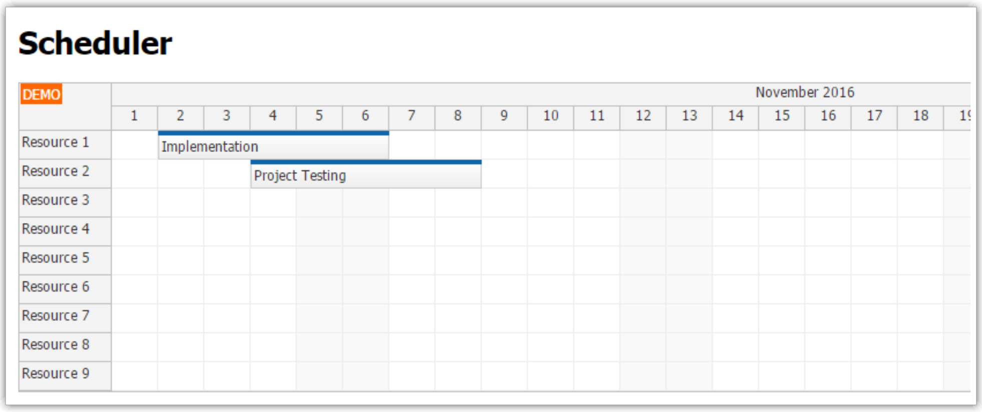
Angular 2: Scheduler UI with Spring Boot Backend (Java)

Angular 2 CLI project that shows how to create Scheduler UI using DayPilot Pro for Angular. Includes a backend REST/JSON application implemented using Spring Boot (Java).

Tags: tutorial angular2 scheduler typescript java spring boot



Downloads

- **angular2-scheduler-spring.20170222.zip** 64 kB
  Source code of the frontend (Angular 2) and backend (Spring Boot/Java) projects.

Features

This tutorial shows how to create a web application with scheduler UI using Angular 2 CLI. It uses a backend implemented in Java using Spring Boot framework.

Frontend (angular2-scheduler-spring-frontend directory):

- Angular 2 CLI project
- Scheduler UI built using DayPilot Angular 2 Scheduler
- Resources (rows) and events are loaded from the backend using REST HTTP call
- Supports event creating using drag and drop
- Event moving using drag and drop
- Includes a trial version of DayPilot Pro for JavaScript (see License below)

Backend (angular2-scheduler-spring-backend directory):

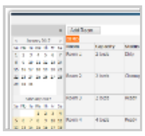
- Implemented in Java
- Uses Spring Boot framework
- Set of simple REST/JSON endpoints
- In-memory H2 database that is initialized (schema) on startup automatically
- Hibernate/JPA is used for ORM

License

Licensed for testing and evaluation purposes. Please see the license agreement included in the sample project. You can use the source code of the tutorial if you are a licensed user of DayPilot Pro for JavaScript. Buy a license (http://javascript.daypilot.org/buy/) .

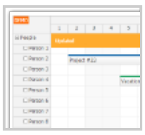
Angular 2 Frontend Project (TypeScript)

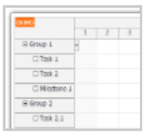
Related Articles


Angular 2 Hotel Room Booking (PHP/MySQL)
Angular 2 hotel room reservation application. The user interface supports managing rooms (create, edit, delete, change status) and reservations (create, edit, move, delete, change status). Includes a PHP/MySQL backend with token-based authentication.

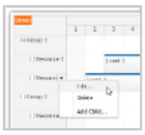
Angular 2: Appointment Calendar Component (TypeScript) + PHP/MySQL Backend
Simple appointment scheduling application built using Angular 2. The calendar view is created using DayPilot Pro Angular 2 Calendar component. The server-side backend is created using PHP and stores events in a MySQL or SQLite database.

Angular 2 Scheduler: Modal Dialog for Event Editing
Angular 2 project with Scheduler component that allows creating and editing events using a modal dialog.

