Meetings Application Requirements

EXPECTED TIME: The time taken for completing this application is expected to be roughly 3 – 4 full days.

Overview of the Application

You are to build an application that maintains meetings for a user (say, yourself). This application helps you manage your meetings - you can filter to view meetings (past, present and future), or search for meetings based on the meetings description. You can add (i.e. create) meetings. **You will by default be part of a meeting you create.** The users added when creating a meeting will automatically be part of the meeting (no concept of accepting a meeting!). However, they can excuse themselves from the meeting (drop off from a scheduled meeting).

The application helps you view your meetings for a day in a calendar view.

Additionally, you can create teams consisting of yourself and other users. **You will by default be part of a team you create**. You can view all teams that you are part of and add members of these teams. You can excuse yourself from the team (leave the team). Once you do so you will not be able to view the team details, nor be part of new meetings where the team has been added (i.e. team members are attendees).

NOTES

- The application will be fully functional even without the concept of teams. **Build the application without the teams feature – make necessary changes to add teams feature only once all other functionality (both frontend and backend) is completed.**
- Handle both success and error scenarios in both backend and frontend apps. Make sure to validate data both in backend and frontend apps.

Assumptions regarding a meeting

- it starts and ends on the same day

- it will not overlap with any other meetings for any user who is part of the meeting (this is not strictly required from the application point of view – this assumption will simplify the calendar view, as you can assume the calendar entries will not overlap) – **no check is necessary for this criteria when creating a meeting**.

A meeting has the following details

- date of meeting
- start and end times (hours 0 – 23, and minutes 0 – 59)
- description (text)
- email ids of attendees (users who will be part of the meeting) / a team's short name (comma-separated) – All members who are part of a team that's added will become attendees of the meeting.

The detailed requirements for the application in general, and User Interface in particular, follow.

Application and the User Interface

The application has a **login / sign up view – this is to be designed by you**. Once a registered user logs in, they will be able to see the following views

- **Calendar**
- **Teams**
- **Meetings**

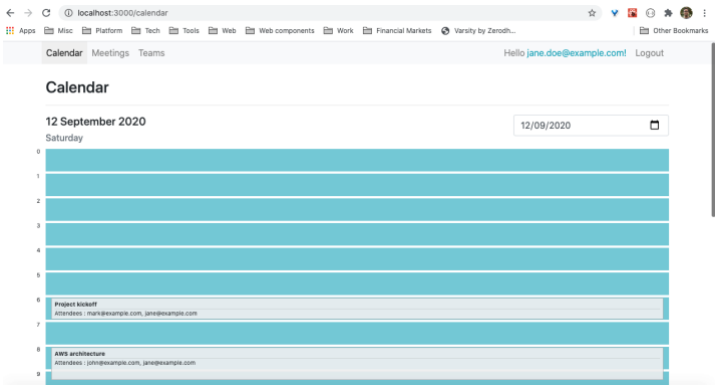
The URLs to be associated with the views are shown in the screenshot – please follow the same when implementing the app.

NOTES:

- The screens are suggestive only. Start with a simple interface and improve once you complete the functionality (i.e. after backend + frontend works fine together). Instead of a graphical calendar, you have a list of meetings for the day, to start with.
- Teams tab (and team related features) may be implemented once the rest of the application is complete in all respects. **Prioritize important features first, and functionality over looks.**

Calendar

This shows the meetings for a day for the logged in user. By default the meetings for current day are shown. The day can be selected using a date picker.



