WEB-011
AngularJS

Workbook
(version 1.0, 18.09.2015)

Workbook tasks

Task 1

Create the simplest AngularJS application

Task

We will create the very basic application which will contain the input field and show the greeting when input field is changing.

Time

1 hour

Detailed description

1) Create index.html and attach necessary scripts

```
<html ng-app="myapp">
    <head>
        <title>Hello AngularJS</title>
        <script src="angular.js"></script>
        <script src="hello.js"></script>
    </head>
</html>
```

2) Copy angular.js from labguide folder to the root folder

3) Insert this code into <body> section:

```
        <div ng-controller="HelloController">
            Your name: <input ng-model="name">
            <button ng-click="update()">update</button>
            <p>{{greeting}}</p>
        </div>
```

4) Create file hello.js and define controller function:

```
angular.module("myapp", [])
    .controller("HelloController", function($scope) {
        $scope.greeting = "";
        $scope.update = function() {
            if ($scope.name) {
                $scope.greeting = "Hello, "+$scope.name+"!";
            }
        }
    });
```

5) Open index.html in the browser.

When you will change the value in index field, the greeting will be immediately updated.

Task 2

Create the server side for the simplest application

Task description

We will be using NodeJS to create server side for the AngularJS application.

Time

1 hour

Detailed description

2.1 Create the server side

1) Download and install NodeJS

2) Create file server.js in the root folder of you application

3) Define necessary variables in server.js

```
var express = require('express');
var app = express();
var path = require('path');
app.use(express.static(path.join(__dirname, 'public')));
```

The last command will allow public folder to work as static folder, which means that if client side will retrieve some document from the server, it will be loaded from public folder.

4) Add app.listen(3000); to the end of server.js to execute the server

5) Add route which will allow to get the greeting from the server:

```
app.get("/greeting", function(req,res) {
    res.send("Hello, "+req.query.name+"! I'm server!");
});
```

6) Install express module by executing

```
npm install express
```

7) Execute the server by command

```
node server.js
```

8) Go to URL http://localhost:3000/greeting?name=John

You should see the response «Hello, John!»

2.2 Modify the client side to work with the server

1) Move index.html, angular.js and hello.js to **public** folder so that it could be retrieved as static resources from the server

2) Modify Hello controller: we will be using $http service to retrieve greeting from the server.
For this change $scope.update method to

```
if ($scope.name) {
     $http.get("/greeting",
          {params:
               {name: $scope.name}
          })
          .success(function(res) {
               $scope.greeting = res;
          });
}
```

It will pass name to the server, and will run success callback function passing it the result from the server.

3) Add $http parameter to the list of controller parameters to inject $http service:

```
angular.module("myapp", [])
.controller("HelloController", function($scope, $http) {…}
```

Now you can open page
http://localhost:3000
to load **index.html**. Enter the name in input field and press Update button to get greeting from the server.

Additional task

Perform live update of data from the server. For this, introduce timer and use $interval service to periodically retrieve updated value from the server without need to press Update button:

```
    $interval($scope.update, 200);
```

In update() function you should pass $scope.name to the server and set $scope.greeting to the updated value.

Also you need to add $interval service to controller dependency injection:

```
    angular.module("myapp", [])
    .controller("HelloController",
         function($scope, $http, $interval) {…}
```

Alternate solution

```
    $scope.$watch("name", function() {
         $scope.update();
    });
```

Task 3

Notes application

Task

We start to develop the notes application which will allow to keep and manage notes.

Time

1 hour

Detailed description

3.1 Create client side of notes application

1) Create index.html in public folder

2) Define the main <html> tag and define application name:

```
<html ng-app="myapp"></html>
```

3) In the <head> section add title, style and libraries:

```
<head>
     <title>Notes application</title>
     <script src="angular.js"></script>
     <script src="notes.js"></script>
</head>
```

4) Define the template which will show the list of notes:

```
<body ng-controller="NotesController">
     <div ng-repeat="note in notes">
          <p>{{note.text}}</p>
     </div>
</body>
```

5) Create notes.js file in public folder

6) Define the module in notes.js:

```
var module = angular.module('myapp', []);
```

7) Define the NotesController in notes.js:

```
module.controller('NotesController',
     function($scope) {
            $scope.notes = [];
        });
```

8) Define update function in NotesController which will load the list of notes from the server asynchronously:

```
        var update = function() {
            $http.get("/notes")
                .success(function(notes) {
                    $scope.notes = notes;
                });
        };
```

9) Inside the NotesController call update method to initially load the list of notes:

```
        update();
```