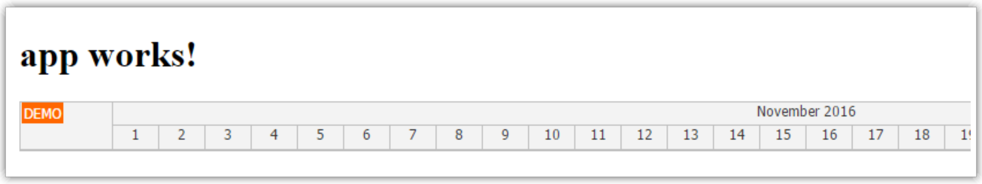

Angular 2 Scheduler Tutorial (TypeScript)
Simple Angular 2 application that shows how to use DayPilot Scheduler.

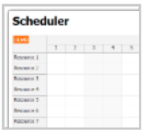

Angular 2: Gantt Chart Component (TypeScript) + PHP/MySQL Backend
Angular 2 project that displays a hierarchy of tasks using a Gantt Chart UI component. The frontend Angular 2 application is connected to a JSON backend implemented in PHP with MySQL database.

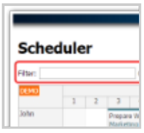
Angular 2 Scheduler: Displaying Event Phases
Angular 2 project that shows how to display events split into phases in the Scheduler component.


Angular 2 Scheduler: Resource Management
This tutorial shows how to use Angular 2 Scheduler to manage resources (create, edit, delete, move) using the Scheduler UI.

Using Scheduler with Angular 2 CLI
Adding DayPilot Scheduler UI control to a new project created using Angular 2




CLI.


Angular 2 Scheduler: Event Filtering
Angular 2 application with Scheduler component that can filter events in real time (by text, category, length).

Note: For a guide on using DayPilot Scheduler with Angular 2 CLI see also a standalone tutorial: Using Scheduler with Angular 2 CLI.

1. Create a New Angular 2 CLI Project

Create a project using Angular CLI:

```
ng new angular2-scheduler-spring-frontend
```

2. Install DayPilot Pro Angular 2 Module

Install DayPilot Pro package from npm.daypilot.org:

```
npm install https://npm.daypilot.org/daypilot-pro-angular/trial/8.3.2721.tar.gz --save
```

3. Create a New Scheduler Module

Create a new module in src/app/scheduler/scheduler.module.ts:

```
import {DataService} from "./data.service";
import {HttpModule} from "@angular/http";
import {BrowserModule} from "@angular/platform-browser";
import {NgModule} from "@angular/core";
import {SchedulerComponent} from "./scheduler.component";
import {DayPilot} from "daypilot-pro-angular";

@NgModule({
  imports:       [ BrowserModule, HttpModule ],
  declarations: [
    DayPilot.Angular.Scheduler,
    SchedulerComponent
  ],
  exports:       [ SchedulerComponent ],
  providers:     [ DataService ]
})
export class SchedulerModule { }
```

All Scheduler-related code will be located in this module. We'll minimize changes to the files generated by Angular CLI (such as app.module.ts, app.component.ts) in order to make Angular CLI version upgrade easier (new Angular CLI are released often and upgrade requires updating all generated code as well).

Note that it's necessary to import DayPilot.Angular.Scheduler from "daypilot-pro-angular" package. We also declare SchedulerComponent and DataServices - two classes that we create in the following steps.

The generated files require the following two modifications:

1. Change src/app/app.component.html as follows:

```
<scheduler-component></scheduler-component>
```

2. Import SchedulerModule in src/app/app.module.ts:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';

import { AppComponent } from './app.component';
import {SchedulerModule} from "./scheduler/scheduler.module";


@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    SchedulerModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```

```
export class AppModule { }
```

4. Create Angular 2 Scheduler Component

Create a new SchedulerComponent class in src/app/scheduler/scheduler.component.ts:

```
import {Component, ViewChild} from '@angular/core';
import {DayPilot} from "daypilot-pro-angular";

@Component({
  selector: 'scheduler-component',
  template: `
    <daypilot-scheduler></daypilot-scheduler>
  `,
  styles: [``]
})
export class SchedulerComponent {
}
```

