Option L (Hospital Manager)

**Time Remainingswitch to elapsed time**

4h 27m

**Due Date**

February 28, 2024 09:42 PM

Option L (Hospital Manager)

Welcome to the JavaScript Belt Exam! In this Belt Exam you will create a small application that allows users to manage a hospital by adding and viewing patients.

Your submission will be evaluated based on your performance in the eight competency areas detailed in the [**JavaScript Belt Rubric**](https://assets.codingdojo.com/boomyeah2015/codingdojo/curriculum/content/chapter/1695309592__rubric.png)​​.

Do everything specified in the visual wireframes and the descriptions provided.

Ask your instructor if you are not sure.

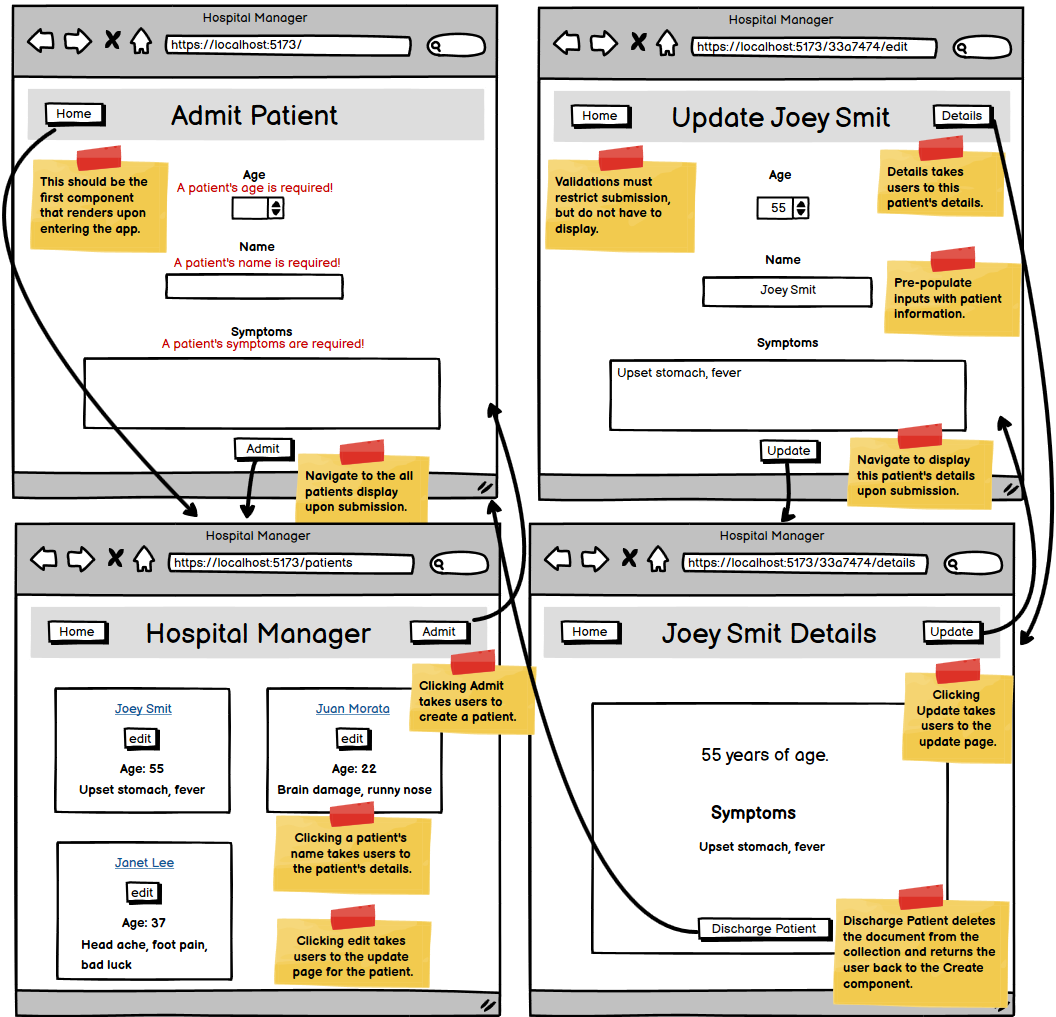
Red Belt vs. Black Belt Scoring

The core material you've completed during the course will allow you to prove proficient competency in each area, which will earn you the score of Red Belt on the Exam.

To earn a Black Belt, proficient competency must be met in all competency areas, in addition to obtaining **mastery-level competency in three of the five available competency areas**.

Red Belt and Base Wireframe

​[**Hospital Manager Wireframe**](https://assets.codingdojo.com/boomyeah2015/codingdojo/curriculum/content/chapter/1699549659__hospitalmanager119.png)​​



**Red Belt Details**

* Root route brings users to a component that adds patients
* Features a component to display all of the patients currently admitted
* A patient details component exists that displays information about that patient
* Users can update the information about a patient
* Users can remove the patient which will remove them from the database
* Validations
  + Patient name, age, and symptoms are all required.
  + Patient names must be at least 1 character long and no more than 40 characters long.
  + Patient age can be no more than 140 and must be at least 1.
  + Patient symptoms must be at least 4 characters long.

