IPTK Project Documentation, Team November, SS2016

Ahmed Zakaria   Matriculation Nr: 2789798
Fadi Boutros   Matriculation Nr: 2886837
Yassin Alkhalili   Matriculation Nr: 2332310
Prasad Sawant   Matriculation Nr: 2017648
Shashikumar Selvaraj   Matriculation Nr: 2438926

TECHNISCHE
UNIVERSITÄT
DARMSTADT

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Contents

1 Project Structure

```
Backend
|- - config
    |- -config.js
    |- -configurations.js
    |- -express.js
    |- -jwt_secret.js
    |- -mongoose.js
    |- -passport.js
    |- -socketio.js
|- - controllers
    |- -conferenceController.js
    |- -downloadFileController.js
    |- -emailController.js
    |- -indexController.js
    |- -reportingController.js
    |- -reviewController.js
    |- -submissionController.js
    |- -taskController.js
    |- -userController.js
|- - middlewares
    |- -validateRequest.js
|- -models
    |- -conferenceModel.js
    |- -emailModel.js
    |- -reviewModel.js
    |- -submissionModel.js
    |- -userModel.js
|- -node_modules
|- -routes
    |- -auth.js
    |- -indexRoutes.js
    |- -reviewRoutes.js
    |- -submissionRoutes.js
    |- -userRoutes.js
    |- -usersRoutes.js
|- -gulp.js
|- -package.json
|- -server.js
```

2 Backend

Our Conference Management system is based on a RESTful stateless API where each resource at the backend has a URI known by the client-side. The client only issues HTTP requests (e.g. GET,POST,DELETE) to our back-end and exchange data in JSON format only. Our CMS is built on mongoDB with mongoose that enables defining structural DB using its modelling properties which gave us the chance to define different schemas for our models in a structured fashion. As for the authentication and the authorization of different requests and users, we implemented basic authentication strategy in which every user has to signup first with a valid E-mail (validated by a confirmation link sent to it) and password after the user confirms his/her E-mail, he can login using his credentials and our token service at the backend generates him a token for authenticating his further requests. Each new login leads to generating a new token which is considered to be more secure approach in case the token was intercepted. We have implemented authorization by defining user roles as follows

- Chair: Has all the authority to view, and delete all submissions, and reviews as well as creating conferences, assigning tasks,etc

- Author: has the authority to submit papers, view assigned tasks but of course is not allowed to review his papers.

- Reviewer: Review papers submitted by other authors and view assigned tasks by the conference chair.

In fact, our main roles can be seen as two roles; the chair and the user who can be either author or reviewer or even both at the same time.
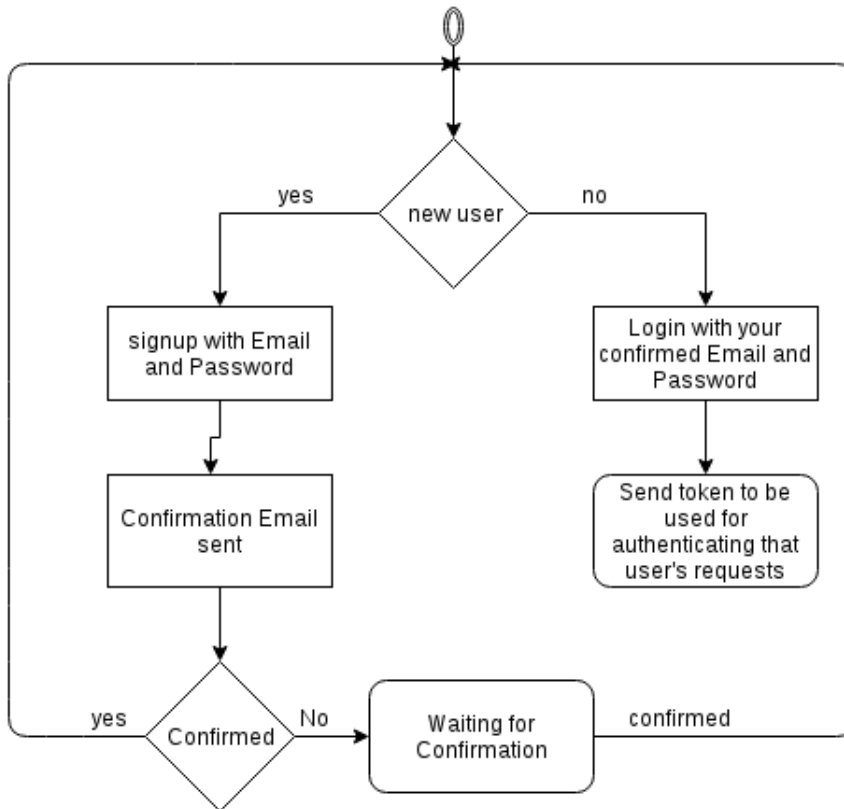