5. Scheduler Configuration

Add "scheduler", "events" and "config" properties to SchedulerComponent class in src/app/scheduler/scheduler.component.ts:

```
import {Component, ViewChild} from '@angular/core';
import {DayPilot} from "daypilot-pro-angular";

@Component({
  selector: 'scheduler-component',
  template: `
  <div class="body">
    <h1>Scheduler</h1>
    <daypilot-scheduler [config]="config" [events]="events" #scheduler1></daypilot-schedul
  </div>
  `,
  styles: [`
  .body {
    padding: 10px;
  }
  `]
})
export class SchedulerComponent {

  @ViewChild("scheduler1")
  scheduler: DayPilot.Angular.Scheduler;

  events: any;

  config: any = {
    timeHeaders : [
      {groupBy: "Month", format: "MMMM yyyy"},
      {groupBy: "Day", format: "d"}
    ],
    days: 30,
    startDate: "2016-11-01",
    scale: "Day"
  };

}
```

If we run the Angular 2 application now using "ng serve" we will see a page with empty Scheduler control at http://localhost:4200/:

Because the backend project will run on a different port (8081) we'll add a proxy configuration that will forward local "/api" requests to the backend server (http://localhost:8081/api):

proxy.conf.json:

```
{
  "/api": {
    "target": "http://localhost:8081",
    "secure": false
  }
}
```

We need to specify the proxy configuration when running the Angular 2 CLI "serve" command:

```
ng serve --proxy-config proxy.conf.json
```

For your convenience, it's also added to package.json "start" script so you can run the development server simply by calling:

```
npm run start
```

Spring Boot Backend Project (Java)

1. Create a New Spring Boot Project

Create a new Maven project that will use org.springframework.boot:spring-boot-starter-parent project as a parent.

Add the following dependencies:

- org.springframework.boot:spring-boot-starter-web
- org.springframework.boot:spring-boot-starter-data-jpa
- com.fasterxml.jackson.datatype:jackson-datatype-jsr310
- com.h2database:h2

Our Scheduler backend project will use Hibernate and H2 database for DB persistence:

pom.xml:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSch
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.4.2.RELEASE</version>
  </parent>
  <groupId>org.daypilot.demo</groupId>
  <artifactId>angular2-scheduler-backend</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <properties>
    <java.version>1.8</java.version>
  </properties>
  <dependencies>
          <dependency>
               <groupId>org.springframework.boot</groupId>
               <artifactId>spring-boot-starter-web</artifactId>
          </dependency>
          <dependency>
             <groupId>com.fasterxml.jackson.datatype</groupId>
             <artifactId>jackson-datatype-jsr310</artifactId>
          </dependency>
          <dependency>
                <groupId>com.h2database</groupId>
                <artifactId>h2</artifactId>
                <scope>runtime</scope>
          </dependency>
          <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-data-jpa</artifactId>
          </dependency>
  </dependencies>

</project>
```

2. Create Spring Boot Application Class

Create org.daypilot.demo.angular2scheduler.Application class:

```
package org.daypilot.demo.angular2scheduler;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.domain.EntityScan;
import org.springframework.data.jpa.convert.threeten.Jsr310JpaConverters;

@EntityScan(
    basePackageClasses = { Application.class, Jsr310JpaConverters.class }
)
@SpringBootApplication
public class Application {
    public static void main(String[] args) throws Exception {
        SpringApplication.run(Application.class, args);
    }

}
```

3. Create Domain Classes (JPA/Hibernate)

Create JPA domain classes (Event and Resource classes in org.daypilot.demo.angular2scheduler.domain package):

Event.java

```
package org.daypilot.demo.angular2scheduler.domain;

import java.time.LocalDateTime;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonProperty;

@Entity
public class Event {

        @Id
        @GeneratedValue(strategy=GenerationType.AUTO)
        Long id;

        String text;

        LocalDateTime start;

        LocalDateTime end;

        @ManyToOne
        @JsonIgnore
        Resource resource;

