**TEAM TANGO**

Detailed Technical

Design

Conference Management System  
*Version 1.0*

Table of Contents

Table of Contents ii

Revision History iii

Approved By iii

1. Introduction 1

1.1 Purpose 1

1.2 System Overview 1

1.3 Design Map 1

1.4 Definitions and Acronyms 1

2. Design Considerations 1

2.1 Assumptions 1

2.2 Constraints 2

2.3 System Environment 2

3. Architecture 2

3.1 Overview 2

3.2 Subsystems and Components 2

4. Database Schema 3

4.1 Document Models 4

4.1.1 Users 4

4.1.2 Conferences 4

4.1.3 Papers 4

4.1.4 Reviews 5

4.2 Document relationships 6

5. Low Level Design 6

Appendix A: Supplemetary documents 13

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Name** | **Reason For Changes** | **Date** |
| *0.1* | Team Tango | Initial draft version |  |
| 0.5 | Team Tango | Added DB schema diagram, architecture flow |  |
| 0.8 | Team Tango | Documented low level design |  |
| 1.0 | Team Tango | Added supplementary documents and Baselined |  |

Approved By

Approvals should be obtained for project manager, and all developers working on the project.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Signature** | **Department** | **Date** |
|  |  |  |  |
|  |  |  |  |

# Introduction

## Purpose

This document attempts to describe the specification, architecture and technical design of the “Tango Conference Management System”.

## System Overview

This system is a web application, built on MEAN technology stack that provides conference management functionalities. The functionalities can be broadly grouped under chairman, reviewer and author modules.

## Design Map

This document elaborates and describes all the assumptions, constraints and design decisions made and implemented under each ‘User Story’. As a supplement, there is also a document that lists all the REST API calls built as part of the system.

## Definitions and Acronyms

**MEAN stack** –MongoDB, Express.js, AngularJS and Node.js

DB – Database

JSON – Javascript Object Notation

Responsive web design – An application with a responsive web design reacts properly and scales to different resolutions(PC,Mobile,Tablet)

API – Application Programming Interface (A set of procedures that can be reused)

# Design Considerations

This section details all the design considerations and assumptions made.

## Assumptions

* The Conference management system assumes there is only one Conference and at least one chairperson is pre-decided and setup in the backend.
* The requirements listed under “Bonus” are optional to implement.

## Constraints

No constraints limiting the development of the project.

## System Environment

The Tango Conference Management system is a web application built on MEAN technology stack. MongoDB is the backend technology that stores data in ‘documents’ format. Node.js is the server-side runtime environment which is also JS based. This serves as the middleware technology. We make use of Express,js which is a popular web development framework for node.js. The front end logic is handled with AngularJS.

# Architecture

This section provides the top level design view of a system and provides a basis for more detailed design work

## Overview

The diagram that follows can be taken as a representation of how the interaction takes place in the system originating from a client. The reverse communication follows the same path and thus a client-server round trip takes place in the system.

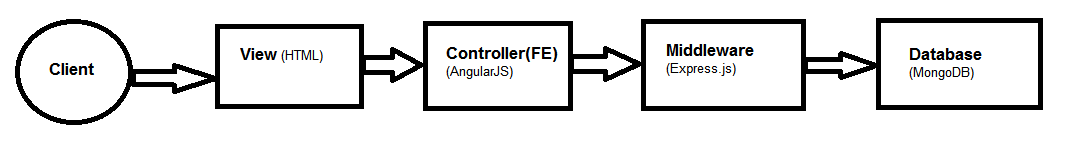


Figure 1 Conference Management System Overview

## Subsystems and Components

The system can be broadly divided into various modules. The common domain independent web application functions such as login, logout and register are handled in the parent module. The specific application features are grouped into user module, papers module and reviews module. A diagrammatic representation can give a clear idea.

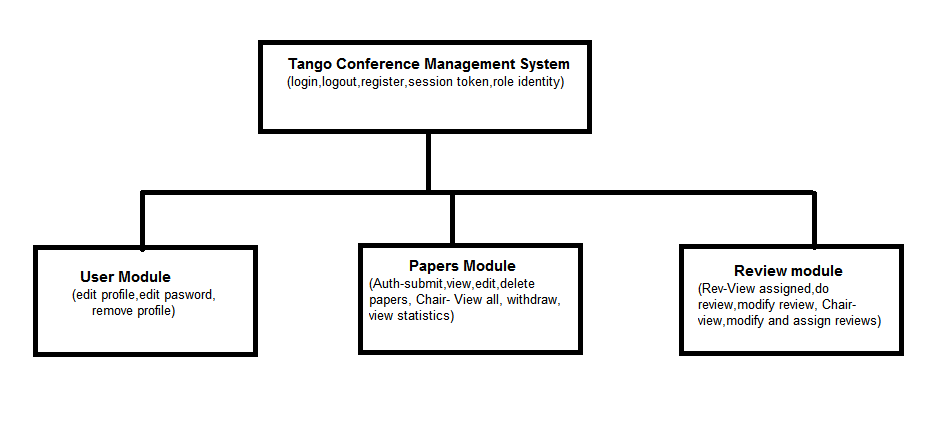


Figure 2 Subsystems of Tango App

Each of these modules span across DB models, API routes, Angular app modules and corresponding views and front end states.