Figure 1: Simple illustration for our strategy for authenticating users through Email password pairs and unique generated tokens"

2.1 Our Models

Our CMS consists of the following Mongoose schemas:

- conferenceModel comprises the following properties:

    title : String representing the conference name

    chair: User object refers to the chair of the conference and it's a mandatory field that's why the required field is set to true

    startdate: start date of the conference

    enddate: end date of the conference

    conferenceLocation: String representing the location of the conference

    authors: array of strings representing the names of the participant authors in the conference

- emailModel comprises the following properties:

    service: String representing the name of the email service provider e.g. Gmail

    auth: contains credentials of the Email server

    from: String representing the sender email

    to: String representing the receiver email

    subject: String representing the subject of the email

    html: html code for the body of the email to be sent

- reviewModel comprises the following properties:

  expertise: is a number from 1 to 5 representing the level of familiarity of the reviewer with the papers topic to be reviewed, where 1 means beginner and 5 means expert.

  overallEvaluation: Number between 1 and 5 representing the evaluation score given by the reviewer to a paper.

  summary: String containing a review summary

  strongPoints: String containing the strong points of the paper.

  weakPoints: String containing the weakness points of the paper.

  detailedComments: String containing detailed comments on the paper being reviewed.

  deadline: Deadline date for the reviewing phase.

  fileName: String representing the name of the paper being reviewed.

  reviewers: Array of strings representing the names of the reviewers of a paper.

- submissionModel comprises the following properties

  title: String representing the submission name

  abstract: String representing Abstract of the paper

  keywords: Array of strings representing keywords about the submission.

  status: String that represents the status of the submitted paper and can be one possiblity of these values incompleted,completed,closed,accepted,rejected

  fileName: String representing the name of the submitted file

  generatedFileName: random string generated for each submission and used in naming the submissions before saving it on our server

  deadline: Date representing the deadline for submissions

  createdOn: Date representing the creation date of the submission

  authorList: array of paperAuthor, where we defined paperAuthor as a mongoose subdocument with the following properties:

  – familyName: family name of the author of the paper

  – givenName: first name of the author of the paper

  – email: Email of the author of the paper.

- userModel comprises the following properties:

  username: unique string that matches only valid emails

  password: String of length longer than 6 characters

  role: String enum that can be : user, reviewer,author,chair

  givenName: String

  familyName: String

  institute: String for the name of the institute of the participating user

  city: String representing the city of the user

  state: String representing the state of the user

  country: String representing the country of the user

  address: String representing the address of the user

  salt:

  zipcode: String representing the zipcode of the user

  created: date of creation of the user object on the system

  is_confirmed: Boolean reflecting the state of the signed up user whether he has confirmed his email or not

  reviews: Array of Review objects, for the review object details refer to the reviewModel, as each user has an array of reviews to review in case his role is a reviewer.

submissons: Array of submission objects, for the submission object details refer to the submissionModel, as each user has an array of submissions in case he is an author

conferences: Array of conference objects, as each user can be participant in more than one conference. Please refer to the conferenceModel for more details on the conference object

tasks: Array of tasks, for details on the taskSchema refere to the taskModel.

