Children in template or ancestors of a component

```
class ContactListViewer {

   @ViewChildren(ContactViewer) contactViewers   :   QueryList<ContactViewer>

}
```

- Dom Renderer

Dependency Injection

```
class ContactService {

   constructor(http:Http){ }

   getContacts() : Observable<Contact[]> {
       return http.get("http://contact.api.be/contacts");
   }

}
```

```
class ContactListViewerComponent {

   constructor(contactService:ContactService){ }

}
```

Providers

```
@Component({
       provide: [ContactService]
});
class ApplicationComponent {

}
```

or

```
@Component({
       provide: [ContactService]
});
class ContactListViewerComponent {

}
```

Exercise

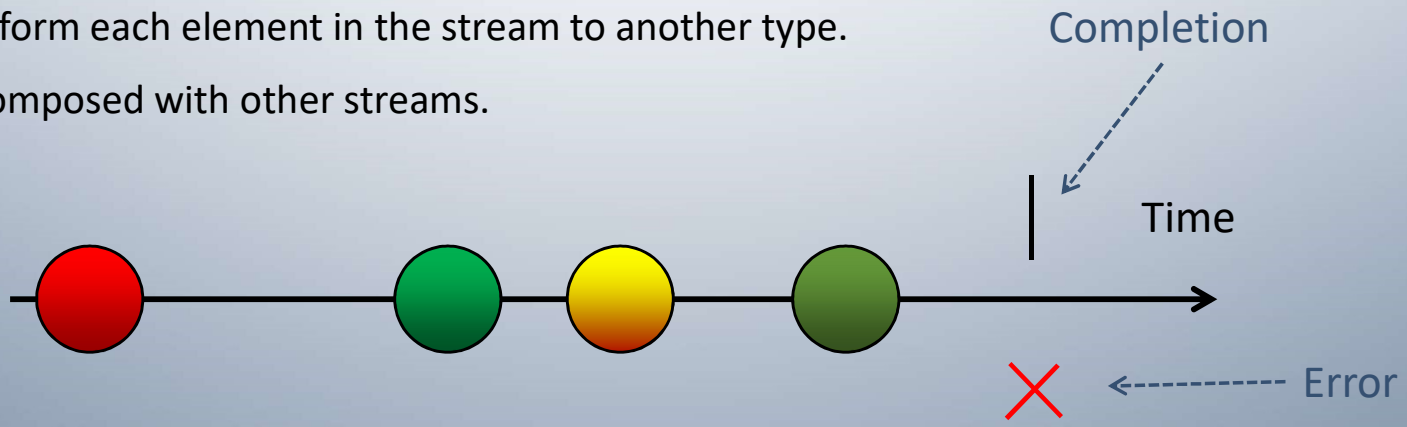
Inject the ContactService in the contact-list-viewer-initial component to get hold of the list of contacts.

Observables

Observables (ReactiveX, RXJS)

- The event system of Angular 2 is built around observables.

- An Observable<T> is a stream of data items (of type T) over time that
  - Can be observed in an asynchronous way
  - Can be transformed into another stream.
    e.g. transform each element in the stream to another type.
  - Can be composed with other streams.



Observables

- Examples
  - Stream of click events from a button.
  - Result of a HTTP call.
  - Stream of data over a web socket.
  - Stream of messages from a queue.

- Typical example
  - Autocomplete in angular 2: http://blog.thoughtram.io/angular/2016/01/06/taking-advantage-of-observables-in-angular2.html

Result of a method call

	Synchronous	Asynchronous
Many	T[]	Observable<T>
One	T	Promise<T>

Observable Subscription

```
Consider obs : Observable <T>


subscription = obs.subscribe(
      item => … handle  item … ,
      error => … handle error …,
      ()       => … handle completion …
)
```

A subscription can be canceled.

Components Revisited

Outputs

application-header.ts

```
@Component({})
class ApplicationHeaderComponent {

    search(value:string) : void {
        this.onSearch(value);
    }

    @Output()
    onSearch : EventEmitter<string>;
```

application-header.html

```
<input type="text" #searchInput>

<i (click)="search(searchInput.value)" />
```

application.html

```
<application-header  (onSearch)="performSearch($event)" />
<div>
                … application content …
</div>
```

Exercise

1. Show the contact-screen component in your application.

2. Define a contactChange Output on the contact-list-viewer component that transfers the selected contact to the contact-screen component (and displays it in json format).