10) Add $http parameter to the list of controller parameters to inject $http service:

```
    module.controller('NotesController',
        function($scope, $http) { … }
```

3.2 Create server-side of notes application

1) Create package.json file in the root folder:

```
    {
          "name": "tutorial",
          "version": "0.0.1",
          "dependencies": {
            "express": "~4.12.4"
          }
    }
```

2) Execute **npm install** to install necessary node modules

3) Define necessary variables in server.js

```
    var express = require('express');
    var app = express();
    var path = require('path');
```

```
     app.use(express.static(path.join(__dirname, 'public')));
```

4) Add app.listen(3000); to the end of server.js to execute the server

5) Add app.get("/notes") route definition to allow client to retrieve notes from the server:

```
     app.get("/notes", function(req,res) {
          var notes = [
              {text: "First note"},
              {text: "Second note"},
              {text: "Third note"}
          ];
          res.send(notes);
     });
```

3.3 Add some styling for the notes

1) Create styles.css in the public folder:

```
     .note {
         background-color:#EEE;
         padding:5px;
         width:80%;
         border:1px solid black;
         margin:10px;
     }
```

2) Attach css to index.html:

```
     <link rel="stylesheet" href="styles.css">
```

3) Apply styles for the notes:

```
          <div ng-repeat="note in notes">
               <p class="note">{{note.text}}</p>
          </div>
```

Now you can start the server by typing

```
     node server.js
```

Then go to the page http://localhost:3000
to see the notes retrieved from the server.

Task 4

Store notes in session

Task description

In this task we will store the notes in the session.

Time

30 min

Detailed description

4.1 Keep notes in session on server side

1) Edit package.json by adding necessary node modules:

```
"body-parser": "~1.12.4",
"cookie-parser": "~1.3.5",
"express-session": "~1.7.6",
"connect-mongo" : "*"
```

2) Execute npm install to download modules

3) Define session variable in server.js:

```
var session = require('express-session');
```

4) Define bodyParser to get parameters from http post requests

```
var bodyParser = require('body-parser')
```

5) Add bodyParser to express to allow it to automatically parse http post parameters:

```
app.use(bodyParser.urlencoded({extended: true}));
app.use(bodyParser.json());
```

6) Define the session parameters for express:

```
app.use(session({
    secret: 'angular_tutorial',
    resave: true,
    saveUninitialized: true
}));
```

7) Implement get notes from session:

```
app.get("/notes", function(req,res) {
        res.send(req.session.notes||[]);
    });
```

8) Implement post notes to keep it in session:

```
    app.post("/notes", function(req, res) {
        if (!req.session.notes) {
            req.session.notes = [];
            req.session.last_note_id = 0;
        }
        var note = req.body;
        note.id = req.session.last_note_id;
        req.session.last_note_id++;
        req.session.notes.push(note);
        res.end();
    });
```

We also introduce the counter for the note. It will allow to use id to delete or update the note.

4.2 Implement add note on client side

1) Add the textarea and «Add note» button to the page:

```
        Add note:<br>
        <textarea rows="10" cols="50" ng-model="text"></textarea>
        <button ng-click="add()">Add note</button>
```

2) Implement add function in NotesController

```
    $scope.add = function() {
        var note = {text: $scope.text};
        $http.post("/notes", note)
        .success(function() {
                $scope.text = "";
                update();
            });
    };
```

Now you can add the note from the web page. Notes should survive after page reload.

Additional tasks

1) Implement remove note

For this:

- create Remove button on index.html
  ```
  <button ng-click="remove(note.id)">
  ```
- create $scope.remove function and pass id parameter to the server:
  ```
  $http.delete("/notes", {params: {id:id}})
  ```
- execute update in success callback after note is deleted
- in server.js implement method which will find for the corresponding id and delete it:

```
app.delete("/notes", function(req,res) {
    var id = req.query.id;
    var notes = req.session.notes||[];
    var updatedNotesList = [];
    for (var i=0;i<notes.length;i++) {
        if (notes[i].id != id) {
            updatedNotesList.push(notes[i]);
        }
    }
    req.session.notes = updatedNotesList;
    res.end();
});
```

2) Implement «Send to top» button to send note to the top of the notes list

Task 5

Store session in mongodb

Task description

In this task we will store session in mongodb to allow notes to survive after the server restart.

