

Meetings Application Requirements

EXPECTED TIME: The time taken for completing this application is expected to be roughly 3 – 5 full days (varies depending on enforcement of non-functional requirements).

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.

Tech Stack

Front-end: HTML, Sass / Less / CSS, TypeScript / JavaScript. Additionally jQuery / React / Angular / Vue may be used (only if the training covers it).

Back-end: Node.js, MongoDB / MySQL

The app must be deployed in a PaaS / IaaS like Heroku, S3, EC2 etc. and should have a deployment workflow configured.

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**

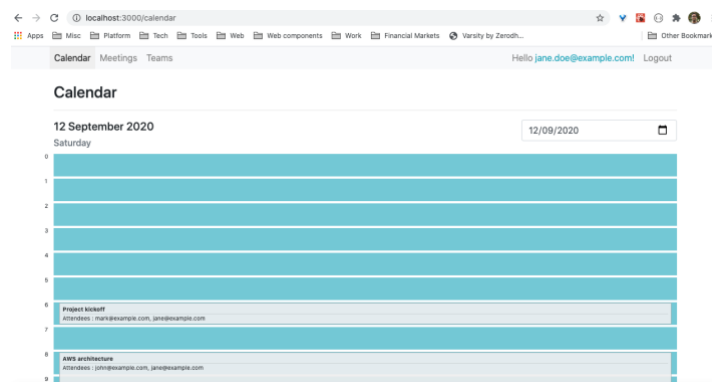
The URLs to be associated with the views are shown in the screenshot – please follow the same when implementing the app.

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

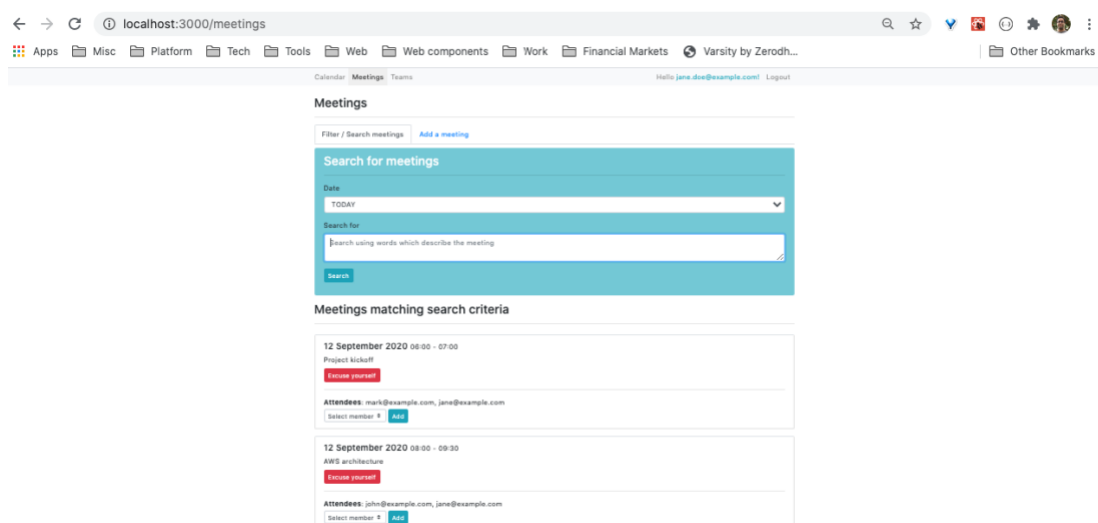
- **Task 1 : Filter / Search meetings** (we will combinedly call it search) for meetings you are part of – additionally you can do the following with results (existing meetings you are part of) that appear on search
 - excuse yourself, i.e. leave a meeting
 - add new attendees (i.e. users)
- **Task 2: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

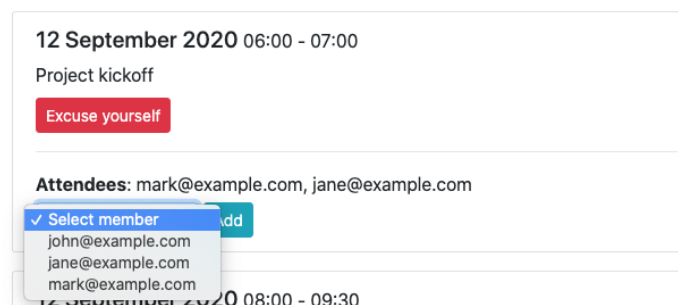
Task 1: Filter / Search meetings

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

Meetings matching search criteria



The screenshot shows a list of meetings. The first meeting is on 12 September 2020 from 06:00 to 07:00, titled 'Project kickoff'. It has a red button labeled 'Excuse yourself'. Below the title, it lists attendees: 'mark@example.com, jane@example.com'. A dropdown menu is open, showing a 'Select member' option with a checkmark, and a list of email addresses: 'john@example.com', 'jane@example.com', and 'mark@example.com'. To the right of the dropdown is a blue button labeled 'Add'. Below the dropdown, the start of the next meeting is visible: '12 September 2020 08:00 - 09:30'.

