Angular 4 - Core Concepts

(backwards compatible w/ Angular 2)

iskra – Tech talk – August 2017

Angular 4 – Getting Started

Getting Started

Angular-cli

The Angular CLI includes everything you need to start writing an Angular4 application.

- A front-end pipeline to transpile TypeScript code into JavaScript code.
- Lots of generators to scaffold your app.
- A web development server that automatically reflects your changes in the browser.

Prerequisites

Both the CLI and generated projects have dependencies that require Node 6.9.0 or higher, together with NPM 3 or higher.

Installation

Start by installing the Angular CLI using npm

```
npm install -g @angular/cli
```

Getting Started

Generating and serving an Angular project

```
ng new PROJECT-NAME
cd PROJECT-NAME
ng serve
```

* You can set the options *--style=sass* or *--style=scss* according to the desired flavor SASS/SCSS

```
ng new project --style=sass
npm install node-sass --save-dev
```

Navigate to http://localhost:4200 The app will automatically reload if you change any of the source files.

You can configure the default HTTP host and port used by the development server.

```
ng serve --host 0.0.0.0 --port 4200
```

You can build your Angular application for production environment as described below.

```
ng build --prod --env=prod --base-href http://your-domain:port
```

Getting Started (scaffolding)

Generate parts of your application (components, modules, directives, services...)

With Angular CLI you are able to (optional) create base code to avoid writing code from scratch.

```
ng g component my-new-component
ng g module my-module
ng g service my-new-service
ng g class my-new-class
ng g interface my-new-interface
ng g directive my-new-directive
ng g pipe my-new-pipe
ng g enum my-new-enum
ng g guard my-new-guard
```

Angular CLI file system

```
albert:iskra albert$ ng new demo
installing ng
  create .editorconfig
  create README.md
  create src/app/app.component.css
  create src/app/app.component.html
  create src/app/app.component.spec.ts
  create src/app/app.component.ts
  create src/app/app.module.ts
  create src/assets/.gitkeep
  create src/environments/environment.prod.ts
  create src/environments/environment.ts
  create src/favicon.ico
  create src/index.html
  create src/main.ts
  create src/styles.css
  create src/polyfills.ts
  create src/test.ts
  create src/tsconfig.app.json
  create src/tsconfig.spec.json
  create src/typings.d.ts
  create .angular-cli.json
  create e2e/app.e2e-spec.ts
  create e2e/app.po.ts
  create e2e/tsconfig.e2e.json
  create .gitignore
  create karma.conf.js
  create package.json
  create protractor.conf.js
  create tsconfig.json
  create tslint.json
Successfully initialized git.
Installing packages for tooling via npm.
Installed packages for tooling via npm.
Project 'demo' successfully created.
albert:iskra albert$
```

Project source code

Assets folder (images, fonts, etc.)

Specific constants and definitions for each project environment

Bootstrap files

Import/enable/disable here your polyfills

TypeScript configuration file

Angular CLI configuration file

Project name, version, dependencies, etc.

Angular 4 - Core Concepts

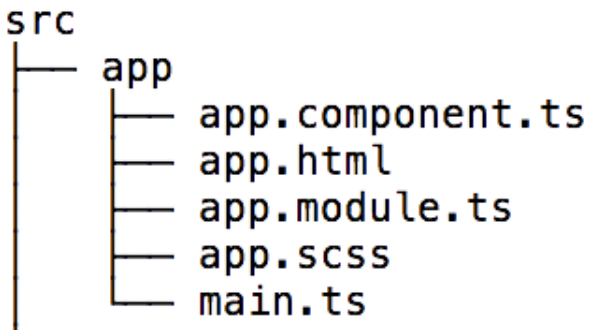
TypeScript

- Super set of JavaScript with added features.
- Created by Microsoft.
- Optional static typing.
- Class based object-oriented programming.
- Resembles languages like Java and C#