**Technology/Frameworks information:**

The following table lists and describes all the module frameworks used in the development of the application.

|  |  |
| --- | --- |
| **Framework/Module** | **Purpose** |
| passport.js | Authentication module for node.js that provides various strategies. We use Local Strategy. |
| Mongoose.js | It is a middleware framework that handles interaction with MongoDB and takes care of boilerplate code. |
| ui-router | For state transitions in Angular side. |
| angular-charts.js | Takes care of rendering graphs and charts. |
| loadash | Provides a library of utility methods. |
| gridfs-stream | To stream in and stream out data from Gridfs backend. Gridfs allows storing data greater than BSON limit(16MB) in chunks. |

# Database Schema

The backend technology used in this application (Mongo) is a schema less DB in principle. However, for representation purposes the following section may be perused.

## Document Models

MongoDB stores and handles backend data as documents. So the schema is declared and defined as models. We use four schematic models in this application.

### Users

|  |  |
| --- | --- |
| **Field Name** | **Field Type** |
| username | String, lowercase, unique |
| firstname | String |
| lastname | String |
| mob | Number |
| email | String |
| organization | String |
| hash | String |
| salt | String |

### Conferences

|  |  |
| --- | --- |
| **Field Name** | **Field Type** |
| confTitle | String |
| confDesc | String |
| chairPerson | ObjectID (Ref : User) |
| submissionDeadline | Date |
| reviewDeadline | Date |

### Papers

|  |  |
| --- | --- |
| **Field Name** | **Field Type** |
| \_creator | ObjectID(Ref: User) |
| title | String |
| authors | ObjectID(Ref: User) |
| reviewer | ObjectID(Ref: User) |
| abstract | String |
| keywords | String |
| filename | String |
| status | Enum String (Incomplete,Completed,Closed,Accepted,Rejected) |
| createdAt | Date, required, default:current date |
| updatedAt | Date, required, default:current date |

### Reviews

|  |  |
| --- | --- |
| **Field Name** | **Field Type** |
| forSubmission | ObjectID(Ref: Paper) |
| reviewer | ObjectID(Ref: User) |
| summary | String |
| overallEvaluation | String |
| reviewerExpertise | String |
| strongPoints | String |
| weakPoints | String |
| detailedComments | String |
| createdAt | Date, required, default:current date |
| updatedAt | Date, required, default:current date |

## Document Relationships

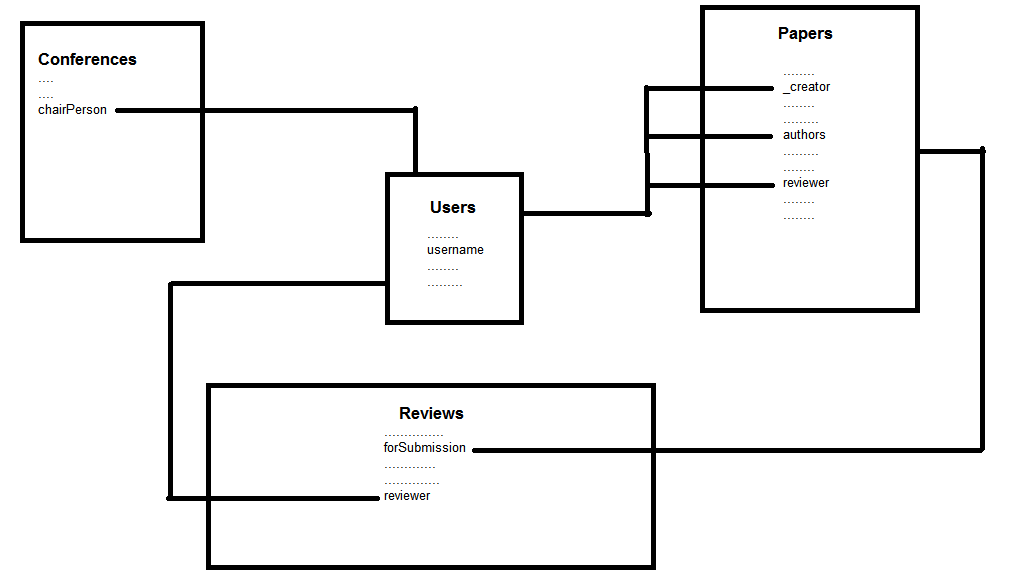


Figure 3 Backend document relationships

The above diagram illustrates how the documents formed based on the defined schema are linked to each other. Each of the references are formed according to the logical relationships. For example, papers can be authored by a user only. So author field in ‘Papers’ refers to the ‘User’ model.