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.

The screenshot shows a web browser at localhost:3000/meetings/add. The page has a header with navigation links (Calendar, Meetings, Teams) and a user profile (Jane Doe). The main content area is titled 'Meetings' and contains a form to 'Add a new meeting'. The form fields include: Date (dd/mm/yyyy), Start time (hh:mm), End time (hh:mm), Description (text area), and Email IDs of attendees (comma-separated). A blue 'Add meeting' button is at the bottom.

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.

The screenshot shows the 'Teams' page in the web application. It displays two team cards and a third card with a plus sign. Each card shows the team name, short name, description, and a list of members. The first card is 'Customer acquisition campaign' with members John and Jane. The second card is 'MERN stack training' with members Jane and Mark. The third card is a placeholder with a plus sign.

Teams View

Additional Features

1. Implementing a profile page with profile pic upload

Every user now gets to add a profile picture, and change name, email and password. Design and implement a profile page for a logged in user. The link to navigate to the page appears on the top-right (in navigation menu) – the link will be a miniature user profile image (use a default image if none is available).

Name, email and password can be updated on profile page. If user edits password, ask for confirmation of new password, and also the old password. You can use an npm package like multer to upload files - <https://www.npmjs.com/package/multer>

Store 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.

NOTE: On email update, all teams where the user is a member has to be updated with new email of user. This need not be done for meetings.

2. Resizing profile images

Let's say, the administrator of the site uploads profile pictures of users to a folder on the server.

Write a Node.js script that runs these files through <https://www.npmjs.com/package/sharp> or Imagemagick tool to automatically crop and resize to create images of the following sizes

- i) 48px x 48px
- ii) 200px x 200px

The generated images are stored in a separate folder – you can name the files like so – an source image with name aravind.png will generate aravind-48x48.png (or .jpeg) and aravind-200x200.png (or .jpeg).

The sizes mentioned above are the default sizes. When running the script, the user can specify command line arguments to specify multiple different sizes – images of those sizes should be generated instead. You can use <https://www.npmjs.com/package/minimist> to make a command line utility that accepts options (design some interesting options)

3. Implementing upload of files to S3

Let's say, the administrator of the site uploads profile pictures of users to a folder on the server.

Write a Node.js script that uploads the pictures to an S3 bucket. Modify the user model to store the path to the image on S3 – once the image is successfully uploaded, the path on S3 to the user's profile image should be stored in the user document.

For this you can use the official AWS SDK for Node.js - <https://www.npmjs.com/package/aws-sdk>

4. Meetings and Teams upload by admin

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.

The administrator when logged in sees only a single page with 2 buttons

- Upload meetings
- Upload teams

These are essentially form upload buttons. The file upload should report error to admin if files other than .xlsx files are uploaded. The uploaded files should be stored in a folder on the server.

You can use an npm package like multer to upload files - <https://www.npmjs.com/package/multer>

Store the files in a folder in the server - say **admin-uploads/meetings** for meetings and **admin-uploads/teams** for teams. You can generate a filename as per time of upload. This way the path to files on server is unique – you can also make a note of the path, and time of upload in a log file.

5. Adding Meetings and Teams using excel files

Let's say, the administrator of the site uploads excel files, with new teams and meetings to be added to the database, to a folder on the server (says **admin-uploads/meetings**, and **admin-uploads/teams**).

Write a Node.js script that reads a files and creates meetings / teams according to the entries in the file. Use a package like <https://www.npmjs.com/package/xlsx> (Documentation: <https://sheetjs.com/>)

to parse Excel files. You can use <https://www.npmjs.com/package/minimist> to make a command line utility that accepts options (design some interesting options).

6. Sending emails when a meeting is scheduled / modified

When a meeting is created, all attendees should receive an email. To implement this use a package like <https://www.npmjs.com/package/nodemailer> (Documentation: <https://nodemailer.com/about/>). Similar email is to be sent to attendee when he/she is added to a meeting.