Time

30 minutes

Detailed description

1) Define MongoStore and pass session to it in server.js

```
var MongoStore = require('connect-mongo/es5')(session);
```

2) Use MongoStore to store session in database (replace previous session definition):

```
app.use(session({
     store: new MongoStore({
         url: 'mongodb://localhost:27017/angular_session'
       }),
     secret: 'angular_tutorial',
     resave: true,
     saveUninitialized: true
}));
```

3) Create db folder and start mongodb:

```
<MONGODB_PATH>/bin/mongod --dbpath c:/angular/notes/db
```

Now you can add the notes and then restart server.
Notes should survive after the server restart.

Task 6

Store notes in files

Task description

In this task we will be using files to store and load nodes.

Time

1 hour

Detailed description

1) Add the variable to work with file system:

```
var fs = require('fs');
```

2) Implement storing notes in the file

```
app.post("/notes", function(req, res) {
     var note = req.body;
     var noteText = JSON.stringify(note)+"\n";
     fs.appendFile("notes.json", noteText, function(err) {

          if (err) console.log("something is wrong");
          res.end();
     });

});
```

It will append notes.json file on each request

3) Implement loading all notes from the file

```
app.get("/notes", function(req,res) {
     fs.readFile("notes.json", function(err, result) {
          if (result) {
               result = ""+result; // convert Object to String
               //remove last \n in file
               result = result.substring(0, result.length - 1);
               result = "["+result+"]";
```

```
                result = result.split("\n").join(",");
                res.send(result);
            } else {
                res.end();
            }
        });
});
```

Additional tasks

1) Implement removing notes from the file
For this:

- introduce the counter and keep it in the session
- set id for each note and increment the counter when adding the note (see task 4)
- add app.delete("/notes") path
- load the list of all notes to the array
- find the note by its id and remove it from the list of the notes
- rewrite the file by using
    fs.writeFile(filename, data, callback)
- see additional documentation for fs.writeFile at https://nodejs.org/api/fs.html#fs_fs_writefile_filename_data_options_callback

2) Imlement support for multiple users
For this:

- introduce input field to enter the name of the user
- keep the user data is the file with the name of the user
- add Update button to load notes from the user's notes file

Task 7

Use mongodb to store notes

Task descrition

In this task we will start to use mongodb to work with data.

Time

2 hours

Detailed desctiption

7.1 Working with mongodb from NodeJS

1) Edit package.json
Add dependency on last mongodb package:

```
        "mongodb": "*"
```

Then execute npm install to install necessary modules.

2) Define variables in server.js

```
     var Db = require('mongodb').Db;
     var Server = require('mongodb').Server;
```

3) Open mongodb connection

```
var db = new Db('tutor',
     new Server("localhost", 27017, {safe: true},
          {auto_reconnect: true}, {}));

db.open(function(){
          console.log("mongo db is opened!");
});
```

4) Add collection notes
Add these lines inside db.open callback

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

5) Allow notes to be loaded from database

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

6) Allow notes to be saved in database

```
app.post("/notes", function(req,res) {
     db.notes.insert(req.body);
     res.end();
});
```

7) Go to db folder and execute mongodb:

```
<MONGODB_PATH>/bin/mongod --dbpath .
```

Now you can restart server and add some notes.

7.2 Using mongo console to find/update/insert notes

1) Start mongo client:

```
<MONGODB_PATH>/bin/mongo
```

2) Change database to tutor:

```
use tutor
```

3) Find all notes:

```
db.notes.find()
```

4) Insert note:

```
db.notes.insert({text:"my note"})
```

5) Find note by query:

```
db.notes.find({text:"my note"})
```

6) Change note text:

```
db.notes.update({text:"my note"}, { $set:{text:"his note"} })
```

7) Change or insert note:

```
db.notes.update({text:"my note"}, { $set:{text:"her note"}})
```

This command will not do anything because there's no note with text "my note".

```
     db.notes.update({text:"my note"}, { $set:{text:"her note"}},
          {upsert:true})
```

This will find for note "my note", and if it was not found, insert the new note with the text "her note".

8) Change multiple notes

Usually update changes only one record. If we need to change all records, we have to add parameter {multi:true}.

Let’s add property lastUpdated to the notes.
Execute this command:

```
db.notes.update({}, {$set:{lastUpdated:new Date().getTime()}})
```

You will see that only one item is updated. To update all items, you need to add {multi:true} parameter:

```
db.notes.update({}, {$set:{lastUpdated:new Date().getTime()}},
{multi:true})
```

It will add lastUpdated to all notes.

9) Ordering items

By default items in collection are ordered as they were added. You can reorder it by using orderBy. For example to order by text (alphabetically) use this command:

```
     db.notes.find().sort( { text: 1 } )
```

This command will order by text in decreasing order:

```
     db.notes.find().sort( { text: -1 } )
```

You also can use several fields for ordering:

```
     db.notes.find().sort( { lastUpdate:1, text: 1 } )
```

It will order by lastUpdate, and then by text.