Calendar view

Meetings

You can do two main tasks here

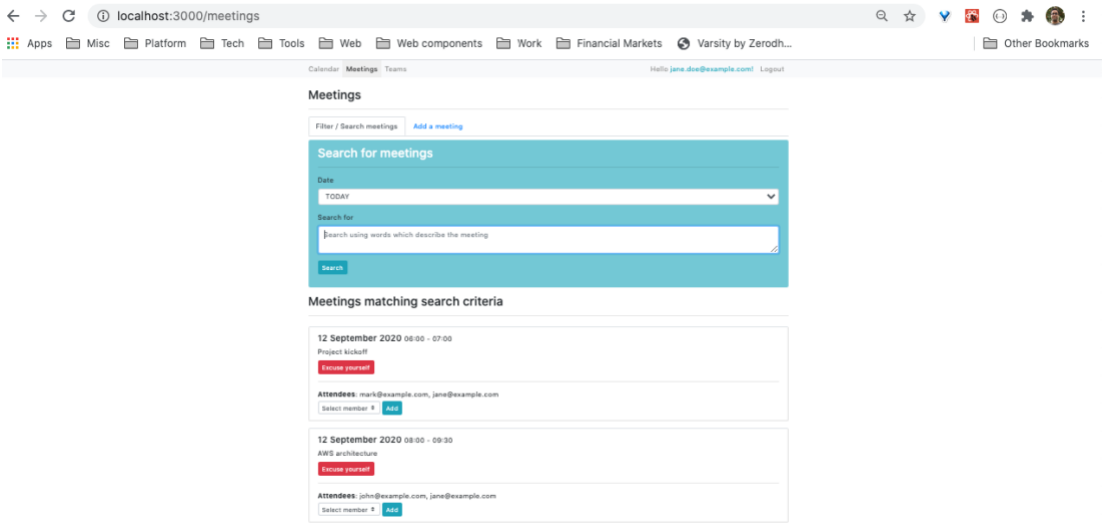
- **Task 1 : Filter / Search meetings** (we will combinedly call it search) for meetings you are part of – additionally you can do the following with results (existing meetings you are part of) that appear on search
    - excuse yourself, i.e. leave a meeting
    - add new attendees (i.e. users)
- **Task 2: Add a meeting (i.e. create a new meeting)**

The two main tasks can be accessed using a tab view like below. When adding a team's short name (eg. @annual-day), all team members must be added as attendees (in the backend). Make sure not to add an attendee's email id twice for a meeting (check must be made in the backend).

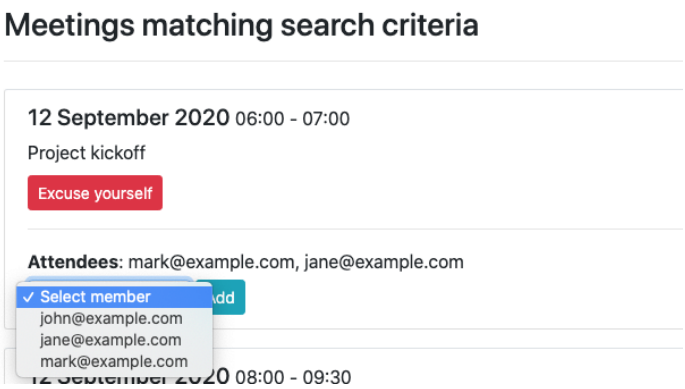
NOTE: Only meetings you are part of must appear in the results.

Task 1: Filter / Search meetings

The UI shows the filter/search input and the results that appear once the backend returns the matching results. You can leave a meeting, or add attendees using the dropdown of user email ids (shown separately).



Filter/Search input with results once backend returns the results



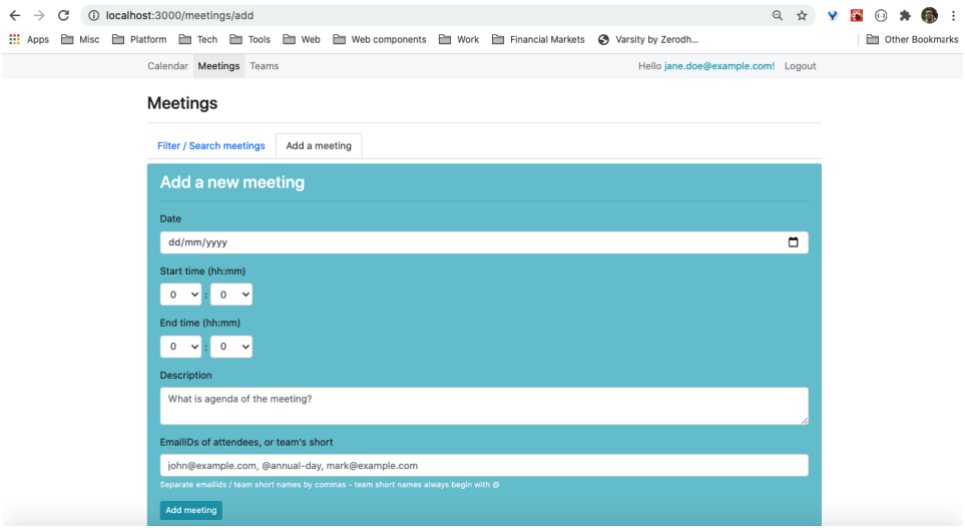
Dropdown of user email ids in search results