**Black Belt and Mastery Details**

If you choose to attempt to obtain a Black Belt score, then you must satisfy all of the Red Belt base requirements and obtain mastery status in three of the five mastery competency areas listed below.

* React I - *Choose one of the following*:
  + **Lifted State**
  + **Separation of Stateful and Stateless Components**
  + **Multiple uses of Conditional Styling (x2)**
* React II - *Choose one of the following*:
  + **Single-state Object**
  + **useReducer**
  + **Custom Hook**
  + **useContext**
* MongoDB - *Choose one of the following:*
  + **Query Filters**
  + **Advanced Validations**
* MERN I - *Choose one of the following:*
  + **Axios HTTP Service**
* MERN II - *Choose one of the following:*
  + **Normalize Server-side Error Messages**
  + **Front-end Validation**

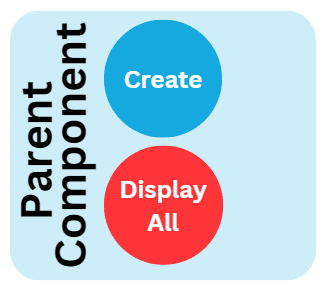
Below is a description of each requirement.

React I

*You may only choose****one****of the following:*

* **Lifted State**

Properly implement a view component that contains the component that creates a document as well as the component that displays all documents. Newly added documents should immediately display without needing a manual refresh. Attempt to match the general styling present in the Red Belt wireframe, but functionality is key here.



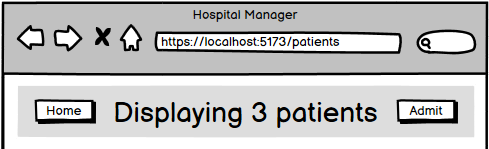
* **Separation of Stateful and Stateless Components**

Using presentational-container design principles, create a reusable Header **or**Button component that can be reused across components and other applications. The appearance should continue to resemble the Red Belt wireframe. Prop names should be generic, and the reusable component should not be stateful or contain much logic itself.

The reusable Button should work for the create and update form submission buttons as well as the delete button.

* **Multiple uses of Conditional Styling (Must do both)**

First, display the number of patients currently in the collection in the header of the component, displaying all patients.



Second, if a patient's age field value is less than 3, display a baby icon in the details component.



If a patient's age field value is greater than 75, display an icon similar to the one displayed below in the details component.



React II

*You may only choose****one****of the following:*

* **Single-state Object**

Use only one useState hook to initialize and manage all of the store fields in the create and update components.

* **useReducer**

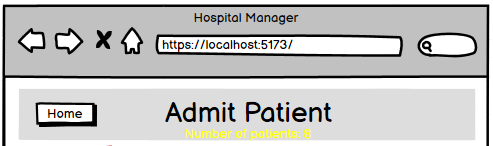
Implement the useReducer hook to handle the application's POST, PUT and/or PATCH requests. To accomplish this, there should not be more than one useReducer hook present in the application.

* **Custom Hook**

Modularize all of the front-end CRUD operations into a custom hook that can be imported into and used by the components that require it.

* **useContext**

Using the useContext hook, display the current number of patients in the header of every view from context. It should update as the number of patients changes.



MongoDB

*You may only choose****one****of the following:*

* **Query Filters**

Create a mongoose query (or queries) with the logic to filter patients depending on whether they're children (17 and younger) or adults (18 and older). Include three buttons in the component to display all patients that allow users to update the component to show children patients, adult patients, and one to reset the table to display all patients.

* **Advanced Validations**

Patients **under 18 years of age** require parental consent. Your schema will need *additional* custom validation logic to ensure they cannot be added to the collection with a "Sorry, we need a parental signature" validation message displayed. This custom validation logic should not affect the existing minimum validation logic for the age field.

MERN I

*You may only choose****one****of the following:*

* **Axios HTTP Service**

Abstract all axios request and response logic into a separate client-side service to be imported into the components where needed. The base URL (i.e. HTTP://localhost:8000/api) should only be written once in the application.

MERN II

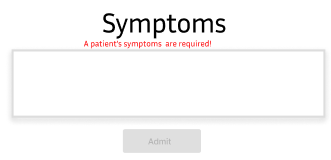
*You may only choose****one****of the following:*

* **Normalize Server-side Error Messages**

Normalize all back-end errors to contain properties for the type of error, the error’s code, the error’s message, and if it’s a validation error, it should contain multiple error messages if there’s more than one. Continue displaying validation error messages coming from the newly normalized error object.

* **Front-end Validation**

Implement front-end validations that match the schema's validation criteria. Users should not be able to submit a request to the server using the user interface alone. Messages should appear in real-time as the users type. The submit button should be disabled until there are no longer any validation errors.



**Submitting the Exam**

Before submitting the exam, remove only the node\_modules folder from the client *and*server folders. The rest of the files must be submitted in order to receive a grade. This includes every folder and file required for the project to run as intended and any extra files the course may require. Failure to include every required folder and file will result in the exam not being graded, which may result in an automatic fail for the exam attempt.