Components

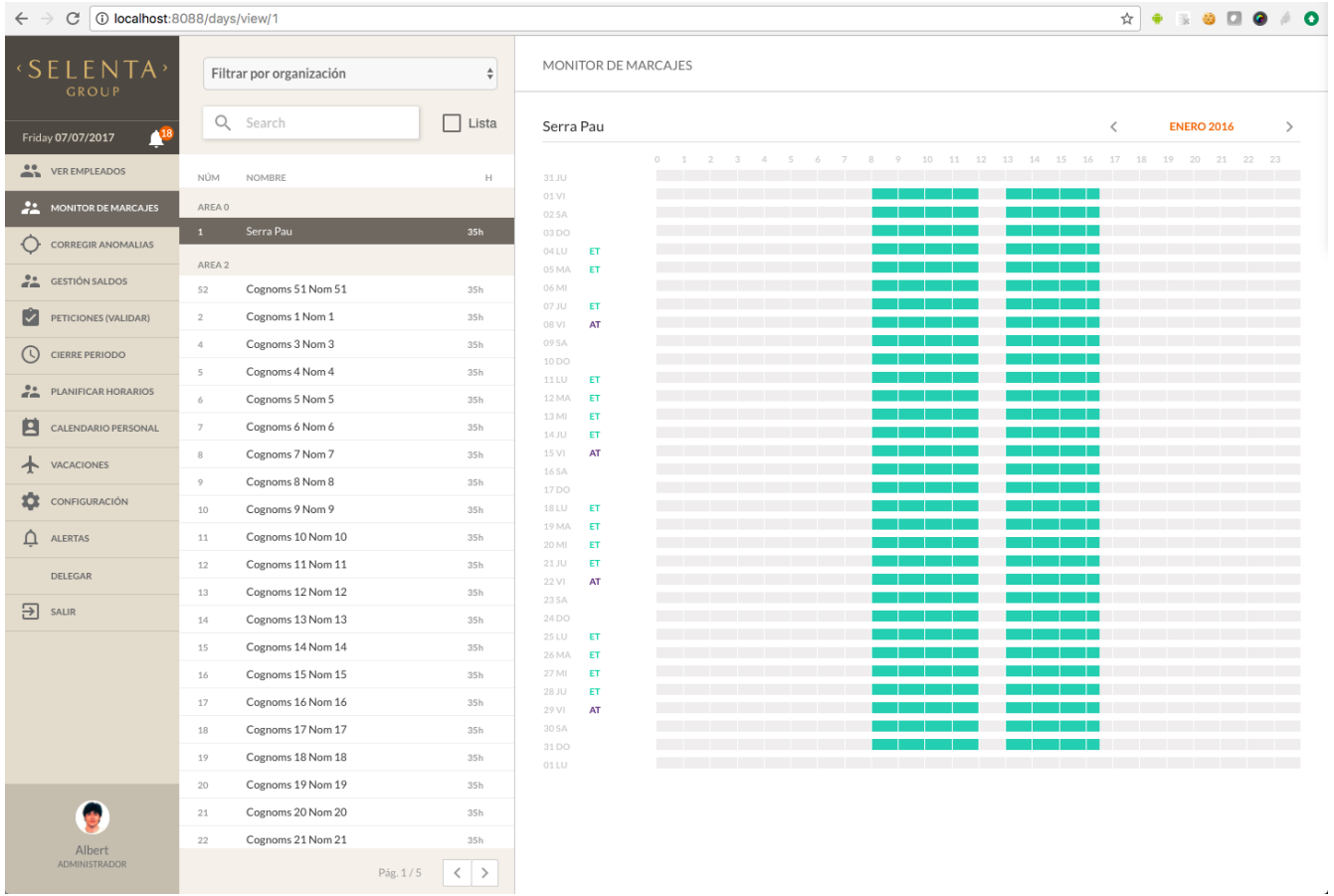
- To build an Angular application you define a set of components, for every UI element, screen, and route.

- An application will always have a root component that contains all other components.



- In other words, every Angular application will have a component tree.