        @JsonProperty("resource")
        public Long getResourceId() {
                return resource.getId();
        }

        public Long getId() {
                return id;
        }

        public void setId(Long id) {
                this.id = id;
        }

        public String getText() {
                return text;
        }


        public void setText(String text) {
                this.text = text;
        }

        public LocalDateTime getStart() {
                return start;
        }

        public void setStart(LocalDateTime start) {
                this.start = start;
        }

        public LocalDateTime getEnd() {
                return end;
        }

        public void setEnd(LocalDateTime end) {
                this.end = end;
        }

        public Resource getResource() {
                return resource;
        }

        public void setResource(Resource resource) {
                this.resource = resource;
        }


}
```

Resource.java

```
package org.daypilot.demo.angular2scheduler.domain;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Resource {

        @Id
        @GeneratedValue(strategy=GenerationType.AUTO)
        Long Id;

        String name;

        public Long getId() {
                return Id;
        }

        public void setId(Long id) {
                this.Id = id;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }


}
```

4. Configure H2 Database (In-Memory)

Create application.properties file in "src/main/resource" and add the following properties:

```
spring.datasource.url=jdbc:h2:mem:mydb
spring.h2.console.enabled=true
spring.jpa.hibernate.ddl-auto=create

spring.jackson.serialization.write_dates_as_timestamps=false
server.port=${port:8081}
```

This configuration will create an in-memory H2 database (called "mydb") on application startup and automatically create the database schema from the domain classes ("spring.jpa.hibernate.ddl-auto" property).

The "spring.h2.console.enabled" property enables the built-in H2 database console which you can use to manage the database (http://localhost:8081/h2console).

We have also added "spring.jackson.serialization.write_dates_as_timestamps" property which will fix date object JSON serialization (see below).

The "server.port" propety changes the default 8080 port to 8081 to avoid conflicts with a local Tomcat server installation.

5. Initialize the Database with Sample Resource Data

We will initialize the database with some data using data.sql file (src/main/resources directory):

```
insert into resource (name) values ('Resource 1');
insert into resource (name) values ('Resource 2');
insert into resource (name) values ('Resource 3');
```

6. Create the DAO Classes

Create the repository (data access) classes in org.daypilot.demo.angular2scheduler.repository package:

ResourceRepository.java

```
package org.daypilot.demo.angular2scheduler.repository;

import org.daypilot.demo.angular2scheduler.domain.Resource;
import org.springframework.data.repository.CrudRepository;

public interface ResourceRepository extends CrudRepository<Resource, Long> {
}
```

EventRepository.java

```
package org.daypilot.demo.angular2scheduler.repository;

import org.daypilot.demo.angular2scheduler.domain.Event;
import org.springframework.data.repository.CrudRepository;
```

```
public interface EventRepository extends CrudRepository<Event, Long> {
}
```

7. Create the Controller with REST/JSON Endpoints

Create a MainController class in org.daypilot.demo.angular2scheduler.controller package:

```
package org.daypilot.demo.angular2scheduler.controller;

import org.daypilot.demo.angular2scheduler.domain.Event;
import org.daypilot.demo.angular2scheduler.domain.Resource;
import org.daypilot.demo.angular2scheduler.repository.EventRepository;
import org.daypilot.demo.angular2scheduler.repository.ResourceRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import com.fasterxml.jackson.datatype.jsr310.ser.LocalDateTimeSerializer;

@RestController
public class MainController {

    @Autowired
    EventRepository er;

    @Autowired
    ResourceRepository rr;

    @RequestMapping("/api")
    @ResponseBody
    String home() {
        return "Welcome!";
    }

}
```

8. Test

Now you can run the application and test the REST API using http://localhost:8081/api. It returns the welcome string:

```
Welcome!
```

Integrating Angular 2 Scheduler Application with Spring Boot Backend

1. DataService Class for Communication with the Backend

First we will create a helper DataService that will make calls to the backend JSON API and return the results using an Observable.

The empty DataService will look like this:

```
import { Http, Response } from '@angular/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/Rx';
import 'rxjs/Rx';
import {DayPilot} from 'daypilot-pro-angular';

