Task 7. Sections

Task description

In this task you have to create sections for notes, with possibility for reordering. Also we will use Bootstrap to create a nice looking design of the application.

Time

2 hours

Detailed description

7.1 Add bootstrap css

We will be using Twitter bootstrap for design of our Notes application.

1) Create **css** folder in project root, copy **bootstrap.min.css** from **bootstrap-init** to **css** folder
2) Copy folder **fonts** from **bootstrap-init** to the project **root** folder

3) Add link to bootstrap to **index.html**:

```
<link rel="stylesheet" href="css/bootstrap.min.css">
```

4) Use **app.component.html** as a template for **AppComponent** (put template to app folder, add templateUrl to component)

5) Use **notes.component.html** as a template for **NotesComponent**

6) Add empty component SectionsComponent. Set selector to 'sections', leave template empty.
Refresh page to see how it is looking with the bootstrap design.

7.2 Show sections

1) Add the sections collection to server.js:

```
db.collection('sections', function(error, sections) {
        db.sections = sections;
});
```

2) Add possibility to get sections:

```
app.get("/sections", function(req,res) {
db.sections.find(req.query).toArray(function(err, items) {
        res.send(items);
});
});
```

3) Update component SectionsComponent which should use sections.component.html as a template (put template to app folder, add templateUrl to component).

4) Define Section interface:

```
interface Section {
  _id: string;
  title: string;
}
```

5) Define sections url:

```
private sectionsUrl = 'sections';  // URL to web api
```

6) Define list of sections:

```
sections: Section[];
```

7) Define readSections() and getSections() method in SectionsComponent:

```
readSections() {
  this.getSections().subscribe(sections=>{
    this.sections=sections;
  });
}

getSections(): Observable<Section[]> {
  return this.http.get(this.sectionsUrl)
    .map(response => response.json() as Section[]);
}
```

Also add map operator to import:

```
import 'rxjs/add/operator/map';
```

Or you can import all Observable operators at once by using

```
import 'rxjs/Rx';
```

8) Add constructor which will show list of sections on startup:

```
constructor(private http: Http) {
  this.readSections();
}
```

9) Run **mongo** and add some sections:

```
db.sections.insert({title:"Work"});
db.sections.insert({title:"Vacations"});
db.sections.insert({title:"Children"});
```

10) Also add test note to Work section (will be used later for testing):

```
db.notes.insert({section:"Work", text:"test work note"});
```

Now refresh the page and see the new design with the sections on the right.

7.3 Allow to select a section and show notes for the section

1) Update sections.component.html to react on section click:

```
<li *ngFor="let section of sections"
   [ngClass]="{active:section.title==activeSection }"
   (click)="showSection(section)" class="list-group-item">
   {{section.title}}
</li>
```

2) Add activeSection field of type string. It will keep the currently selected section.

3) Add showSection() to SectionsComponent:

```
showSection(section:Section) {
   this.activeSection = section.title;
}
```

4) In readSections() set default activeSection if it was not set:

```
if (this.activeSection == null && this.sections.length>0) {
   this.showSection(this.sections[0]);
}
```

Add this inside subscribe callback.

5) Reload page and try to switch active section.

7.4 Show notes for the selected section

1) Add field section:string to NotesComponent. It will be the title of active section. Set its initial value to "Work".

2) Update add() to keep the section with the new note:

```
let note = { text: this.text, section: this.section };
```

3) Update getNotes() to retrieve notes only for the selected section:

```
let params: URLSearchParams = new URLSearchParams();
params.set('section', this.section);
return this.http.get(this.notesUrl, {search:params})
      .map(response => response.json() as Note[]);
```

Also update getNotes signature from Promise to Observable,
and update readNotes to use subscribe instead of then.

4) Reload page. Now it should add notes to section Work and show notes only from this section(since it is hardcoded).

7.5 Implement communication of SectionsComponent with NotesComponent

To reflect change of the active section, we should update section in NotesComponent and reread sections on every change of active section in SectionsComponent. We will use AppComponent for communication.

To do this:
1) Pass section title to NotesComponent in app.component.html:

```
<notes [section]="'Work'"></notes>
```

2) Add @Input() decorator to section field in NotesComponent

```
@Input() section: string;
```

3) Since @Input property will be not available in component constructor, move reading notes from constructor to ngOnInit() – here we will already know the section passed from AppComponent. Also add OnInit interface:

```
class NotesComponent  implements OnInit
```

4) Now reload page. It should work the same way, showing notes for "Work" section.

5) To pass section name from SectionsComponent, introduce variable section in AppComponent – it will behave as mediator:

```
section: string;
```

Also define this method in AppComponent:

```
setSection(section:string) {
   this.section = section;
}
```

6) Update app.component.html by adding event handler to <sections>:

```
<sections (sectionChanged)="setSection($event)"></sections>
```

Now on every change of section setSection() method will be fired with title of the section as a parameter.

7) Add @Output field to SectionsComponent to emit event:

```
@Output() sectionChanged: EventEmitter<string> =
        new EventEmitter<string>();
```

8) Update showSection() so that it emits the event on every change of the section:

```
showSection(section:Section) {
   this.activeSection = section.title;
   this.sectionChanged.emit(this.activeSection);
}
```

9) In app.component.html pass remove hardcoded "Work" section and pass section field instead:

```
<notes [section]="section"></notes>
```

10) Also change method name ngOnInit() to ngOnChanges() in NotesComponent and add implements OnChange. It will allow execute this method and update notes list not only on component initialization, but also on every change of the section.