Components

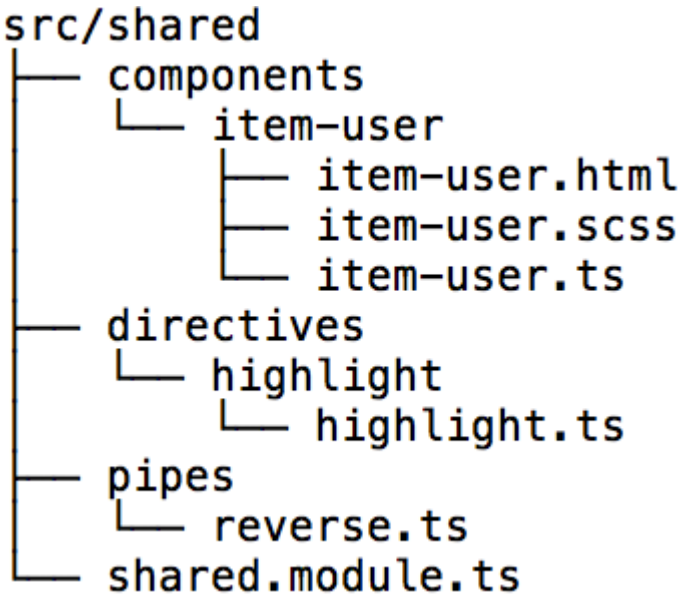

localhost:8088/days/view/1
SELENTA GROUP
Filtrar por organización
Search
Lista
Friday 07/07/2017
VER EMPLEADOS
MONITOR DE MARCAJES
CORREGIR ANOMALIAS
GESTIÓN SALDOS
PETICIONES (VALIDAR)
CIERRE PERIODO
PLANIFICAR HORARIOS
CALENDARIO PERSONAL
VACACIONES
CONFIGURACIÓN
ALERTAS
DELEGAR
SALIR
Albert
ADMINISTRADOR
NÚM
NOMBRE
H
AREA 0
1 Serra Pau 35h
AREA 2
52 Cognoms 51 Nom 51 35h
2 Cognoms 1 Nom 1 35h
4 Cognoms 3 Nom 3 35h
5 Cognoms 4 Nom 4 35h
6 Cognoms 5 Nom 5 35h
7 Cognoms 6 Nom 6 35h
8 Cognoms 7 Nom 7 35h
9 Cognoms 8 Nom 8 35h
10 Cognoms 9 Nom 9 35h
11 Cognoms 10 Nom 10 35h
12 Cognoms 11 Nom 11 35h
13 Cognoms 12 Nom 12 35h
14 Cognoms 13 Nom 13 35h
15 Cognoms 14 Nom 14 35h
16 Cognoms 15 Nom 15 35h
17 Cognoms 16 Nom 16 35h
18 Cognoms 17 Nom 17 35h
19 Cognoms 18 Nom 18 35h
20 Cognoms 19 Nom 19 35h
21 Cognoms 20 Nom 20 35h
22 Cognoms 21 Nom 21 35h
Pág. 1 / 5
MONITOR DE MARCAJES
Serra Pau
ENERO 2016

Components

Which files constitutes a single component?

- .ts
- .html *(optional)*
- .scss *(optional)*

Component scaffold
`ng g component my-new-component`



Components (.ts)

```typescript
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute, NavigationExtras, ActivatedRouteSnapshot } from '@angular/router';
import { Shift } from 'model/basemodels';
import { WorkdayManagerAPIClient, NodeWorkdayGroup }   from 'services/WorkdayManagerAPIClient';
import { HoramaticStateManager } from 'services/HoramaticStateManager';
import { HoramaticWorkdayReadonlyComponentDelegate } from 'modules/horamatic-workday-module/horamatic-workday-readonly-component/horamatic
import { HoramaticWorkdayCreateComponentDelegate } from 'modules/horamatic-workday-module/horamatic-workday-create-component/horamatic.wor
import { LoaderHud } from 'utils/LoaderHud';

export interface HoramaticWorkdayComponentDelegate {
  setWorkdayDelegate(delegate_:HoramaticWorkdayComponent);
}

@Component({
  selector: 'horamatic-workday',
  templateUrl: './horamatic.workday.component.html',
  styleUrls: ['./horamatic.workday.component.css'],
  providers: [WorkdayManagerAPIClient]
})
export class HoramaticWorkdayComponent implements OnInit, HoramaticWorkdayReadonlyComponentDelegate, HoramaticWorkdayCreateComponentDelega

  public id: string = '';
  public columns:Array<any> = [
    {title: 'JORNADAS', name: 'name', headerClassName: 'employee-list-fields-name', cellClassName: 'employee-list-item-workday'}
  ];
  public page:number = 1;
  public itemsPerPage:number = 5;
  public numPages:number = 0;
  public length:number = 0;
  public workdayManagerAPIClient: WorkdayManagerAPIClient;

  public config:any = {
    paging: true,
```

@Component decorators apply metadata on classes. When Angular will use the class, it will get this metadata to configure the expected behavior.