@Injectable()
export class DataService {

  constructor(private http : Http){
  }

}
```

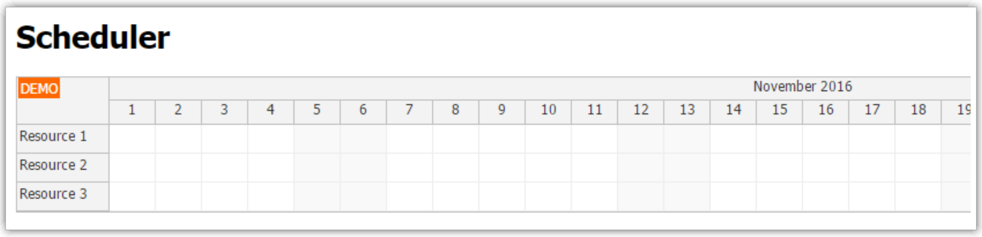
We need to register the DataService class as a provider in scheduler.module.ts:

```
// ...

@NgModule({
  // ...
  providers: [
    DataService
  ],
  // ...
})

// ...
```

We will also ask for an instance of DataService to be injected into SchedulerComponent class (scheduler.component.ts) so we can use it:

```
// ...
export class SchedulerComponent {

  // ...

  constructor(private ds: DataService) {}

  // ...

}
```

2. Loading Scheduler Resources



We want to load the Scheduler rows (resources) as soon as the SchedulerComponent is displayed.

```
package org.daypilot.demo.angular2scheduler.controller;

import org.daypilot.demo.angular2scheduler.domain.Event;
import org.daypilot.demo.angular2scheduler.domain.Resource;
import org.daypilot.demo.angular2scheduler.repository.EventRepository;
import org.daypilot.demo.angular2scheduler.repository.ResourceRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import com.fasterxml.jackson.datatype.jsr310.ser.LocalDateTimeSerializer;

@RestController
public class MainController {

    @Autowired
    EventRepository er;

    @Autowired
    ResourceRepository rr;

    @RequestMapping("/")
    @ResponseBody
    String home() {
        return "Welcome!";
    }

    @RequestMapping("/api/resources")
    Iterable<Resource> resources() {
        return rr.findAll();
    }

}
```

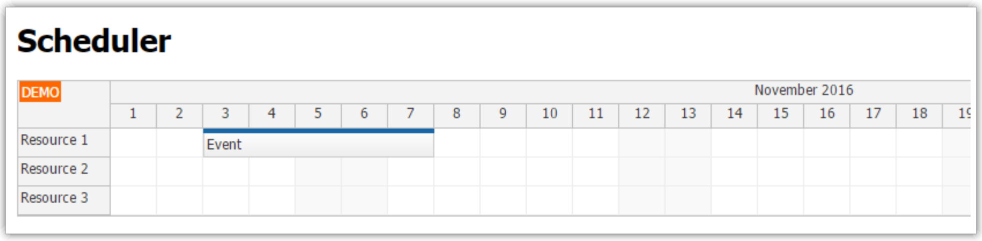
The new endpoint ("/api/resources") returns an array of resources in JSON format:

```
[{"name":"Resource 1","id":1},{"name":"Resource 2","id":2},{"name":"Resource 3","id":3}]
```

Now we want to request the resource data using the Angular 2 frontend and pass it to the Scheduler:

```
import {Component, ViewChild, AfterViewInit} from '@angular/core';
import {DayPilot} from "daypilot-pro-angular";
import {DataService} from "./data.service";

@Component({
  selector: 'scheduler-component',
  template: `
  <div class="body">
    <h1>Scheduler</h1>
    <daypilot-scheduler [config]="config" [events]="events" #scheduler1></daypilot-schedul
  </div>
  `,
  styles: [`.body { padding: 10px; }`]
})
export class SchedulerComponent implements AfterViewInit {

  // ...