NOTES:

- The search / filter dropdown has options "ALL", "PAST", "TODAY", "UPCOMING" (self-explanatory). On selecting an option, the meetings are displayed in chronological order (earliest meeting first). "TODAY" is selected in this dropdown by default.
- To match the search terms entered in the textarea – the matched meeting should have AT LEAST ONE of the search terms in its description should show up (additionally of course, the dropdown conditions are also met).
- The search results list view, by default shows the meetings for the current day (since "TODAY" is selected by default in the filter meetings dropdown).

Task 2 : Add a meeting

You can add a meeting (i.e. create a new meeting) – when doing so, the email id of participants, or team short name, can be given for attendees (separated by commas). If interested you can even use tags to suggest email ids / team short names (https://www.npmjs.com/package/react-tag-autocomplete). Remember to add user who creates a meeting as one of the attendees.

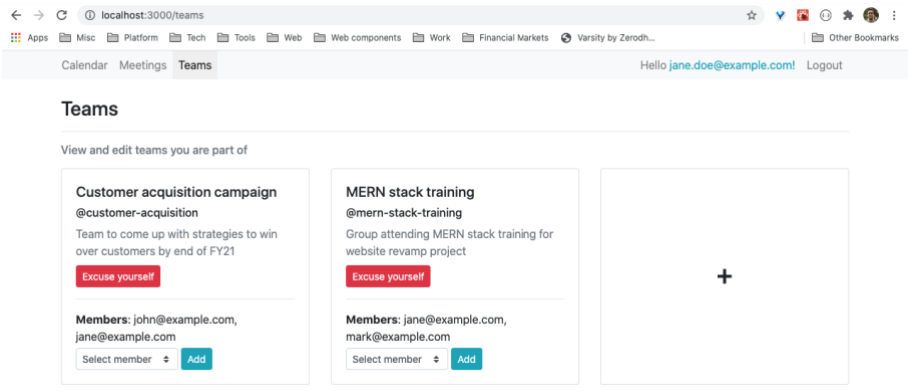


Add a meeting

Teams

You can view the details of teams you are part of, and add members to them. You can excuse yourself from the team (leave the team), but cannot remove anyone. A team has a name, a short name (begins with @), a description, and a list of users (identified by their email ids). In order to add a user to a team, a dropdown with user names exists – this is similar to the interface to add a user (as attendee) to a meeting (refer earlier screenshot).

NOTE: When the + card is clicked, a form appears inside the card, using which a team is created - **You have to design the UI for this form**.



Teams View

Additional Feature 1

Design and implement a profile page for a logged in user. The link to navigate to the profile page appears on the top-right (in navigation menu).

Every user now gets to add a profile picture. This can be uploaded via the profile page. You can use an npm package like multer to upload files - https://www.npmjs.com/package/multer

Start off by storing the files in a folder in the server (say profile-images/ folder. You can generate a filename as per username of user. This way the path to user's image on server is well known – you can also make a note of the path in the user model.

Next upload the files to an S3 bucket instead of local folder on the server. For this you can use the official AWS SDK for Node.js - https://www.npmjs.com/package/aws-sdk

Additionally user must be able to change password (after confirmation of current password) and also update Email ID – remember that this will result in corresponding changes in all of the places the email id is duplicated.