- The taskModel comprises the following properties:

    taskType: An enumerated String that can be either 'submitting' task or 'reviewing' task.

    taskDesc: String representing more detailed description of the task.

    createdOn: Date of creation of the task

    createdBy: reference to User mongoose object that reflects the assigner of the task

    conferenceId: reference to conference mongoose object that reflects the conference in which these tasks are created.

    submissionId: reference to submission mongoose object that reflects the submission id of the submission that has to be reviewed and it is set to NULL in case of submitting task.

    assignedTo: A mandatory string that refers to the assignee email address

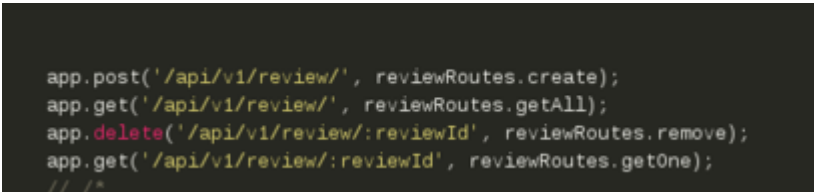   isUserNotified: A boolean that is by default false and set when the chair assigns the user a task.

2.2 Controllers and Routes

The backbone of our system's backend is the controllers, where we implemented all the HTTP verbs needed to satisfy and consume any HTTP requests directed to our RESTful API. In this section of the documentation, we will go through and explain the core idea behind our controllers. We have implemented 7 controllers as follows:

- conferenceController

- downloadFileController

- emailController

- reviewController

- submissionController

- taskController

- userController

Beside the above mentioned controllers and for the sake of a better design and code readability we have implemeted route files, they can only be seen as intermediate layer between the controller and the main indexRoutes.js file as it's a best practice approach not to expose the controller code inside the index file so we used the following route files:

- indexRoutes: this is the main routes file that defines all the offered API routes by our CMS linked to the controllers by the other different route files we implemented.

```
app.post('/api/v1/review/', reviewRoutes.create);
app.get('/api/v1/review/', reviewRoutes.getAll);
app.delete('/api/v1/review/:reviewId', reviewRoutes.remove);
app.get('/api/v1/review/:reviewId', reviewRoutes.getOne);
// /*
```

Figure 2: code snippet for indexRoutes.js illustrating how it defines the api routes of the reviewController by using the reviewRoutes as intermediate layer "

- reviewRoutes

- submissionRoutes

- usersRoutes

- authRoutes

3 Frontend

The CMS is implemented using various development technologies, like for the frontend we used Angular2 despite that it's still subject to changes but actually it adopts the concept of Component-based development and object orientation which makes easier for developers and it's totally written in TypeScript which meets the ECMAScript 6 specification, in addition we used NPM, HTML, CSS3 and Bootstrap for responsive design and for the backend Node.js, Principles of RESTful service (Resource identification through URI, Resource manipulation using GET and POST operations, Resource decoupling from their representation so they can be accessed in JSON format).

In our Application the logged in user can have Multiple Roles (Chair, Author and Reviewer) and also the user can create multiple conferences as a Chair.

3.1 Technologies Used

Name	Version
Angular2	2.0.0-beta.15
TypeScript	1.8.10
es6-shim	0.35.0
bootstrap	3.3.7
bootswatch	3.3.7
jquery	3.1.0
reflect-metadata	0.1.2
rxjs	5.0.0-beta.2
systemjs	0.19.26
zone.js	0.6.10
typings	0.7.12
lite-server	2.2.0
concurrently	2.0.0
HTML	5
CSS	3.3.7

Table 1: Technology names and their versions

3.2 Implementation

The application is structured into components for each of the business logic (as mentioned earlier), so the collective of the entire components makes the final working application. The root of our application is the AppComponent and the application is launched by bootstrapping the AppModule in the main.ts file. When Angular launches the app, it places the HTML rendering of AppComponent in the DOM, inside the <pm-app> element tags of the index.html.

The Application is better understood as we go further explaining how the individual blocks of Angular2 is utilized in our implementation.(The explanation is referenced from https://angular.io)

3.3 Modules

Angular apps are modular and Angular has its own modularity system called Angular modules or NgModules

3.3.1 Modules used

- Bootstrap (bootstrap property identifies AppComponent as the bootstrap component.)

- NgModel

- NgFormModel, NgFormControl

- Validators (Used for Validations using control messages)

- Component.