  constructor(private ds: DataService) {}

  ngAfterViewInit(): void {
    this.ds.getResources().subscribe(result => this.config.resources = result);
  }
}
```

3. Loading Scheduler Events



In order to load the event data from the server we will add events() method to MainController class.

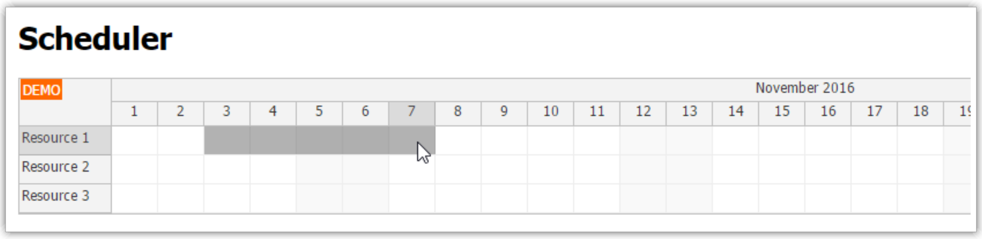
This method will be mapped to "/api/events" endpoint. It requires the data range to be specified using "from" and "to" query string parameters ("/api/events?from=2016-11-01T00:00:00&to=2016-11-01T00:00:00").

```
package org.daypilot.demo.angular2scheduler.controller;

import java.time.LocalDateTime;

import javax.transaction.Transactional;

import org.daypilot.demo.angular2scheduler.domain.Event;
import org.daypilot.demo.angular2scheduler.domain.Resource;
import org.daypilot.demo.angular2scheduler.repository.EventRepository;
import org.daypilot.demo.angular2scheduler.repository.ResourceRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.format.annotation.DateTimeFormat.ISO;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import com.fasterxml.jackson.datatype.jsr310.ser.LocalDateTimeSerializer;

@RestController
public class MainController {

        @Autowired
        EventRepository er;

        @Autowired
        ResourceRepository rr;

    @RequestMapping("/api")
    @ResponseBody
    String home() {
          return "Welcome!";
    }

    @RequestMapping("/api/resources")
    Iterable<Resource> resources() {
        return rr.findAll();
    }

    @GetMapping("/api/events")
    @JsonSerialize(using = LocalDateTimeSerializer.class)
    Iterable<Event> events(@RequestParam("from") @DateTimeFormat(iso=ISO.DATE_TIME) LocalD
        return er.findBetween(from, to);
    }
}
```

Angular 2:

```
import {Component, ViewChild, AfterViewInit} from '@angular/core';
import {DayPilot} from "daypilot-pro-angular";
import {DataService} from "./data.service";

@Component({
  selector: 'scheduler-component',
  template: `
  <div class="body">
    <h1>Scheduler</h1>
    <daypilot-scheduler [config]="config" [events]="events" #scheduler1></daypilot-schedul
  </div>
  `,
  styles: [`.body { padding: 10px; }`]
})
export class SchedulerComponent implements AfterViewInit {

  @ViewChild("scheduler1")
  scheduler: DayPilot.Angular.Scheduler;

  // ...

  constructor(private ds: DataService) {}

  ngAfterViewInit(): void {
    this.ds.getResources().subscribe(result => this.config.resources = result);

    var from = this.scheduler.control.visibleStart();
    var to = this.scheduler.control.visibleEnd();
    this.ds.getEvents(from, to).subscribe(result => this.events = result);
  }
}
```

4. Creating Events using Drag and Drop



We will handle the onTimeRangeSelected UI event of the Scheduler to create a new event. But first, we need to create the JSON endpoint in the backend.

The event handler is specified using onTimeRangeSelected property of the config object.

- It displays a simple prompt dialog to get the new event name.
- It calls /api/events/create endpoint to store the new event. The endpoint returns event data object.
- We wait until the new event data object is returned and we add it to the "events" array.
- The Scheduler displays it as soon as the change of "events" is detected.

Angular 2 Frontend: SchedulerComponent (scheduler.component.ts)

```
import {Component, ViewChild, AfterViewInit} from '@angular/core';
import {DayPilot} from "daypilot-pro-angular";
import {DataService, CreateEventParams} from "./data.service";

@Component({
  selector: 'scheduler-component',
  template: `
  <div class="body">
    <h1>Scheduler</h1>
    <daypilot-scheduler [config]="config" [events]="events" #scheduler1></daypilot-schedul
  </div>
  `,
  styles: [`.body { padding: 10px; }`]
})
export class SchedulerComponent implements AfterViewInit {

  // ...