If Email id is used as the primary key you may omit this feature. Instead you can implement this – use Imagemagick tool (there are many npm packages that use this tool to resize images – find an appropriate one) to crop and resize to create an image of size 48px x 48px, and upload this one to S3 too – this image is to be displayed in the profile pic in the top-right in the navigation menu.

Additional Feature 2

Design and implement changes for role-based access to the application for admin and other (normal) users. The normal users get to see whatever was discussed above – i.e. the Calendar, Teams and Meetings.

Maintain a field indicating if a logged in user is admin. If so, a different UI appears. This is described below.

There will be 3 tabs.

- Users
- Teams
- Meetings

Each of these views shows a data table with list of users, teams and meetings.

You can use a React JS data table implementation like
https://www.npmjs.com/package/material-table
or https://mdbootstrap.com/docs/react/tables/datatables/

The admin must be able to edit and delete users, team and meetings through these data table views (and sort, search and filter).

Recommended approach to creating the application

STEP 0: Before you start to code

Understand the requirements clearly. Raise questions with concerned people in case there is lack of clarity in some requirement, project timelines etc.

First, come up with a design (at least high-level) before developing your application. Don't worry - you most likely will deviate from the initial design. It is but natural.

STEP 1: Design the backend (DB + API Design)

Function comes before form. An app that has no GUI but does what is intended is still useful. An app that looks good but does nothing is only a piece of art – maybe a joy to build, and admire, but unable to solve the problem at hand.

STEP 1a: Design the Database

When designing the Database in MongoDB, come up with entities, decide how the entities are related, and how you will represent those relationships. Keep in mind any data in collections that might be useful to duplicate in another collection.

STEP 1b: Design the API

For every feature listed in the requirement make sure you have a very clear idea of the API that will be called from the frontend and what inputs need to be passed and how (query params, path params, request body, HTTP headers etc.), and response to be sent.

Decide the routes required by the application. Come up with a design based on RESTful web API design principles. For each route, make sure you understand what inputs are expected (path params, query params, request body, HTTP headers if any), what database queries need to be made to service the request, and what possible responses need to be sent. Carefully construct the DB queries including selection criteria, fields to project etc. In case of updates, make sure you understand what queries need to be made on collection(s), and which update operators to apply to get desired result.

STEP 2: Design the frontend app in React

As a first step it is better to decide what components you will have in the application, the props for each and how they will interact (via props for example), state to be maintained by the application, and any of its components. This will not be very clear in the beginning and will change as you go through the process of coding your application. However, it is a good idea to come up with the high-level structure, especially the component hierarchy.

a. List out the different components you would create for this application. Make sure to include a top-level component to enclose the application UI.
b. What will the UI of each of these components show?
c. Which of these will be stateful (i.e. the UI would change with time based on time-varying data) and which of these stateless (i.e. once rendered, the UI would never change)?
d. What is the hierarchy of the components (we call this the "Component tree")?
e. What data would the parent components need to pass to each of their child components (i.e. what are the props)? Decide the data types for each of these props.
f. What will the state of this application / its components have? In which component(s) will you store the state? What was the reason for your choice?

NOTE: When state is to be shared between 2 or more components, it is stored in the nearest common ancestor of the components. For more details check the React documentation https://reactjs.org/docs/lifting-state-up.html

STEP 3: Develop the application

Use Create React App to set up the project. Follow your design to come up with the application code.

- Make sure to modularize and organize your code. Create folders for routes, controllers, utilities, models, database operations in the backend, and components, services, utilities, models etc. in the frontend.
- Decide the order in which you will implement the features. For example, you may first create a backend which implements some features – example – user sign up, and sign in / sign out (logout) and authenticated access to APIs. Then you can implement meeting related API – first adding a meeting, then search for meetings etc. Then you can work on the calendar API, and finally the APIs to implement teams, adding attendees using a team etc.

Finally test your applications works well in all use cases.