Forms & Validation

Forms

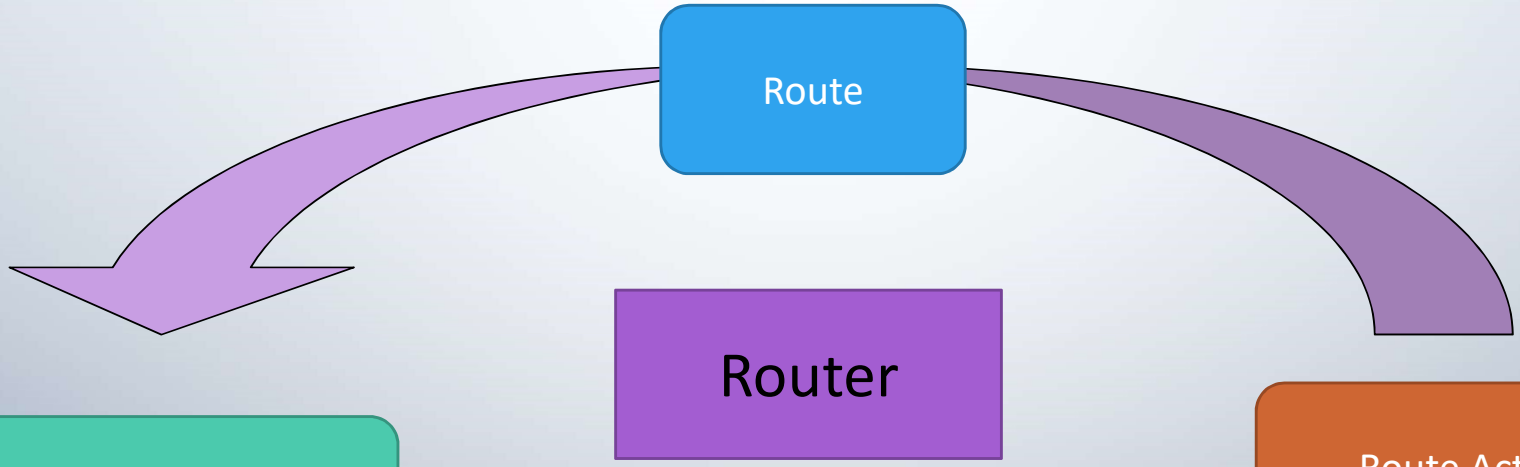
Provide

- Binding of input controls to the model
- State of the form (touched <-> untouched, dirty <-> pristine)
- Input constraint specification & validation
- Error state of the form (valid <-> invalid, errors)
- Activates style classes (ng-touched, ng-dirty, ng-invalid)

Component Routing

ROUTING

```
{ path: "contacts", component: ContactScreen }
```




```
...
<router-outlet>
...
```

target_component.html

```
...
<a routerLink="/contacts" >
...
```

Exercise

1. We are switching to the “src/with-routing” directory. In index.html, change the System.import so that we can handle routing.

2. In “application-with-routing.module.ts”, add 2 routes to applicationRoutes:
   - A route with path ‘contacts’, pointing to the ContactScreenWithRouting component
   - A route with path ‘inbox’, pointing to the InboxScreen component.

3 Add the router links to ApplicationHeaderWithRouting so that we can navigate to the 2 screens mentioned above.

4 Add a <router-outlet></router-outlet> to your application.

ROUTING

- In code

```
router.navigate(["contacts"]);
```

- URL Link

http://localhost:9001/contacts/contact/2

ROUTING

- Parameters

```
{ path: "contact/:id", component: ContactEditor }
```

```
<a routerLink="['contact',2]" >
```

```
<a routerLink="['contact',2,{name:'Don
```

http://localhost/contact/2

http://localhost/contact/2;name=Donald;age=71

- Child routes

```
{path: "contacts", component: ContactScreen,

        children : [
                { path: "contact/:id", component: ContactEditor }
        ]

}
```

Router outlet

Exercise

1. In “application-with-routing.module.ts”, add a child route to the ‘contacts’ route:
   - A ‘contact’ route, pointing to the ContactEditorWithRouting component. This route will be used to show the contact that is selected (in the list) in the editor. What does this mean for the path?
2. For each <a> in the contact-list-viewer-with-routing component, add a router link for the route defined above.
3. Add another <router-outlet></router-outlet> so that it can be filled with an editor. Where should you put it?
4. In the close-method of the ContactEditorWithRouting component, navigate back using the router.navigate method.

Http

Further Topics

- Component Lifecycle (angular.io), Change Detection (Pascal Precht), Dynamically adding components
- RXJS : reactive extensions (ReactiveX)
- Build & development tools : Node, npm, gulp
- Application Loading & Bundling : System, Webpack
- Testing
- Handy : debugger (also in ide's), console, https://augury.angular.io

Further Reading

- angular.io
- Blogs by Victor Savkin
- Blogs by Pascal Precht (e.g. change detection)
- Angular blogspot
- Angular on Github