9.1) Removing item from collection

```
db.notes.remove({text:"my text"})
```

Remove collection with indexes

```
db.notes.drop();
```

10) Remove field from the items

To remove field lastUpdated from notes, execute this command:

```
db.notes.update({}, {$unset: {lastUpdated:""}}, {multi:true})
```

7.3 Delete notes from database by id

1) Add ObjectID variable to server.js:

```
var ObjectID = require('mongodb').ObjectID;
```

2) Implement delete in server.js

```
app.delete("/notes", function(req,res) {
    var id = new ObjectID(req.query.id);
    db.notes.remove({_id: id}, function(err){
        if (err) {
            console.log(err);
            res.send("Failed");
        } else {
            res.send("Success");
        }
    })
});
```

3) Add remove button to index.html template

```
<div ng-repeat="note in notes">
    <p class="note">{{note.text}}
        <button ng-click="remove(note._id)">Remove</button>
    </p>
</div>
```

4) Define remove function in NotesController

```
    $scope.remove = function(id) {
        $http.delete("/notes",{params:{id:id}})
        .success(function(res) {
            update();
        });
    }
```

Now you can restart server and refresh page, and delete notes by clicking remove button.

Additional tasks

1) Save date and time of adding note in date field. Print it in hh:mm dd.mm.yyyy format (use filter to format the date):

```
    {{date | date: 'HH:mm dd.MM.yyyy'}}
```

2) Add possibility to reorder notes (add "send to top" button)
For this:

- add order property to the note
- find the note with minimal order number by using query

```
db.notes.find().sort( { order: -1 } ).limit(1)
```

- read the minimal order value and decrement it
- update the note by setting order to decremented value
- use ordering by order when loading the notes from mongodb

3) Implement sorting notes by natural order, by special order property and by creation date
For this:

- add <select> with possible orderings:

```
<select ng-model="order">
    <option value="order">Sort by order</option>
    <option value="text">Sort alphabetically</option>
    <option value="date">Sort by date</option>
</select>
```

- pass the order to the server and use it in sort when retrieving notes

4) Add possibility to edit notes

Task 8

Sections

Task description

In this task you have to create sections for notes, with possibility for reordering.

Time

1 hour

Detailed description

8.1 Add bootstrap css

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

1) Add bootstrap.min.css to public/bootstrap folder

2) Add link to bootstrap:

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

3) Copy design from labguide/index.bootstrap.html to your index.html <body>, replacing the old <body> contents

Refresh page to see how it is looking with the bootstrap.

8.2 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) In the index.html find <!-- Sections --> comment and add sections after that:

```
      <ul class="list-group">
            <li href="#" class="list-group-item"
                ng-repeat="section in sections">
                    {{section.title}}</li>
      </ul>
```

4) Run mongo and add some sections:

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

5) Now add function readSections to NotesController:

```
     var readSections = function() {
          $http.get("/sections")
          .success(function(sections) {
               $scope.sections = sections;
               update();
          });
     }

     readSections();
```

Now refresh the page and see the sections on the right.

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

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

```
   <li href="#" class="list-group-item"
          ng-repeat="section in sections"
          ng-click="showSection(section)"
          ng-class="{active: section.title==activeSection}">
```

```
                {{section.title}}
    </li>
```

2) Add showSection() to NotesController:

```
     $scope.showSection = function(section) {
          $scope.activeSection = section.title;
          update();
     }
```

3) Change update function to show notes only for the selected section (added params):

```
     var update = function() {
          var params = {params:{section:$scope.activeSection}};
          $http.get("/notes", params)
               .success(function(notes) {
                    $scope.notes = notes;
               });
     };
```

4) Set initially active section to the first section in list

Add to readSections function, inside success callback:

```
          if ($scope.activeSection == null &&
                    $scope.sections.length>0) {
               $scope.activeSection =
                    $scope.sections[0].title;
          }
```

This will select the first section when page is initially loaded.

5) 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.

8.4 Add section

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

1) Add to index.html, above the <ul> with sections, the input group to add the section:

```
<div class="input-group" style="padding-bottom:20px">
    <input type="text" class="form-control"
           placeholder="New section name" ng-model="newSection">
    <span class="input-group-btn">
        <button class="btn btn-default"
                ng-click="addSection()">Add</button>
    </span>
</div>
```

2) 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();
          });
     });
});
```

3) Add function $scope.writeSections to NotesController

It will write the current list of sections ($scope.sections) to the database.