11) Now update all your notes to put it to some section, because notes with no sections are not supported anymore:

Run mongo and execute

db.notes.update({}, {$set: {section:"Old notes" }}, {multi:true});

Also add section "Old notes" to sections collection.

12) Reload page and check that now we can select section, see the notes of the selected section and add the note to the active section.

7.6 Add section

Now we should allow user to add a new section from the UI.

1) In sections.component.html update input box to enter new section name:

```
<input type="text" class="form-control" placeholder="New section name" #newSection>
```

2) Update button Add to process click:

```
<button class="btn btn-default" type="button" (click)="addSection(newSection)">Add</button>
```

3) Now that's possible the section has no _id (just added section has no id), so we should update Section interface:

```
export interface Section {
  _id?: string;
  title: string;
}
```

_id is now optional. Also add **export** because we will need to use it from outside.

4) Add method addSection() to SectionsComponent:

```
addSection(newSection: HTMLInputElement) {
  let title = newSection.value;
  if (!title) return;

  // check for duplicates
  if (this.sections.map(s=>s.title).find(t=>t===title)) return;

  const section: Section = { title };
  this.sections.unshift(section);
  this.showSection(section);

  // write sections to server and clear add section input box
  this.writeSections().subscribe(res=>newSection.value = "");
}
```

Method unshift will add section to the beginning of sections list.
Also we will switch to the newly added section.

5) Add writeSections() method which will send the list of sections to the server:

```
writeSections() {
  return this.http.post(this.sectionsReplaceUrl, this.sections);
}
```

Also define sectionsReplaceUrl in component:

```
sectionsReplaceUrl = "/sections/replace";
```

6) Add possibility to replace old section list by new list in server.js. It will be used for adding and reordering of the sections.

```
app.post("/sections/replace", function(req,resp) {
// do not clear the list
if (req.body.length==0) {
        resp.end();
}
db.sections.remove({}, function(err, res) {
        if (err) console.log(err);
        db.sections.insert(req.body, function(err, res) {
                if (err) console.log("err after insert",err);
                resp.end();
        });
});
});
```

Now you can refresh page and check the adding of the new section.

7.7 Add sections reordering [optional]

We will be using library Dragula which implements the possibility to reorder list by drag&drop. Here you can find the full description of the library:
https://github.com/valor-software/ng2-dragula

To add this, do the following:
1) Copy **dragula.css** from **bootstrap.min.css** to **css** folder

2) Add dragon libraries to package.json:
```
"dragula": "^3.7.2",
"ng2-dragula": "^1.2.2-0"
```
Then run npm install.

3) Configure systemjs.config.js:
Add to map {…} section these lines:
```
'ng2-dragula': 'npm:ng2-dragula',
'dragula':     'npm:dragula/dist/dragula.js'
```
Also add defaultJSExtensions to System.config root:
```
System.config({
  defaultJSExtensions: "js",
 ...
```

4) Edit app.module.ts: add DragulaModule to imports:
```
imports:      [ ..., DragulaModule ],
```
Also add this import to imports in app.module.ts:
```
import {DragulaModule} from "ng2-dragula";
```
5) Update sections list by adding dragula directive in sections.component.html:
```
<ul class="list-group" [dragula]='"sections"'>
```

6) Update constructor in SectionsComponent to subscribe to drop event:
```
constructor(private http: Http, private dragulaService: DragulaService) {
   this.readSections();
   dragulaService.drop.subscribe(this.onDrop.bind(this));
}
```
Add this import to SectionsComponent (**NB!** WebStorm may suggest the wrong import path):
```
import {DragulaService} from "ng2-dragula";
```

7) Implement onDrop method in SectionsComponent:
```
onDrop(value) {
   let [bag, elementMoved, targetContainer, srcContainer] = value;
   if (targetContainer.children) {
      let arr = Array.from(targetContainer.children);
      this.sections = arr.map((li:HTMLLIElement)=>
        { return {title: li.textContent.trim() } });
      this.writeSections().subscribe();
   }
}
```

And update method addSection to subscribe.
Now you can refresh the page and check how section drag&drop is working.

7.8 Add sections filter [optional]

We will add the input field to filter out sections which title starts with the entered value. For this we will create pipe.

1) Create pipe SectionFilterPipe with this code:

```
@Pipe({
  name: 'sectionFilter'
})
export class SectionFilterPipe implements PipeTransform {
  transform(sections: Section[], v: string):Section[] {
    if (!sections) return [];
    return sections.filter(
      s => s.title.toLowerCase().startsWith(v. toLowerCase()));
  }
}
```

2) Add this pipe to AppModule declarations

3) Update sections.component.html: add input box for the section filter:

```
<input type="text" class="form-control" placeholder="Section filter"
#filter>
```

just before <ul> block. Modify ngFor in <li>:

```
<li *ngFor="let section of sections | sectionFilter: filter.value"
```

4) Reload page and check the result. When you will enter something in Section filter and move focus outside input box, sections list will be filtered.

5) To filter sections on every keypress, add keypress event binding to input box:

```
<input ... #filter (keyup)="0">
```

This will fire keyup event and make Angular to update template.

6) However, if we will add new section that satisfies filter, we will not see it until we change filter value. That happens because pure pipe rerun only if its parameters change. In this case sections is checked as a reference, and reference doesn't change on adding new section. To make pipe to be executing on any update, make it impure: set **pure:false** in **@Pipe** decorator of pipe. It will execute pipe on any change.

Additional tasks

1) Implement filter for the notes. It should look for any substring in notes.
2) Implement tags for notes, with possibility to filter with tags and storing tags in database