# Low Level Design

This section describes the low level technicalities of the application design. Each of the sections could be traced back to the list of requirements.

## General requirements – GR

### GR - Responsive web design

This application should be able to scale to different resolutions and support different devices as well. We use ‘bootstrap’ for building the UI along with AngularJS to build the ability for the application to cater to all client types (PC, Mobile, Tablet). Bootstrap uses a grid system with which it is able to correctly scale the application at different resolutions.

For example, a UI page can be represented as being 12 columns and the sub sections can split and use these grids. Our application consistently uses a 12 column grid size.

## User Management – UM

### UM- Registration

Users can enter their profile details and create a valid user account in the system. A typical code flow in MEAN stack is employed to achieve this. The following actions are involved.

* The front end HTML page obtains input values for the various user data fields and calls the angular controller method.
* The data fields are packaged under “userdata” ng-model.
* Controller method passes the data to a factory method which in turn calls the middleware POST API with $http service to save the details as a document in the DB.
* Subsequently, a session token is generated and saved and the user is landed on the home screen.

**Logic:**

register(userdata){

>$http.post (/register) [ saves in DB ]

onSuccess : savetoken and goto Home page in UI

onFailure: Prompt to try again

}

#### UM- Roles

A user can be a ‘Chairman’ or a ‘Regular’ user. In our application, if a user’s id exists under ‘chairPerson’ field in ‘Conference’ Schema, then the user is a chairman.

Logic:

If conference.chairperson contains loggedInUserID

Chairperson=true

Else

Chairperson=false

This is identified during login when the ‘sidebar’ is rendered. Menu options are displayed according to this user profile.

### UM- User Profile -Save/Edit

A user should be able to edit the profile details he/she entered during signup at any point as long as their account exists in the system. After the user logs in, they are provided an option to view or edit their profile. From this view, they can reenter the data and save.

**Logic:**

saveProfileData(modifiedData){

make $http.put(/updatedata) call [updates in DB]

onSuccess: reload UI state and display modified data

onFailure: Don’t save but prompt again to retry

}

### UM- User Profile –Remove

A user can choose to delete his account from the system

**Logic:**

From logged in state,

deleteAccount(){

call $http.delete(data) [deletes from DB]

onSuccess: logout

onFailure: Prompt to retry

}

### UM- Login

Users can login with a valid username and password combination. Our application uses passport.js to make this process simpler. Token generation, management and such are taken care of by this module.

**Logic:**

If(enteredusername and hash of enteredpassword same as savedusername and savedhash)

Generate token and land in home

Else

Prompt to retry with correct combination

### UM- Logout

Users can choose to logout from the system.

**Logic:**

Logout(){

//clear session values

//land in login page again

}

## Author Functions - AF

### AF- Make a paper submission

Any valid user can login using his/her credentials and choose to make a submission.

**Logic:**

With valid session,

createpaper(inputdata){

call $http.post [creates paper with input fields]

upload(file) [call $http upload api

onSuccess: land in a new page and show user submissions

onFailure: prompt to retry

}

### AF- Access the submissions

Any valid user who has previously made a submission can view all of his/her submissions.

**Logic:**

With valid session

getMyPapers(inputData){

call $http get(mypapers) with inputData.userid[retrieves correct papers from Papers Schema]

(compares input user id with ‘authors’ field in ‘Papers’]

onSuccess: List all retrieved submissions

onFailure: Prompt to retry

}

### AF-View a Submission

Any user who has previously made a submission can view a list of his/her submissions as per 5.3.2. He/She can also click on any of those entries to view all the previously entered details including the uploaded file.

**Logic:**

viewOneSubmission(paperid){

retrieve data from $rootscope and display

}

### AF-Edit a Submission

Any user who has previously made a submission can view a list of his/her submissions as per 5.3.2.

By clicking on any of those entries in the list, the user can view the selected entry in detail with editable fields. Any changes made in the fields in this page can be saved by clicking on the save button.

**Logic:**

updateSubmission(modifiedinputdata){

call $http.put [updates paper with modified input fields]

only if(file is changed)

delete(oldfile)

upload(newfile) [call $http upload api]

onSuccess: land in a new page and show user submissions

onFailure: prompt to retry

}

### AF-Withdraw a Submission

Any user who has previously made a submission can view a list of his/her submissions as per 5.3.2. He/She can click on any of those entries to view in detail and then choose to “Withdraw” the submission as well.