```
    $scope.writeSections = function() {
          if ($scope.sections && $scope.sections.length>0) {
               $http.post("/sections/replace", $scope.sections);
```

```
        }
    };
```

4) Add function $scope.addSection:

```
$scope.addSection = function() {
     if ($scope.newSection.length==0) return;

     // check for duplicates
     for (var i=0;i<$scope.sections.length;i++) {
          if ($scope.sections[i].title==$scope.newSection) {
               return;
          }
     }

     var section = {title: $scope.newSection};
     $scope.sections.unshift(section);
     $scope.activeSection = $scope.newSection;
     $scope.newSection = "";
     $scope.writeSections();
     update();
}
```

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

8.5 Add note to the section

We have to add note to the active section.

For that add this line to $scope.add function:

```
note.section = $scope.activeSection;
```

Also add check that note is not empty to avoid adding empty notes:

```
if (!$scope.text || $scope.text.length==0) return;
```

8.6 Add sections reordering

We will be using library which implements the possibility to reorder list by drag&drop. Here you can find the full description of the library:
http://marceljuenemann.github.io/angular-drag-and-drop-lists/

To add this, do the following:
1) Add labguide/angular-drag-and-drop-lists.js to public/js folder

2) Add link to library to index.html:
```
    <script src="js/angular-drag-and-drop-lists.js"></script>
```
3) Update sections list to support reordering by dragging:

```
    <ul class="list-group" dnd-list="sections">
      <li href="#" class="list-group-item"
         dnd-draggable="section"
         dnd-moved="sections.splice($index, 1); writeSections();"
         ng-repeat="section in sections"
         ng-click="showSection(section)"
         ng-class="{active: section.title==activeSection}">
              {{section.title}}
      </li>
    </ul>
```

dnd-moved contains operations performed after the section was moved:

- remove section from the list of sections
- writeSections(), which replace the sections collections on the server

4) Add dependency to the module:
```
    var module = angular.module('myapp', ['dndLists']);
```

Now you can refresh the page and check how section drag&drop is working.

Additional tasks

1) Add sections filering:

Add input field before sections <ul>:

```
     <input ng-model="sectionsFilter">
```

In sections <ul> change ng-repeat:

```
     ng-repeat="section in sections
          | filter: sectionsFilter">
```

2) Implement tags for notes, with possibility to filter with tags and storing tags in database

Task 9

Routes

Task description

In this exercise we will support for the routing in application. Additionally we will define the folder structure for the big Angular application.

Time

1 hour 30 minutes

Detailed description

9.1 Adding routes

1) Create public/routes folder

2) Create file routes.js in routes folder

3) Move the definition of module from notes.js to routes.js, add ngRoute dependency:

```
var module = angular.module('myapp', ['dndLists', 'ngRoute']);
```

4) Add routes.js to index.html:

```
    <script src="routes/routes.js"></script>
```

5) Add angular routes library to index.html

```
    <script src="js/angular-route.min.js"></script>
```

6) Add routes configuration to routes.js:

```
module.config(
   function($routeProvider) {
       $routeProvider.
           when('/', {
               templateUrl: 'routes/notes/notes.html',
               controller: 'NotesController'
           }).
          otherwise({
               redirectTo: '/'
           });
   );
```

7) Create file routes/notes/notes.html

8) Cut all contents of <div class="container"> in index.html (the last and biggest one) and put it to notes.html

9) Change the original <div> by adding ng-view attribute:

```
        <div class="container" ng-view></div>
```

10) Move notes.js to routes/notes/notes.js
11) Add notes.js to index.html:

```
        <script src="routes/notes/notes.js"></script>
```

Now you can check if the application works correctly.

9.2 View section

This feature will allow to preview only one section and send it as URL to someone, or add to bookmarks.

1) Create folder routes/viewSection

2) Add viewSection.html with this contents:

```
<div class="panel panel-primary">
     <div class="panel-heading">
          <h3 class="panel-title">{{section}}</h3>
     </div>

     <ul class="list-group" ng-repeat="note in notes">
          <li class="list-group-item">{{note.text}}</li>
     </ul>
</div>
```

3) Create ViewSectionController in routes/viewSection/viewSection.js:

```
module.controller("ViewSectionController",
function($scope, $http, $routeParams) {

     $scope.section = $routeParams.name;

     var params = {params: {section:$routeParams.name}};

    $http.get("/notes", params)
          .success(function(notes) {
               $scope.notes = notes;
          });
});
```

4) In route.js, add viewSection part:

```
   when('/section/:name', {
       templateUrl: 'routes/viewSection/viewSection.html',
       controller: 'ViewSectionController'
   })
```

5) In notes.html add a link to the section in the header:

```
<div class="panel-heading">
     <h3 class="panel-title">
          Add your note to
               <a ng-href="#/section/{{activeSection}}">
               {{activeSection}}</a>
     </h3>
</div>
```

Now you can refresh page, open some section and go to link in the header. You should see the section contents.

9.3 Navigating sections

Now we want to provide section navigation using URL. It will allow to use Back/Forward buttons in browser, as well as add section to bookmarks.

The current section should be shown in URL. For this:

1) Update routes.js to read section from URL:
Add "/:section?" to read it in NotesController.

```
       $routeProvider.
           when('/:section?', {
               templateUrl: 'routes/notes/notes.html',
               controller: 'NotesController'
           }).
```

Example: http://localhost:3000#Work

2) Add services $routeParams, $location to NotesController

3) Set activeSection to the section in route params:
```
     $scope.activeSection = $routeParams.section;
```

4) In $scope.showSection function add line
```
          $location.path(section.title);
```
This will keep URL actual, with the name of the section, when user will change the section.

Additional tasks

1) Move/copy note to another section.

2) Create web site on the base of notes. It should have the sections as menu items, and notes as the page contents. Show it on the separate route.

3) Implement subsections.

Task 10

Form with validation

Task description

In this task you will create a registration form and validation for the form.

Time

3 hours

Detailed description

10.1 Create the form

1) Add the button Register to upper form after Sign In button:

```
<a class="btn btn-primary" ng-href="#/register">Register</a>
```

2) Add new route to routes.js:

```
   when('/register', {
       templateUrl: 'routes/userForm/userForm.html',
       controller: 'UserFormController'
   })
```

Note: you have to put this when('/register') **before** when('/:section?'), otherwise when('/:section?') will match first and when('/register') will not be processed.

3) Create routes/userForm/userForm.html

```
<div class="panel panel-primary">
     <div class="panel-heading">User registration form</div>
     <div class="panel-body">

     </div>
</div>
```

4) Add the form inside inner div

```
<form class="css-form" name="userForm" novalidate></form>
```

novalidate turns off HTML5 validation

5) Add User name field:
```
<div class="form-group">
   <label for="userName">User name</label>
   <input type="text" class="form-control" id="userName"
name="userName" placeholder="Username" >
</div>
```

6) Do the same field for password (type="password"), repeat password, date of birth, e-mail (type="email")

7) Add Submit button in the end
```
<button type="submit" class="btn btn-primary">Submit</button>
```

8) Add controller in routes/userForm/userForm.js:
```
module.controller("UserFormController", function($scope, $http)
{

     $scope.user = {};

});
```

Now you can click Register button and should see the form.

10.2 Add validation to the form

1) Add ng-model to bind form to Angular: for example, for userName add ng-model="user.userName"

2) For userName, password and password2 add required attribute

3) For date of birth, add pattern
```
     ng-pattern="/^[0-9][0-9]\.[0-1][0-9]\.[1-2][0-9][0-9][0-9]$/"
```

4) Allow form submit only when the form is valid: add ng-disabled to Submit button:
```
     ng-disabled="userForm.$invalid"
```

Show e-mail only if user wants to subscribe to newsletters

1) Add checkbox Subscribe for newsletters

```
<div class="checkbox">
     <label><input type="checkbox"
          ng-model="user.subscribe">
               Subscribe for newsletters
     </label>
</div>
```

2) Add attribute to e-mail address div:

```
ng-show="user.subscribe"
```

10.3 Add hint when field is incorrect

If the date of birth is in the wrong format, show the hint message. Add this <span> to the bottom of the date of birth <div>:

```
<span ng-show="userForm.dateOfBirth.$error.pattern">
     Date of birth should be in format dd.mm.yyyy
</span>
```

10.4 Add check that password and repeat password are the same

1) Create a directive in routes/userForm/userForm.js:

```
<input ng-model=”password”>

<input match-to=”password” ng-model=”password2”>

module.directive("matchTo", function() {
    return {
        require: "ngModel",
        scope: {
            otherValue: "=matchTo"
        },
        link: function(scope, element, attributes, ngModel) {

            ngModel.$validators.matchTo = function(modelValue) {
                return modelValue == scope.otherValue;
            };

            scope.$watch("otherValue", function() {
                ngModel.$validate();
            });
        }
    };
});
```

2) Add this <span> to Repeat password <div>:

```
<span ng-show="userForm.password2.$touched &&
               userForm.password2.$error.matchTo">
     Passwords should match
</span>
```

3) Add attribute to password2 input field:

```
     match-to="user.password"
```

10.5 Add check that userName is unique

1) Add directive uniqueUser:

```
module.directive('uniqueUser', function($http, $timeout) {
       var timer;
       return {
         restrict: 'A',
         require: 'ngModel',
         link: function(scope, elem, attr, ctrl) {
           scope.$watch(attr.ngModel, function(value) {
             if (timer) $timeout.cancel(timer);
             timer = $timeout(function(){
             $http.get('/checkUser?user='+value)
               .success(function(result) {
                   ctrl.$setValidity('unique', result);
               });
             }, 200);
           })
         }
       }
     });
```

2) Add span with error hint to userName <div>:

```
<span ng-show="userForm.userName.$dirty &&
          userForm.userName.$error.unique">
          User name is not unique
</span>
```

3) Add directive to userName <input ng-model=”user”>:

```
              unique-user
```

4) Add "/checkUser" to server.js:
Since we don't have users table and can't check userName for uniqness, let create some mock implementation for a while:

```
     app.get("/checkUser", function(req,res) {
          res.send(req.query.user.length>2);
     });
```

5) Alternate directive to check user for uniqness may be looking this way. It uses $asyncValidators and $q service to return asynchronous validation result.

```
module.directive('uniqueUser', function($http, $q) {
       var timer;
       return {
         restrict: 'A',
         require: 'ngModel',
         link: function(scope, elem, attr, ngModel) {
           ngModel.$asyncValidators.unique =
           function(modelValue, viewValue) {
                 var value = modelValue || viewValue;

                 return $http.get('/checkUser?user=' + value).
                    then(function(response) {
                         if (!response.data) {
                              return $q.reject();
                         }
                         return true;
                    });
                };
           }
       }
```

```
    });
```

10.6 Add user to mongodb

1) In server.js add new collection users:

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

2) Add possibility to post user to users collection:

```
        app.post("/users", function(req,res) {
             db.users.insert(req.body, function(resp) {
                  req.session.userName = req.body.userName;
                  res.end();
             });
        });
```

3) In userForm.html:

```
     add ng-submit="submitForm()" to <form> tag
```

4) In userForm.js:
Add submitForm to UserFormController

```
     $scope.submitForm = function() {
            $http.post("/users", $scope.user)
            .success(function(data) {
                 console.log("saved!");
                 $location.path("/");
            });
     }
```

Now you can register the user. To check that user was added to database, execute mongo and look at list of the users:

```
              db.users.find();
```

10.7 Implement check userName for uniqueness

Implement it yourself.

Additional tasks

1) Check that user's age is >12 (create special directive)

2) Implement drop-down to select country and city (list of countries and cities should be loaded from database), so that if user selects country, the list of cities would be loaded

Task 11

Multiuser support

Task description

In this task you will submit the form and create users on server side.

Time

3 hours

Detailed description

11.1 Show sections and notes for the current user

1) Execute mongo
2) Run

```
use tutor
db.sections.update({},{$set:{userName:"demo"}}, {multi:true})
db.notes.update({},{$set:{userName:"demo"}}, {multi:true})
db.users.insert({userName:"demo",password:"demo"})
```

3) Check updates:

```
show collections
db.sections.find()
db.users.find()
db.notes.find()
```

4) In server.js

1. Add function setUser to add userName to query objects:

```
function setUserQuery(req) {
     req.query.userName = req.session.userName || "demo";
}
```

2. Update all queries to include user:

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

```
        });
```

Also update post notes.

3. Define sections as the array in users table.
   New structure of user in users collection will be like this:

```
        { userName: "John",
          password:"jpass",
          sections: ["Work", "Vacation", "Hobby"]
        }
```

   For this:
   4.1 Update app.get("/sections"):

```
        app.get("/sections", function(req,res) {
             var userName = req.session.userName || "demo";
             db.users.find({userName:userName})
             .toArray(function(err, items) {
                  var user = items[0];
                  res.send(user.sections);
             });
        });
```

   4.2 Rewrite app.post("/sections/replace"):

```
        app.post("/sections/replace", function(req,res) {
             var userName = req.session.userName || "demo";
             db.users.update({userName:userName},
                       {$set:{sections:req.body}},
             function() {
                  res.end();
             });
        });
```

11.2 Implement login

Now lets implement the login functionality.

1) Create UserService in routes/login/userService.js:

```
module.factory("UserService", function($http, $rootScope) {
     var service = {};
     service.userName = "";
     service.loggedIn = false;

     service.login = function(login, password) {

          $http.post("/login", {login:login, password:password})
               .success(function(res) {
                    if (res) {
                         service.loggedIn = true;
                         service.userName = login;
                         console.log("logged in!");
                    } else {
                         console.log("wrong user/password!");
                         $rootScope.wrongPassword = true;
                    }
               });
     }

     return service;
});
```

2) Create login controller in routes/login/loginController.js:

```
module.controller("LoginController", function($scope, UserService)
{

     $scope.loggedIn = UserService.loggedIn;

     $scope.login = function() {
          UserService.login($scope.username, $scope.password);
     }

});
```

3) Update login form in index.html:
- add ng-controller="LoginController" to <div id=”navbar”>
- add ng-model="username" to username and ng-model="password" to password
- add ng-show="!loggedIn" ng-submit="login()" to <form>

4) Implement login in server.js

```
app.post("/login", function(req,res) {
     db.users.find(
               {userName:req.body.login,
                    password:req.body.password})
          .toArray(function(err, items) {
               if (items.length>0) {
                    req.session.userName = req.body.login;
               }
               res.send(items.length>0);
          });
});
```

11.3 Use $rootScope from controller

Now let's show the error message on wrong login.
First approach: using $rootScope

1) Add error message to the form:
Under the login form, add the following message:

```
<div ng-show="wrongPassword" style="color:white">
        Wrong username or password
</div>
```

2) Add services $rootScope, $timeout to UserService parameters for dependency injection

3) On wrong username/password, add the following code:

```
console.log("wrong user/password!");
$rootScope.wrongPassword = true;
$timeout(function() {
     $rootScope.wrongPassword = false;
}, 1000);
```

11.4 Using $q service to return promise from the service

But service may access only rootScope, and it's more correct to separate service and UI, so let's add the reaction to LoginController.

For this, we would need to return deferred object from the service.

1) Define deferred object in UserService.login():
```
var deferred = $q.defer();
```

2) On successful login, do
```
deferred.resolve("logged in");
```

3) On wrong login, do
```
deferred.reject("wrong username/password");
```

4) Return promise from UserService.login() function:
```
return deferred.promise;
```

5) Process deferred result in LoginController:
```
$scope.login = function() {
     UserService.login($scope.username, $scope.password)
     .then(
          function() {
               $scope.loggedIn = true;
               $location.path("/");
               $route.reload();
          },
          function() {
               $scope.wrongPassword = true;
               $timeout(function() {
                    $scope.wrongPassword = false;
               }, 1000);
          }
     );
}
```

11.5 Implement logout

1) Add logout code to the index.html inside <div id="navbar">:

```
<div ng-show="loggedIn" style="color:white"
          class="navbar-form navbar-right">
     User: {{username}}
     <a class="btn btn-primary" ng-click="logout()">Logout</a>
</div>
```

2) Add processing of logout to LoginController:

```
$scope.logout = function() {
     UserService.logout().then(function() {
          $scope.loggedIn = false;
          $location.path("/");
          $route.reload();
     });
}
```

3) Implement logout in UserService:

```
service.logout = function() {
     return $http.get("/logout");
}
```

4) Implement logout in server.js:

```
app.get("/logout", function(req, res) {
     req.session.userName = null;
     res.end();
});
```

Additional tasks

1) After page reload user needs to login again. How to fix it?

2) Implement edit of the user data

Task 12

Use $resource service to access REST service

Task description

In this task we will be using $resource service to access the notes.

Time

1 hour

Detailed description

1) Add ngResource to scripts in index.html:

```
<script src="https://code.angularjs.org/1.4.3/angular-
resource.js"></script>
```

2) Add ngResource dependency to the module in routes.js:

```
var module = angular.module('myapp', ['dndLists', 'ngRoute',
'ngResource']);
```

3) Define Note factory in notes.js:

```
module.factory('Note', function($resource) {
       return $resource('/notes');
     })
```

4) Add Note to the list of parameters in NotesController

5) Change update function to this implementation:

```
     var update = function() {
          $scope.notes = Note.query(
                    {section:$scope.activeSection});
     };
```

6) Change $scope.add function to this implementation:

```
     $scope.add = function() {
          if ($scope.text.length==0) return;
          var note = new Note();
          note.text = $scope.text;
```

```
            note.section = $scope.activeSection;
            note.$save(function() {
                $scope.text = "";
                update();
            });
        };
```

Additional tasks

1) Create NoteService. It should use $resource service to retrieve user data.