Project Name – ToDo Manager

Frontend Technologies allowed - HTML5, CSS3, JS, Bootstrap and Angular

Backend Technologies allowed - NodeJS, ExpressJS and Socket.IO

Database Allowed - MongoDB and Redis

**Application Index:**

1. User management System
   * 1. Signup
     2. Login
     3. Forgot password
2. To do list management (single user)
   1. ToDo List
   2. Create button
   3. List CRUD
   4. Sub-todo-items - child
   5. Mark an item as "done" or "open".
   6. Old ToDo Lists
3. Friend List
   1. Friend requests
   2. Friend list.
   3. Real time friends notifications.
4. To do List management (multi-user)
   1. CRUD actions by friends on user todo list
   2. Real time action notification
   3. Undo-redo actions
   4. Undo by button or shortcuts
5. Error Views and messages
6. Checklist

7 Documentation

1. Deliverables

# User management System

1. *Signup - User should be able to sign up on the platform providing all details like FirstName, LastName, Email and Mobile number. Country code for mobile number (like 91 for India) should also be stored. You may find the country code data on these links (*[*http://country.io/phone.json,http://country.io/names.json*](http://country.io/phone.json,http://country.io/names.json)*)*
2. *Login - user should be able to login using the credentials provided at signup.*
3. *Forgot password - User should be able to recover password using a link or code on email. You may use Nodemailer to send emails. (Please use a dummy gmail account, not your real account).*

**Backend**

**Model : userModel:**

userId : string, unique, index

firstName: string

lastName : string

password : string

email : string

mobileNumber : number

countryCode : string

createdOn : date

functionalities :

1. *Signup - User should be able to sign up on the platform providing all details like FirstName, LastName, Email and Mobile number. Country code for mobile number (like 91 for India) should also be stored. You may find the country code data on these links (*[*http://country.io/phone.json,http://country.io/names.json*](http://country.io/phone.json,http://country.io/names.json)*)*

*Solution:*

1. *Login - user should be able to login using the credentials provided at signup.*

*Solution:*

1. *Forgot password - User should be able to recover password using a link or code on email. You may use Nodemailer to send emails. (Please use a dummy gmail account, not your real account).*

*Solution :*

* *On clicking forgot password user is redirected to a page that gives input fields to provide email and mobileNumber. This email and mobileNumber must be same as provided during signup.*
* *On submitting server checks if the email and mobileNumber are same as provided during signup.*
* *If they are same a response containing a token with three minutes of validity will be provided*
* *The client can then use this response to convert it into a link for the user to edit the password.*

# To do list management (single user)

* 1. *Once user logs into the system, he should see an option to create a ToDo List*
  2. *User should be able to create, a new empty list, by clicking on a create button*
  3. *User should be able to add, delete and edit items to the list*
  4. *User should also be able to add sub-todo-items, as child of any item node. Such that, complete list should take a tree shape, with items and their child items.*
  5. *User should be able to mark an item as "done" or "open".*
  6. *User should be able to see his old ToDo Lists, once logged in.*

**Backend:**

**Model : listModel**

listId: string,

previousId : string, (= listId after modification)

listStatus : string , default open, (options?open,done)

createdBy: string,

modifiedBy: string,

createdDate: string,

updatedDate: string,

description: string,

modified: Boolean, default : false,

isDeleted : Boolean, default false, --- used for displaying on the list and manipulating

deletedBy : string, =>(username),

listTitle : string,

isChildOf : string (listId)

listTitle : {type : String, default : ''},

    itemDescription : {type : String, default : ''},

    itemId : {type : String, default : '', index : true, unique : true},

    itemPreviousId : {type : String, default : ''},

    itemModifiedId : {type : String, default : ''},

    itemOwner : {type : String, default : ''},

    itemModifiedBy : {type : String, default : ''},

    itemParent : {type : String, default : ''},

    itemChildren : {type : Object, default : {}},

    itemDueDate : {type : Date, default : ''},

    itemStatus : {type : String, default : 'open'},

    itemIsDeleted : {type : Boolean, default : false}

**Solution design:**

*Once user logs into the system, he should see an option to create a ToDo List . User should be able to create, a new empty list, by clicking on a create button*

Solution :

createNewListFunction:

* It takes userId and listTitle as parameters to create a new list with userId as the owner of the list and listTitle as the title of the list
* It takes listDescription as input to create a description for the list

*User should be able to add, delete and edit items to the list. User should also be able to add sub-todo-items, as child of any item node. Such that, complete list should take a tree shape, with items and their child items.*

Solution:

createNewItemFunction: add new or sub-toDo-items

* This adds new items to existing lists
* It looks for existing item titles in the same list with status “open” and are not deleted. If there are no such items then it creates a new item in the list or as a sub-toDo-item

editItem function :

* This function edits existing items by creating a whole new item with same details other than the ones that were edited.
* This is done in order to provide for undo functionality where it can be reverted to the previous­ version.

editList function :

* This function first creates a new list document with edited values of the existing list version by setting “listIsHidden” to “false” making it display and then makes the current version’s “listIsHidden” to “true” making it to not display. This will emulate an editing action.

deleteItem function :

* This function deletes the selected item on passing the itemId.

deleteList function :

* This function can delete the list on passing the listId. This also deletes all the items of the list

getUserAllLists function :

* This gets a list of all the lists owned by the user whose userId is same as the ownerId of the list

getAllListItems function :

* This gets a list of all the Items with the given listId.

