Challenger Test

Typescript / React

Overview

In this test you need to fix and extend an existing application. Imagine being hired to continue work on a project after your college who had started the work on it went on holidays.

One of the tested skills is the ability to work with the documentation, so please make sure to read this document to the very end before starting coding.

Requirements

The software is designed to be used by the spaceship captain to monitor current ship Crew and assign a job to the new crew Members.

Each member has the following properties:

- Id (string)
- firstName
- lastName
- joinStardate (float)
- log (string)
- job

There are 3 available Jobs on the ship:

- Medic
- Engineer
- Pilot

Until a crew member is assigned a job he has a virtual job - Unassigned

You are provided with an emulator, which emulates the crew events as well as network latency for asynchronous calls. Your task is to work on the client software that consumes the services provided with the emulator. Obviously you can NOT change the code of the emulator.

UI Specification

The UI consists of two screens: **Settings** and **Crew Members**,

Settings Screen

The screen has 3 inputs for Job Split, which is a desired distribution between available jobs represented as percentage values (user can only enter integers).

The screen has an “Apply” button that applies the values using an async call to the `settingsService`.

Crew Members Screen

The Crew Members Screen shows the current crew members as a table sorted by their last name

The table should update automatically when the related data changes. You can subscribe to the change events using the provided API.

Status Widget

In the top right corner we need to show the current Stardate (the usual float value, e.g. 1234.5) and the summary for the crew members, showing total numbers of members assigned to each job as well as the grand total of the crew members on the ship.

Auto-assign Feature

The new (unassigned) members must be assigned to their jobs automatically as soon as they appear in the system. This can be done using the `crewService` call. The decision should respect the settings described above.

Assign jobs only to the new unassigned members based on the latest settings. Do **NOT** reassign jobs to existing members

Example 1:
Crew: 3 medics, 2 engineers, 6 pilots
Settings: 25% medics, 25% engineers, 50% pilots
Decision: the new member should become an engineer

Example 2:
Crew: 2 medics, 2 engineers, 2 pilots
Settings: 40% medics, 40% engineers, 20% pilots
Decision: the new member should become either a medic or an engineer

The emulation always starts with a crew having 1 medic, 2 engineers and 1 pilot independent of the settings.

Important note! The function making the decision which job to assign should be implemented as a separate module independent of the React and UI. We'll consider moving to the backend at some point.

Technical Limitations

In your final solution:

- You **MUST NOT** change any code in `src/emulator`
- You **MUST** make changes only in `src/components`
- You **MUST NOT** `import` anything from `src/emulator/internals` (see details in the next section)
- You **MAY** use the third party libraries, but you **DO NOT NEED TO**. React and React Hooks are powerful enough
- Fixing issues identified by the QA is the priority
- You **MAY** do further improvements **ONLY AFTER** those major problems are solved

Emulator API Specification

The emulator resides in `src/emulator` folder. You shouldn't change code in any of the files within it in your final implementation.

You can only import (use) files placed directly under `src/emulator`, namely:

- crewService.ts
- settingsService.ts
- operationService.ts
- types.ts

You **CAN NOT** import anything under internals directory - this would kill the whole purpose of the emulator.

Some services provide methods for subscribing to events e.g. `onMembersAdded` or `onJobSplit`. Every such method returns an unsubscriber function (see type `Unsubscriber`). The returned function should be called without parameters to unsubscribe.

Crew Service

File: `src/emulator/crewService.ts`

This service allows watching Crew changes, loading the entire crew member list and assigning jobs to specific members.

Notice the difference between `getSummary()` and `getCrew()`. The summary gives you only the totals but returns the value immediately, while `getCrew` is an asynchronous call.

```
interface ICrewService {
  /**
   * synchronously returns an object containing job totals
   * and grand total of crew members
   */
  getSummary (): Summary

  /**
   * fires an event each time there is a member added
   * or changed proving Summary (same one as returned in getSummary())
   */
  onSummary (subscriber: Subscriber<Summary>): Unsubscriber

  /**
   * fired each time a new member is added
   */
  onMemberAdded (subscriber: Subscriber<Member>): Unsubscriber

  /**
   * fired each time the member information is updated, e.g.
   * - when the last name is changed
   * - when the job is changed
   *
   * Note: the event is fired even for the client which
   * previously called assignJob()
   */
  onMemberUpdated (subscriber: Subscriber<Member>): Unsubscriber

  /**
   * ASYNCHRONOUS call assigning Job to a crew member.
   * The subject member is identified by member.id field
   */
  assignJob (memberId: MemberId, job: Job): Promise<void>

  /**
   * ASYNCHRONOUS call retrieving all crew members as an array
   */
  getCrew (): Promise<Crew>
}
```

SettingsService

File: `src/emulator/settingsService.ts`

The services allows to get and set Job Spit settings.

Note, both getting and setting is synchronous.

```
interface ISettingsService {
  onJobSplit (subscriber: Subscriber<JobSplit>): Unsubscriber
  setJobSplit (jobSplit: JobSplit): void
  getJobSplit (): JobSplit
}
```

OperationSerice

File: `src/emulator/operationService.ts`

This is just a convenience service for controlling the emulation and getting current Stardate.

```
interface IOperationService {
  getStardate (): Stardate
  play(): void
  pause(): void
}
```

Note from the previous developer

Hello!

I started the work on this project just a day before my holidays, so I could only implement a part of it.

- I added Pause/Start Emulation button, so that we can first adjust the settings and then run the emulation.
- I implemented the Settings Screen and it seems to work fine
- I implemented the Status Widget and as you can see it always shows the actual totals as they change with time
- I only started working on Crew Members, but had no time to finish it. I made the basic design for the table, and as a test made it load the list on component mount
- I didn’t touch the Auto-assigning Feature at all

You may want to use the SettingPage as an example of subscription implementation.

Hopefully you can finish the missing parts in no time! Cheers!

Email from a QA engineer

Hi there!

Our team tested the application in its current state and made a checklist of all pieces that I found missing or not functioning compared to the specification.

Problems ordered by severity (the most important are on top):

1. The crew members aren't sorted by the last name
2. The table on the Crew Screen shows the list state only once, but as the new members are added or as the existing are getting updated the list doesn't change
   - I found a workaround - if you go to Settings and then back to Crew Members, then it updates
3. Auto-assigning feature doesn't work
4. The settings screen has no validation

About the code style. Try to uncomment `EXTEND_ESLINT=true` in `.env` file to keep the same code style, but it's really optional and has the lowest priority of all.