- OnInit (Used for Initilizations)

- HTTP_PROVIDERS, JSONP_PROVIDERS, ROUTER_PROVIDERS,

6

- RouteConfig,ROUTER_DIRECTIVES, Router (Used for Routing Directives)

- ControlGroup (Used for Control Directives)

- FormBuilder,Injectable

- Http, Headers, Response, URLSearchParams, Jsonp, ConnectionBackend (Used for HTTP Directives)

- Observable, Directive, ElementRef, EventEmitter, Output

3.4 Components

A component controls a patch of screen called a view. In this project we use various components as mentioned in the below table, so that each of the business logic can be represented as a separate view.

3.4.1 Used Components

The app component is the root of our application which imports all the necessary modules and components (from the Table 2) to get bootstrapped.

3.5 Template

A template is a form of HTML that tells Angular how to render the component.

3.5.1 Used Template

The application has segregated each business logic as specified earlier into components and these components are rendered using HTML. The HTML files follows the same naming convention as the Components specified in the above table with the .html file extension likeapp.component.html etc.

3.6 Metadata

Metadata tells Angular how to process a class. In TypeScript, one can attach metadata by using a decorator.

3.6.1 Used Decorator and Metadata

```
@Component({
  selector: 'pm-app',
  templateUrl: 'app/app.component.html',
  directives: [ROUTER_DIRECTIVES, ControlMessagesComponent, ResultMessagesComponent, ConfirmComponent],
  providers: [AppService, JSONP_PROVIDERS, ValidationService, HTTP_PROVIDERS, ROUTER_PROVIDERS, ConfirmService]
})
```

Figure 3: Component Decorator

The @Component decorator, identifies the class immediately below it i.e., AppComponent class as a component class.

```
export class AppComponent implements OnInit {
  pageTitle: string = "Conference Management";
  isLog: boolean;
```

Figure 4: AppComponent Class

The @Component decorator takes a required configuration object with the information Angular needs to create and present the component and its view.

Here are a few of the possible @Component configuration options:

No.	Folder Name	Component Name (*.component)	Important Functions Used
1	root	app	ngOnInit() - for Initialization of services, pages etc., removeProfile() – for removing a user profile, logout() - for user logout
2	home	welcome	Opens up the home page
3	login	login	login()- to log into the Conference Management System
4	conference	conference, create-conference, public-conference	checkConferenceDate()- validation for start and end date, submit()-conference details are submitted and created
5	authors	invit-author	invit-for inviting authors for a Conference, removeAuthor() - for removing an author from a conference
6	papers	paper-create, paper-detail, paper-list,paper-edit,reviewer-papers,paper-filter.pipe, author-papers-conference, conference-papers-list	addAuthor() - for adding authors to a submission, removeAuthor() - removes added author, search() - searches for keyword from wiki, addKeyword() - for adding keywords, removeKeyword() -removes added keyword, create() - creates a new Paper submission, getPaper() -to get the Paper details, getFile() - to get the submitted file
7	reviewers	assign-review	getReviewer() - get the reviewers by id, assign() -assign reviewers for a Paper, removeReviewer() - to remove a reviewer from a Paper
8	review	review-create, review-detail	addReviews() – to add a review for the Paper
9	shared	confirm, control-message, redirect, result-message,star-component	_setLabels() - to set the title and message of the confirmation form, _show() – to show the confirmation window, activate() – to activate the title and message, _hideDialog() - hides the confirmation window, errorMessage() –to get the error message while submitting a form, messageClass() – to display either Success or Alert message, result() - returns the message
10	signup	signup	signup() –for signing up a new user, getAddress() - to populate the user address based on Google Place
11	chart	chart	setTimeLineChart()-creates data table for conference timeline chart, setStatusChart()-creates data table for status pie chart, getStatusCount()-gets count of the status, setCountryChart()-creates data table for user column chart, setTopicChart()-creates data table for topic bar chart
12	datepicker	datepicker.ts, datepickercontainer.ts, daypicker.ts, monthpicker.ts, yearpicker.ts	various functions are used which creates calendar objects

Table 2: CMS Application Components

- selector: CSS selector that tells Angular to create and insert an instance of this component where it finds a <pm-app> tag in parent HTML. For example, if an app's HTML contains <pm-app></pm-app> as shown below, then Angular inserts an instance of the AppComponent view between those tags.

- templateUrl: address of this component's HTML template, 'app/app.component.html'.