Components (.html)

```
<div class="context-sidebar">
  <div class="context-list-filters">
    <div class="context-list-filters-area">
      <select [(ngModel)]="selectedGroupId" [disabled]="showAsList">
        <option value=''>{{config.groupSelectorPlaceholder}}</option>
        <option *ngFor="let group of data" [value]="group.id">{{group.name}}</option>
      </select>
    </div>
    <div class="context-list-filters-search">
      <label for="content-list-filters-search">
        <i class="material-icons">search</i>
      </label>
      <input  id="content-list-filters-search"
              type="text"
              *ngIf="config.filtering"
              placeholder="Buscar"
              [(ngModel)]="searchFilter"/>

      <div class="context-list-filters-checkbox">
        <md-checkbox [(ngModel)]="showAsList" color="primary" (change)="drawWorkdaysList()" [checked]="false" class="pull-right">Lista</md-
      </div>
    </div>
  </div>
  <ngx-elastic-datatable  [config]="config"
                          [items]="data"
                          [columns]="columns"
                          [currentPage]="page"
                          [filterString]="searchFilter"
                          [filterGroupId]="selectedGroupId"
                          (didSelectedRow)="didSelectedRow($event);"
                          (didChangedTotalPages)="didChangedTotalPages($event);"
                          (didChangedPageOfSelectedRow)="didChangedPageOfSelectedRow($event)"></ngx-elastic-datatable>
  <div class="context-list-nav">
```

Components

Input and Output properties

A component has **input** and **output** properties, which can be defined using property decorators.

```
41
42      @Input() public items:Array<any> = [];
43      @Output() public onSelectRow:EventEmitter<any> = new EventEmitter();
44
```

Data flows into a component via input properties. Data flows out of a component via output properties.

Components

Input and Output properties

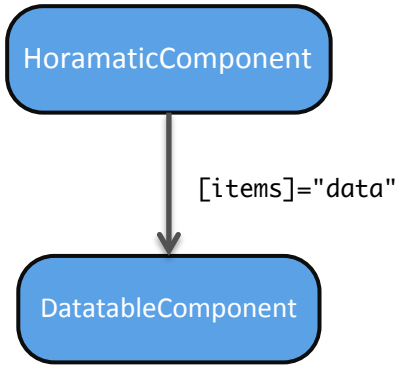
Input and output properties are the public API of a component. You use them when you instantiate a component in your application.

```
    </div>
  </div>

  <ngx-elastic-datatable  [config]="config"
                          [items]="data"
                          [columns]="columns"
                          [currentPage]="page"
                          [filterString]="searchFilter"
                          [filterGroupId]="selectedGroupId"
                          (didSelectedRow)="didSelectedRow($event);"
                          (didChangedTotalPages)="didChangedTotalPages($event);"
                          (didChangedPageOfSelectedRow)="didChangedPageOfSelectedRow($event)"></ngx-elastic-datatable>

  <div class="context-list-nav">
    <div class="orange-theme">
      <button md-icon-button class="mat-primary add-button" (click)="showAddNewWorkday()">
```