  @ViewChild("scheduler1")
  scheduler: DayPilot.Angular.Scheduler;

  config: any = {
    timeHeaders : [
      {groupBy: "Month", format: "MMMM yyyy"},
      {groupBy: "Day", format: "d"}
    ],
    days: 30,
    startDate: "2016-11-01",
    scale: "Day",
    onTimeRangeSelected: args => {
      let name = prompt("New event name:", "Event");
      this.scheduler.control.clearSelection();
      if (!name) {
        return;
      }
      let params: CreateEventParams = {
        start: args.start.toString(),
        end: args.end.toString(),
        text: name,
        resource: args.resource
      };
      this.ds.createEvent(params).subscribe(result => {
        this.events.push(result);
        this.scheduler.control.message("Event created");
      } );
    }
  };

  // ...

}
```

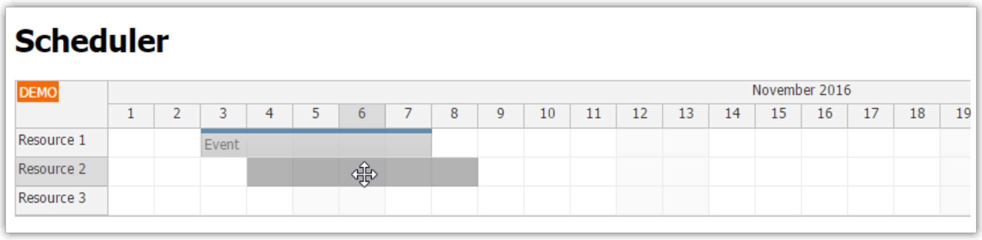
Angular 2 Frontend: DataService class (data.service.ts)

```
import { Http, Response } from '@angular/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/Rx';
import 'rxjs/Rx';
import {DayPilot} from 'daypilot-pro-angular';

@Injectable()
export class DataService {

  constructor(private http : Http){
  }

  // ...

  createEvent(data: CreateEventParams): Observable<EventData> {
    return this.http.post("/api/events/create", data).map((response:Response) => response.
  }

}

export interface CreateEventParams {
  start: string;
  end: string;
  text: string;
  resource: string | number;
}

export interface EventData {
  id: string | number;
  start: string;
  end: string;
  text: string;
  resource: string | number;
}
```

Spring Boot Backend: MainController.java

```
package org.daypilot.demo.angular2scheduler.controller;

// ...

@RestController
public class MainController {

        @Autowired
        EventRepository er;

        @Autowired
        ResourceRepository rr;


    // ...
    @PostMapping("/api/events/create")
    @JsonSerialize(using = LocalDateTimeSerializer.class)
    @Transactional
    Event createEvent(@RequestBody EventCreateParams params) {

        Resource r = rr.findOne(params.resource);

        Event e = new Event();
        e.setStart(params.start);
        e.setEnd(params.end);
        e.setText(params.text);
        e.setResource(r);

        er.save(e);

        return e;
    }


    public static class EventCreateParams {
        public LocalDateTime start;
              public LocalDateTime end;
              public String text;
                public Long resource;
    }

}
```

5. Drag and Drop Event Moving



Event moving is enabled by default in the Scheduler.

- We need to handle onEventMove event and notify the server about the new location.
- This time we don't update the event data in "events" array. It will be updated automatically (the default eventMoveHandling action is set to "Update").

Angular 2 Frontend: SchedulerComponent

```
import {Component, ViewChild, AfterViewInit} from '@angular/core';
import {DayPilot} from "daypilot-pro-angular";
import {DataService, CreateEventParams, MoveEventParams} from "./data.service";

@Component({
  selector: 'scheduler-component',
  template: `
  <div class="body">
    <h1>Scheduler</h1>
    <daypilot-scheduler [config]="config" [events]="events" #scheduler1></daypilot-schedul
  </div>
  `,
  styles: [`.body { padding: 10px; }`]
})
export class SchedulerComponent implements AfterViewInit {