*User should be able to mark an item as "done" or "open".*

Solution:

markItemAsDone :

* this takes itemId of the item to be marked as done and updates it to done.

markItemAsOpen :

* this takes itemId of the item to be marked as open and updates it to open.

*User should be able to see his old ToDo Lists, once logged in.*

Solution:

# Friend List –

*User should be able to send friend requests, to the users on the system.*

Solution :

sendFriendRequest :

* this creates a request item with the sender’s details and the receiver’s userId

*Once requests are accepted, the friend should be added in user's friend list.*

Solution :

checkRequest:

* this checks for pending friend requests with “isFriend” as “false”, which will mean that the request has not yet been accepted

acceptFriend :

* this takes the request senders details from “checkRequest” function and sends a new request item with acceptors details with “isFriend” as “true”, to the sender, as a mutual sharing of details which emulates accepting friend request
* this also changes the request the receiver sent as “isFriend” to “true”

friendCheck

* this enables a check to be performed on selected user and retrieves friend data

*Friends should be notified, in real time using notifications.*

Solution: socket service takes care of all the real time notification functionalities

# To do List management (multi-user) –

*Friends should be able to edit, delete, update the list of the user.*

Solution

A middleware friendCheck is used to authorize the use of access to editing, updating and deleting a list and its content. Only the owner of the list and friends of owner are allowed to perform CRUD operations on list-items and edit delete and update on the list

*On every action, all friends should be notified, in real time, of what specific change is done by which friend. Also the list should be in sync with all friends, at any time, i.e. all actions should be reflected in real time.*

Solution : socket takes care of all the real time notification functionalities.

*Any friend should be able to undo, any number of actions, done in past. Each undo action, should remove the last change, done by any user. So, history of all actions should be persisted in database, so as, not to lose actions done in past.*

Solution:

undoAction : this function sets the previous version of the item to “isHidden” : “false”, which makes it display the previous version as current item. And sets the existing version as “isHidden” : “false” which makes is no more visible and by doing so emulates an undo action.

redoAction : this function sets the next version of the item to “isHidden” : “false”, which makes it display the previous version as current item. And sets the existing version as “isHidden” : “false” which makes is no more visible and by doing so emulates a redo action.

undoListAction : this function allows the users and their friends to undo the last action by setting the previous version’s property “listIsHidden” to “false” and the current version to “listIsHidden” to “true”, which displays the previous version to emulate undo action.

redoListAction : this function allows the users and their friends to redo the last action on the list by setting the next version’s property “listIsHidden” to “false” and the current version “listIsHidden” to “true” which displays the next version if available to display making it emulate redu action.

*Undo action should happen by a button on screen, as well as, through keyboard commands, which are "ctrl+z" for windows and "cmd+z" for mac.*

1. Error Views and messages - You have to handle each major error response (like 404 or 500) with a different page. Also, all kind of errors, exceptions and messages should be handled properly on frontend. The user should be aware all the time on frontend about what is happening in the system.

checklist

* Rate limiting - Your APIs should have pagination or rate limiting to avoid send huge chunks of messages as API response.
* Quality of JavaScript code - Your application's Javascript code should contain proper indentation and comments. It should be broken down into functions for better maintainability and it should not contain any logical bugs. It should use modern javascript as much as possible.
* Responsive Design -Your website should be full responsive.
* Intuitive - make the platform as easy to understand as possible. You have think about all the possible error cases and you have to handle them by giving alert messages to user. You must use elements like progress bars and loaders to handle the UX better.
* Quality of Frontend application(Angular application) - All the best practices associated with Angular must be followed.
* Quality of Backend application - All the best practices associated with Advance rest API development must be followed.

Documentation –

* Documentation should be done properly as taught to you in training. Documentations of REST API endpoints and Socket Endpoints should be separate.

Deliverables from Candidate

1. Project description - A TXT/Doc file containing the description of your project and all your assumptions. It should also describe the features of the project and also any extra features that you have coded.
2. Github repository link of this project should be mentioned in the TXT file.
3. You have to host the built versions of the applications on AWS and mention the URL of that application in the TXT file. URL of both Frontend and REST API must be mentioned along with documentation link(if any).