- directives array of the components or directives that the template requires. The application uses the following directives

  ROUTER_DIRECTIVES -Used for routing

  GoogleplaceDirective-Used for getting address using Google Place API

  ControlMessagesComponent - displays error message in form validation

  ResultMessagesComponent - Used to display messages like success,error messages based on user actions

  GoogleChart - it is a wrapper for the Google Visualizations

  DatePicker - provides form to select date in calendar

- providers: array of dependency injection providers for services that the component requires. This is one way to tell Angular that the component's constructor requires a AppService so it can perform the necessary functions.

  AppService

  JSONP_PROVIDERS

  ValidationService

  HTTP_PROVIDERS

  ROUTER_PROVIDERS

  ConfirmService

3.7 Data Binding

Angular supports data binding, a mechanism for coordinating parts of a template with parts of a component. Add binding markup to the template HTML to tell Angular how to connect both sides.
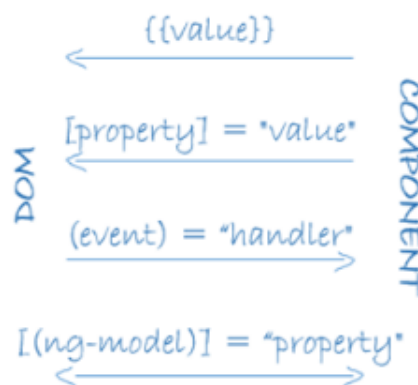

Figure 5: Data Binding

As shown in figure 5, there are four forms of data binding syntax. Each form has a direction — to the DOM, from the DOM, or in both directions.

3.7.1 Used Data Bindings

In the application, the app.component.html has {currentUser} [control]="form.controls.email", (click)=logOut() and

signup.component.html uses [(ngModel)] ="address".

- {{currentUser}} - specifies interpolation which displays the currentUser property within <a> hyperlink tag.

- [control]="form.controls.email" - specifies property binding which passes the value from the parent AppComponent to the control property of the child ControlMessagesComponent

- (click)="logOut()" - specifies event binding, which calls the components logout method when the user clicks on Log Out.

- [(ngModel)]="address" - specifies two-way data binding of the property address

3.8 Interfaces

The application implements an interface file app.interface which defines the interfaces based on the business logic that could be used for the components and services.

3.8.1 Used Interfaces

IPaper, IPaperAuthor, IReview, IUser, ICountry, IPlace

3.9 Service Directive

Service specifies the value, functions which are consumed by the components in the application.

3.9.1 Used Service Directives

The application uses AppService as the main service provider for the components. Various services like login (), getPapers () etc., has been defined in app.service.ts which are used by other components in the application.

Service Functions	Description
getPapers()	gets submissions JSON object
getPaper(id)	gets a submission with a specific ID
getUserConference()	JSON object regarding list of all the conferences is retrieved
getAllConference()	JSON object assigned to a particular author is retrieved
getConferenceSubmission(conferenceId)	Submission details JSON object for the conference is retrieved
getFiles(generatedFileName, filename)	Get the submitted .pdf files
assignReviewers(username, submissionId, conferenceId)	Adds reviewers for the particular conference id
removeReviewers(username, submissionId, conferenceId)	Removes reviewers for the particular conference id
inviteAuthor(username, conferencId)	Used for inviting authors to conference
removeAuthor(username, conferencId)	Removes authors assigned to a conference
getConferenceDetails(conferencId)	JSON object containing details about a conference is retrieved
paperSubmission(_paper, attFile)	Used to push submission details like paper id and attached file to the server
paperSubmissionEdit(_paper, attFile)	Pushes edited submission details to the server
getReviews()	JSON object regarding the reviews submitted for a submission is retrieved
addReview()	Pushes details regarding review for a submission
createConference(_conf)	New conference JSON object is submitted
changePassword(password, oldPassword)	Used for changing the current password
getUserProfile()	Details regarding user profile is retrieved
updateProfile(_user)	Pushes updated user profile details
signup(_user)	New user details are submitted for registration
isRegister(username)	Checks if new user is registered with the system
login(email, password)	Posts request for login to server with email and password as parameters
searchWiki(term)	Uses wikipedia API to retrieve the keywords while creating new submission
checkCredentials()	Checks if the user is logged in
isLog()	Checks if the user is currently logged into the system
getCurrentUserEmail()	Retrieves the current logged in Users Email address
getToken()	Used to keep the session active
logout()	The current user session is ended

Table 3: List of services implemented in AppService

3.10 Dependency Injection

It is a way to supply a new instance of a class with the fully-formed dependencies it requires. Most dependencies are services. Angular uses dependency injection to provide new components with the services they need.