NOTE: To test this feature you can create a couple of Gmail accounts. For using Nodemailer to send emails to Gmail, check the Nodemailer documentation here -

<https://nodemailer.com/usage/using-gmail/>

EXTRA CREDITS: Attach a calendar invitation with this email

7. Creating a meeting invite file (.ical) for a scheduled meeting

Write a Node.js script that takes the meeting id as a command line argument and generates a meeting invite in .ical format. You can use a package like <https://www.npmjs.com/package/ical->

[generator](#). The file may be stored in an assets/meetings folder with the meeting id as its name (and .ical extension). A note of its path can be made in the meetings model (in a new field).

8. Admin panel

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.

- User
- Teams
- Meetings

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

For React, use a data table implementation like

<https://www.npmjs.com/package/material-table>

or <https://mdbootstrap.com/docs/react/tables/datatables/>

For Vue, use a data table implementation like

<https://vuetifyjs.com/en/components/data-tables/#usage>

or choose one listed on <https://madewithvuejs.com/blog/best-vue-js-datatables>

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

9. Profile page feature with profile pic resized and uploaded to S3

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>

Also run Imagemagick tool (there are many npm packages that use this tool to resize images – find an appropriate one) to automatically 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.

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.

Recommended approach to creating the application

NOTE: If you are building only the frontend / backend of the app, you can ignore steps mentioned below accordingly. For the frontend you may choose one of Vanilla JS/jQuery/React/Angular/Vue as per whatever is covered as part of training.

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 Vanilla JS/React/Angular/Vue)

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 for example in React, input attributes in Angular etc.)? Decide the data types for each of these.
- 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: For React/Vue, 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 Vanilla JS/jQuery/React/Vue/Angular 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.

Non-functional requirements (front-end)

The application must function as per requirements given. Additionally the following non-functional aspects must be taken care of.

1. The site must be responsive using a mobile-first approach, fully functional, and look good across a variety of mobile devices.
2. Take care of Adaptive Web Design Practices where applicable.
3. The application must be compatible across popular desktop and mobile web browsers.
4. Site must be SEO friendly. Site has a site-map.
5. Every page of the site must be accessible using keyboard and popular screen readers.
This includes page structure, navigation, forms, media and custom widgets.
6. The site must be performant and every page must load fast.
7. Project must be structured well.
8. All code has unit tests, and project shall follow TDD approach.

Acceptance Criteria (front-end)

All features must work as mentioned. This is an absolute must. Apart from the features, the non-functional requirements must be met. In particular,

1. The application works well on devices of various form factors, and platforms (tested on iPhone 6+ if possible, and on some popular Android phones, and tablets).
2. The calendar page must be print-friendly.
3. The application MUST (manually) tested in most, if not all of, IE 11+, Edge, Safari, Chrome, Firefox and popular mobile browsers (versions released over last 4 years)
4. HTML is semantically correct. Site is well-structured, has no broken links, and has a site-map. Relevant metadata for pages and page elements is added.
5. The application passes WCAG AA level of compliance. In particular, right ARIA attributes are used to make all custom components accessible through keyboard. At least one of VoiceOver with Safari on Mac / JAWS or NVDA with Chrome / Firefox / Edge on Windows has been used for testing Screen Reader based access.
6. Performance optimization with respect to front end must be made in HTML, Sass / Less / CSS and JS / DOM. Memory leaks should not be present. Animations must be smooth. All necessary steps are taken to ensure every page is optimized for fast page load.
7. Project is built as an SPA, uses an MVC or an MV* architecture, and is designed and developed in a way that is scalable and maintainable. The code is linted. Sass / Less is used for styling. TypeScript may optionally be used in place of JavaScript. TypeScript / JavaScript must be written using ES2015+ features and use a transpiler like Babel to have it compiled to ES5.
8. All code has unit tests, and project follows a TDD approach.
9. The project is deployed in a PaaS / IaaS and using a continuous deployment workflow whereby deployment happens only after unit tests pass, and code coverage is at least

75%. (additionally other criteria like automated a11y tests etc. may be added – but this is optional).

Project milestones (front-end)

The project shall be built by participants as the training progresses. Assuming a 30-day extensive training on front-end technologies, the following shall be delivered as per deadlines below.

1. **Day 7:** HTML and CSS parts - Structure, layout and styles for all pages. AWD / RWD is taken care of. SEO is taken care of.
2. **Day 15:** A fully functional application that integrates with the backend, takes care of basic performance techniques, and is WCAG 2.1 AA level compliant.
3. **Day 22:** A better-structured and better-designed application that is performance-optimized. TDD practices deployed to create unit tests.
4. **Day 27:** A Single Page Application (SPA) which uses latest ES2015+ features, templates, routing and a component-based architecture.
5. **Day 30:** A better-structured workflow for the app using Webpack, and refactored using jQuery and ES2015+ (in place of Vanilla JS). More unit tests, and enforcement of text coverage. Code is additionally linted.