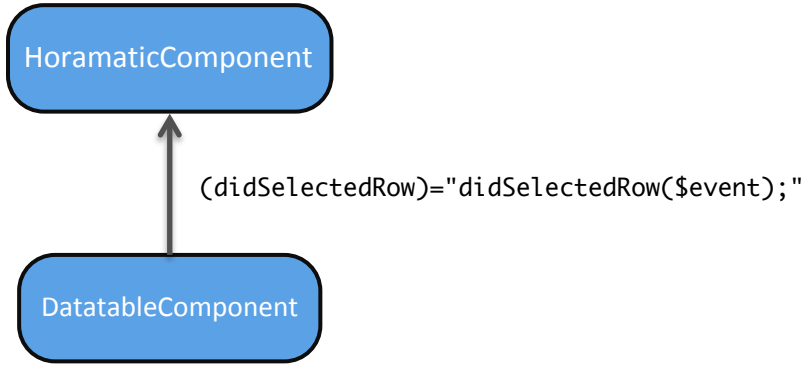
Components

Input and Output properties

Parent ➔ Child



Child ➔ Parent



Components

```
export class DatatableComponent {

  @Output() public tableChanged:EventEmitter<any> = new EventEmitter();
  @Output() public onSelectRow:EventEmitter<any> = new EventEmitter();

  @Input() public items:Array<any> = [];
  @Input()
  public set config(conf:any) {
    if (!conf.className) {
      conf.className = 'table-striped table-bordered';
    }
    if (conf.className instanceof Array) {
      conf.className = conf.className.join(' ');
    }
    this._config = conf;
  }

  @Input()
  public set columns(values:Array<any>) {
```

Angular Template Syntax

Property Bindings

Say we have a component that renders a Todo component.

```
<todo-cmp [model]="myTodo"></todo-cmp>
```

This tells Angular that whenever myTodo changes, Angular needs to automatically update the todo component by calling the model setter with a new todo.

Angular Template Syntax

Event Bindings

```
<todo-cmp
    [model]="todo"
    (complete)="onCompletingTodo(todo)">
</todo-cmp>
```

This tells Angular that if an event called "complete" is fired, it should invoke onCompletingTodo.

Angular Template Syntax

```
@Component({selector: 'todo-cmp'})
class TodoCmp {

  @Input() model;

  @Output() complete = new EventEmitter();

  onCompletedButton() {
    this.complete.next(); // this fires an event
  }

}
```

Angular Template Syntax

Two-Way Bindings

Combination of property binding and event binding.

```
<input [ngModel]="todo.text"
       (ngModelChange)="todo.text=$event">
```



```
<input [(ngModel)]="todo.text"></input>
```

Property bindings are used to pass data from the parent to the child, and event bindings are used to pass data from the child to the parent. So we can use the two to implement two-way bindings.

Angular Template Syntax

Interpolation

```
<div>Hello {{name}}</div>
```

Passing Constants

```
<show-title [title]=" 'Some Title' "></show-title>
```

References

```
<video-player #player></video-player>
<button (click)="player.pause()">Pause</button>
```

Angular Template Syntax

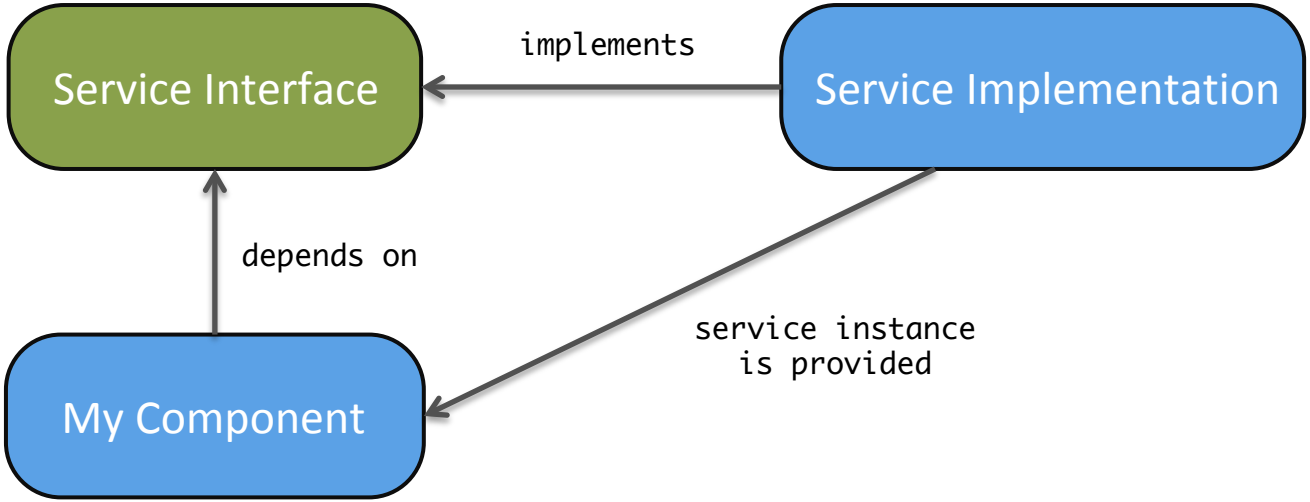
Templates and *

Templates are used to create views, chunks of DOM you can dynamically manipulate. The * syntax is a shortcut that lets you void writing the whole <template> element.

```
<todo-cmp
  *ngFor="let t of todos; let i=index"
  [model]="t"
  [index]="i">
</todo-cmp>
```