3.10.1 Used Dependency Injection

For example: the below services are maintained as instances by Angular2 injector and also alternatively it can create new service instance from a provider.

- constructor(private _logInService: AppService, fb: FormBuilder, private _router: Router, private _confirmService: ConfirmService) { }

4 Future work

We have chosen to implement the backend as RESTful API for easy reuse in the future, as this totally decouples the technologies used at the backend from the ones used at the front end which enables us to implement for example any mobile client e.g android client and use the same backend as we are using now.

5 List of Features and the Status:

Sl No.	Tasks	Status(Done/Undone)
1.0.0	Responsive design	Done
2.0.0	User Functions	
2.1.0	Register to the system at least with email, password(emails are unique)	Done
2.1.1	Users can have multiple roles (e.g., author and reviewer)	Done
2.1.2	Basic profile (given name, family name, postal address)	Done
2.1.3	Affiliation (institution, city, state, country)	Done
2.2.0	login to the system at least with registered email, password	Done
2.3.0	edit my profile (password, additional information)	Done
2.4.0	remove my profile	Done
2.5.0	logout	Done
3.0.0	Author Functions	
3.1.0	create a new submission (if submission is open)	Done
3.1.1	title	Done
3.1.2	registered authors [1..*] (given name, family name, email affiliation)	Done
3.1.3	abstract	Done
3.1.4	keywords (multi-selection) that describe the submission (e.g., research fields)	Done
3.1.5	upload the paper (.pdf)	Done
3.2.0	access current submissions (overview)	Done
3.2.1	status: incompleted, completed, closed, accepted, rejected	Done
3.3.0	look at a submission	Done
3.4.0	edit a submission (if submission is open)	Done
3.5.0	withdraw a submission	Done
4.0.0	Reviewer Functions	
4.1.0	access assigned submissions (overview + status)	Done
4.2.0	look at an assigned submission	Done
4.2.1	details, pdf download or view	Done
4.3.0	make a review >template (if review is open)	Done
4.3.1	reviewer expertise: (1) not familiar w/ the topic - (5) expert	Done
4.3.2	overall evaluation: (1) strong reject - (5) strong accept	Done
4.3.3	summary	Done
4.3.4	major strong points	Done
4.3.5	major weak points	Done
4.3.6	detailed comments	Done
4.4.0	edit a review (if reviewing phase is open)	Done

Table 4: Features List with status

Sl No.	Tasks	Status(Done/Undone)
5.0.0	Chair Functions	
5.1.0	list all paper submissions	Done
5.1.1	look at a submissions (details, pdf download or view)	Done
5.1.2	withdraw a submission	Done
5.2.0	list all authors	Done
5.2.1	look at detail information	Done
5.3.0	list all reviewers	Done
5.3.1	look at detail information	Done
5.4.0	list all reviews	Done
5.4.1	look at a review	Done
5.5.0	assign papers to reviewers	Done
5.5.1	conflict avoidance: an author is not allow to review his own paper	Done
5.6.0	do the schedule management	Done
5.6.1	automatic: set close deadlines for submissions and reviews	Done
5.6.2	manual: (re-)open/close submissions and reviews	Done
5.6.3	create new conference	Done
5.6.4	Invite author to conference	Done
5.7.0	view fancy summary charts or reports (e.g., total submission, acceptances, topics, countries etc.)	Done
6.0.0	Bonus features (OPTIONAL)	
6.0.1	users must confirm their registration by clicking on a registration link sent via email	Done
6.0.2	email notification to submitters and reviewers (status changes)	Done
6.0.3	[NEW] a user can create multiple conference events and becomes the chair (=administrator) of them	Done

Table 5: Features List with status continued

6 User manual for navigating through the Application

The User Manual has been documented as video and attached in the Documentation folder.