**Logic:**

Withdraw(paperid){

Call $http.delete (remove entries based on paperid from backend)

//delete uploaded file as well

}

## Reviewer Functions –RF

### RF- Access assigned submissions

Any user with a valid login credentials can login and choose to view all submissions assigned to him/her to review.

**Logic:**

getMyAssignments(userid)

{

call $http get(myreviews) with inputData.userid[retrieves correct papers to review from Papers Schema]

(compares input user id with ‘reviewer’ field in ‘Papers’]

onSuccess: List all retrieved submissions

onFailure: Prompt to retry

}

### RF- View an assigned submission individually

Any user who has previously made a submission can view a list of his/her assignments as per 5.4.1. He/She can also click on any of those entries to view all the previously entered/empty input fields including the uploaded file.

**Logic:**

viewOneSubmission(paperid){

retrieve data from $rootscope and display

}

### RF- Make a review

A user who is viewing a single submission assigned against him/her as per 5.4.2 can download the paper, review offline and then click on “Make review” to enter all the post review data. This data is then saved with a http call.

**Logic:**

Submitreview(reviewdata){

//call $http.post to reviews schema with reviewdata [API saves the sent data in the DB)

onSuccess:land on my reviews page

onFailure: prompt to retry

}

### RF- Edit a submitted review

From the page with list of assignments as per 5.4.1, the user can click on any of the entries to view his/her previously saved reviews. Any changes made in the page can be saved again.

mofidyReview(newdata){

$http.put call to reviews(with new data) 🡪API updates the DB

}

## Chairman Functions –CF

### CF- List all submissions

A user with chairman privileges can login with his/her valid credentials and choose to view all submissions made for the conference as a list.

**Logic:**

getAllSubmissions(){

$http.get(allpapers) [API retrieves all documents under PaperSchema]

onSuccess: display all documents

onFailure: Log error and prompt to try again

}

#### CF- View a submission

From the page with all submissions as per 5.5.1, the chairman can click on any of those entries to view the individual submission in detail and download the attached file if necessary.

**Logic:**

viewOneSubmission(paperid){

retrieve correct document from $rootscope

onSuccess: display all details

if(downloadfile= =yes)

make downloadfile call

}

5.5.1.2 CF- Withdraw a submission

After clicking on one entry from the list of submissions and landing on the detailed view page as mentioned in 5.5.5.1, the chairman can choose to withdraw/delete a submission from the conference.

**Logic:**

Withdraw(paperid){

Make $http.delete(paperid) [API removes document from the DB]

Also call API to deletefile also

}

### CF -List all authors with details

A chairman can view a list of all the authors participating in the conference grouped by the paper that they have authored.

**Logic:**

viewAllAuthors(){

make a $http get call to Papers and retrieve all documents

(populate ‘authors’ reference field)

Display all the details under authors object in a table

}

### CF- List all reviewers with details

A chairman can view a list of all the reviewers participating in the conference grouped by the paper that they have reviewed/ they have been assigned to review.

**Logic:**

viewAllReviewers(){

make a $http get call to Papers and retrieve all documents

(populate ‘reviewer’ reference field)

Display all the details under reviewer object in a table

}

### CF - Assign papers

A chairman can manage papers and assignment of papers to potential reviewers after establishing a valid session after authentication.

**Logic:**

AssignReviewer(paperid, userid){

Only If(paperid.author not userid)

$http put call to PaperSchema to update userid under “reviewer” reference field.

}

### CF- Schedule management

5.5.5.1 CF- Automatic

The submission of papers or submission of reviews is not possible once the submission deadline and review deadline have passed already.

**Logic:**

During login after conference date updation,

Make call to conference Schema to get submission and review deadlines.

Only If(submissiondeadline is later than current date)

Show submission/modification options

Only if (reviewdeadline is later than current date)

Show review/modification options

5.5.5.2 CF- Manual

The chairman can manually change the submissiondeadline and reviewdeadline at any point which would mean the submission/review menu gets enabled again.

**Logic:**

UpdateConfdata(newsubdeadline,newreviewdeadline){

Make $http put call to Conference [API updates the dates in Conference Schema]

}

### CF- Graphical charts

The chairman can view statistics related to conference in the form of charts for easy interpretation.

Currently our application shows a chart mapping papers that are accepted/completed/rejected

**Logic:**

renderChart(){

$http get call to Paper [API fetches all paper documents]

Count papers in each status

Render chart based on data

}

### CF- Additional chair

The chairman can add another user registered in the application as a chairman also.

When updating conference, he can enter other registered usernames and make them chairman.

**Logic:**

addChair(newchairuserid){

make $http put call to ConferenceSchema [API updates and adds the new user id to chairperson list]

}

Appendix A: Supplementary documents

* Traceability Matrix



* List of REST routes