Dependency Injection

The idea behind dependency injection is simple. If you have a component that depends on a service. You do not create that service yourself. Instead, you request one in the constructor, and the framework will provide you one. By doing so you can depend on interfaces rather than concrete types.



Dependency Injection

To see how it can be used, let's look at the following component, which renders a list of talks using the for directive.

```
@Component({
  selector: 'talk-list',
  templateUrl: 'talks.html'
})
class TalkList {
  constructor() { /*get the data from service here*/ }
}
```

talks.html

```
<h2>Talks:</h2>
<div *ngFor="let t of talks"> {{t.name}} </div>
```

Dependency Injection

One approach to use a service is to create an instance of the service.

```
class TalkList {
  constructor() {
    var backend = new TalksAppBackend();
    this.talks = backend.fetchTalks(); // Return an obj mockup
  }
}
```

This is fine for a simple app, but not good for real applications. In a real application TalksAppBackend will make http requests to get the data. This means that the unit tests for this component will make real http requests instead of returning an object mockup. This problem is caused by the fact that we have coupled our component to the service.

Dependency Injection

We can solve this problem by injecting an instance of TalksAppBackend into the constructor, so we can easily replace it in tests, like this.

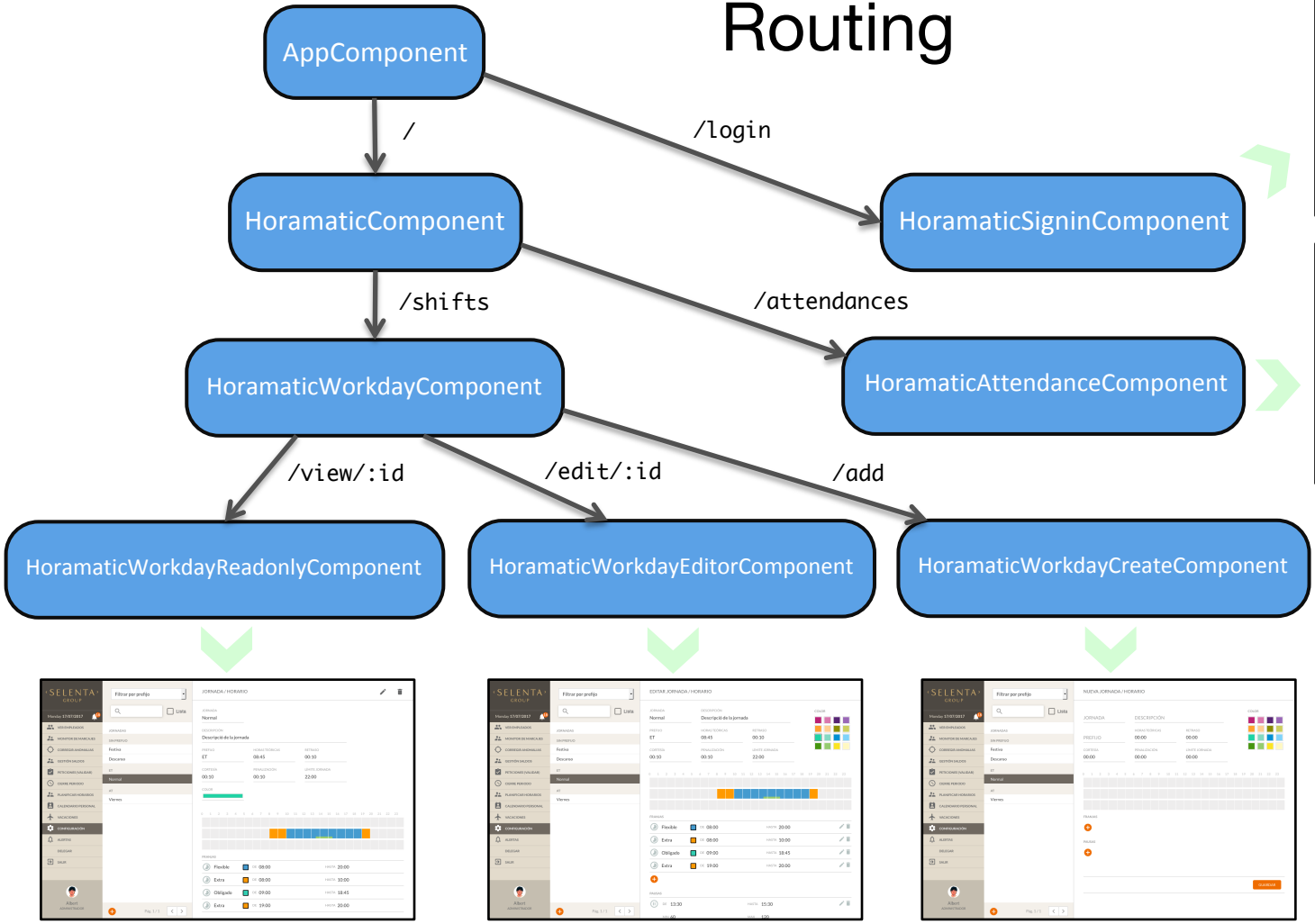
```
@Component({
  selector: 'talk-list',
  templateUrl: 'talks.html',
  providers: [TalksAppBackend]
})
class TalkList {
  constructor(backend:TalksAppBackend) {
    this.talks = backend.fetchTalks();
  }
}
```