  @ViewChild("scheduler1")
  scheduler: DayPilot.Angular.Scheduler;

  // ..

  config: any = {
    // ...
    onEventMove: args => {
      let params: MoveEventParams = {
        id: args.e.id(),
        start: args.newStart.toString(),
        end: args.newEnd.toString(),
        resource: args.newResource
      };
      this.ds.moveEvent(params).subscribe(result => {
        this.scheduler.control.message("Event moved");
      });
    }
  };
  constructor(private ds: DataService) {}

  // ...

}
```

Angular 2 Frontend: DataService

```
import { Http, Response } from '@angular/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/Rx';
import 'rxjs/Rx';
import {DayPilot} from 'daypilot-pro-angular';

@Injectable()
export class DataService {

  constructor(private http : Http){
  }

  // ...

  moveEvent(data: any): Observable<EventData> {
    return this.http.post("/api/events/move", data).map((response:Response) => response.js
  }

}

export interface MoveEventParams {
  id: string | number;
  start: string;
  end: string;
  resource: string | number;
}

export interface EventData {
  id: string | number;
  start: string;
  end: string;
  text: string;
  resource: string | number;
}
```

Spring Boot Backend: MainController.java

```
package org.daypilot.demo.angular2scheduler.controller;

// ...

@RestController
public class MainController {

       @Autowired
       EventRepository er;

       @Autowired
       ResourceRepository rr;

    // ...

    @PostMapping("/api/events/move")
    @JsonSerialize(using = LocalDateTimeSerializer.class)
    @Transactional
    Event moveEvent(@RequestBody EventMoveParams params) {

       Event e = er.findOne(params.id);
       Resource r = rr.findOne(params.resource);

       e.setStart(params.start);
       e.setEnd(params.end);
       e.setResource(r);

       er.save(e);

       return e;
    }

    public static class EventMoveParams {
      public Long id;
      public LocalDateTime start;
         public LocalDateTime end;
           public Long resource;
    }
}
```

Spring Boot Gotchas

Enable java.time.* classes in Hibernate

In order to handle the Java 8 DateTime objects properly in the domain classes it's necessary to add Jsr310JpaConverters.class to @EntityScan annotation of the Application class:

```
@EntityScan(
     basePackageClasses = { Application.class, Jsr310JpaConverters.class }
)
```

Without this setting, the LocalDateTime fields of domain classes won't be created as TIMESTAMP in the database.

Serialize the resource reference as plain id in JSON

We are using the original domain classes for JSON serialization so we need to flatten the structure - replace the Resource reference with a resource id.

Original (Resource.java):

```
        @ManyToOne
        Resource resource;
```

Updated (Resource.java):

```
        @ManyToOne
        @JsonIgnore
        Resource resource;

        @JsonProperty("resource")
        public Long getResourceId() {
                return resource.getId();
        }
```

Without this setting, the resource id

Serialize the dates in ISO format in JSON

In order to serialize DateTime objects to JSON properly (as a ISO 8601 string) we need to add *com.fasterxml.jackson.datatype:jackson-datatype-jsr310* package as a dependency and add the following property to application.properties:

```
spring.jackson.serialization.write_dates_as_timestamps=false
```

Share

Related Questions

Servers (2 replies) asked by Sergey on Jan 28, 2017
angular2-scheduler-spring-frontend npm start (1 reply) asked by Roger on Dec 16, 2016
See all related questions (http://forums.daypilot.org/tagged/article-61900-c) [forums.daypilot.org]

Ask a Question (http://forums.daypilot.org/question/new?code=2FiB7q61dYYXcQsQSCL2YNAWIRyRjjHYxv5heluSABGcKWUilcT1PDykmO143yRqNoceJ%2FOS3s9MthQa%2FhmC48dX8WnZ5sxAGubeJ%2FVhwSrFbAfQB5TiGpR%2BrTf7Hh%2Bj9iP%2FPoolLjU2CM5QpPMu4kqxTGKTvNOj8pGbG