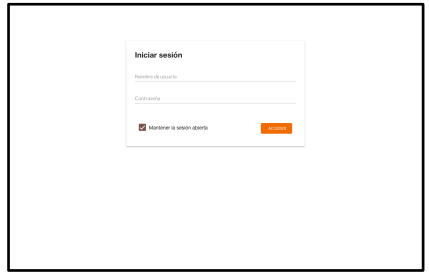
The TalksAppBackend service has to be specified in the TalkList component or its ancestor.

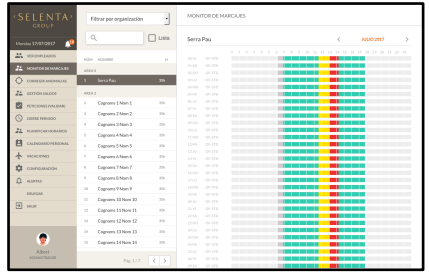
Routing

localhost/shifts/view/1

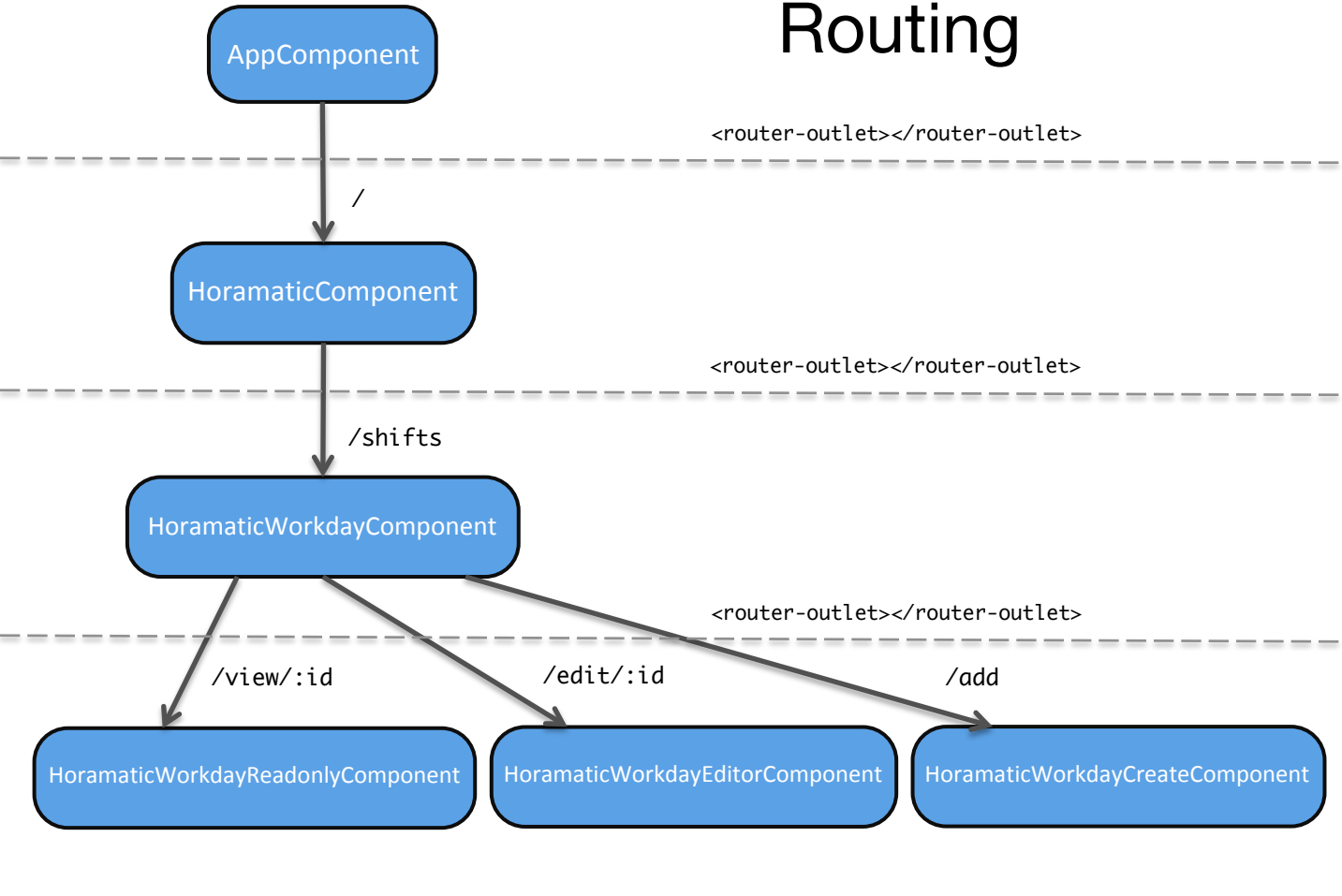
Routing

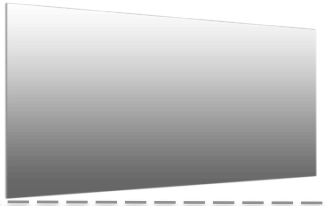

AppComponent
/
/login
HoramaticComponent
HoramaticSigninComponent
/shifts
/attendances
HoramaticWorkdayComponent
HoramaticAttendanceComponent
/view/:id
/edit/:id
/add
HoramaticWorkdayReadonlyComponent
HoramaticWorkdayEditorComponent
HoramaticWorkdayCreateComponent

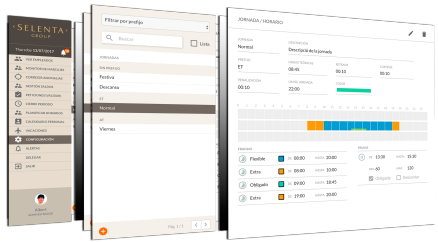

Iniciar sesión
Nombre de usuario
Contraseña
Mantener la sesión abierta
ACCEDER



Routing


AppComponent
<router-outlet></router-outlet>
/
HoramaticComponent
<router-outlet></router-outlet>
/shifts
HoramaticWorkdayComponent
<router-outlet></router-outlet>
/view/:id
/edit/:id
/add
HoramaticWorkdayReadonlyComponent
HoramaticWorkdayEditorComponent
HoramaticWorkdayCreateComponent





Routing

```
export const horamaticWorkdayRoutes: Routes = [
   {
      path: 'shifts'',
      component: HoramaticComponent,
      children: [
      {
         path: '',
         component: HoramaticWorkdayComponent,
         children: [
         {
            path: 'view/:id'',
            component: HoramaticWorkdayReadonlyComponent,
         },
         {
            path: 'edit/:id'',
            component: HoramaticWorkdayEditorComponent
         },
         {
            path: 'add'',
            component: HoramaticWorkdayCreateComponent
         }]
      }]
  }];
```

localhost/shifts/view/1