GOAL SYSTEMS

optimizing your world

Code Challenge
Frontend Engineer

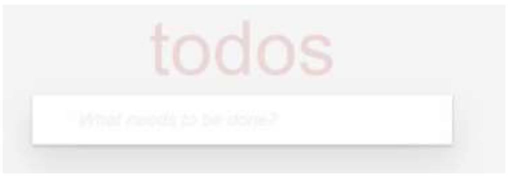
PROBLEM STATEMENT

The objective of the exercise is to display a list of tasks with two statuses: completed and incompleted. In addition, several filtering options are available.

Functional Requirements

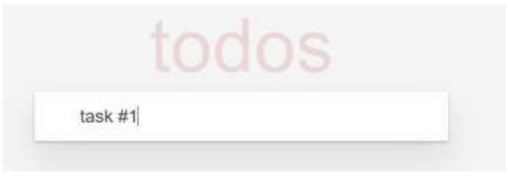
1.1 Emtpy todos

When there are no todo's, #main and #footer should be hidden.



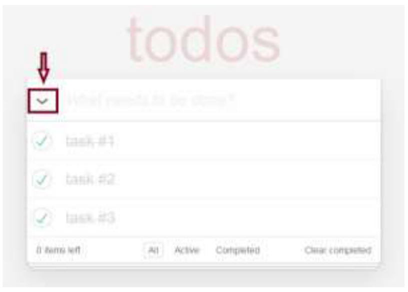
1.2 New todo

New todo's are entered in the input at the top of the app. The input element should be focused when the page is loaded. Pressing Enter creates the todo, appends it to the todo list, and clears the input. Make sure the input it's not empty before creating a new todo.



1.3 Mark all as complete

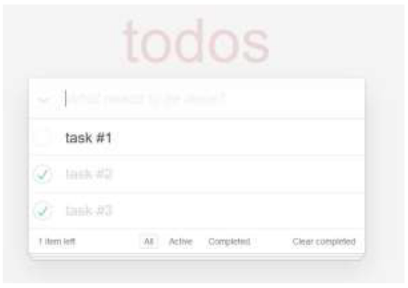
This checkbox toggles all the todos to the same state as itself. Make sure to clear the checked state after the Clear completed button is clicked. The Mark all as complete checkbox should also be updated when single todo items are checked/unchecked. Ej. When all the todos are checked it should also get checked.



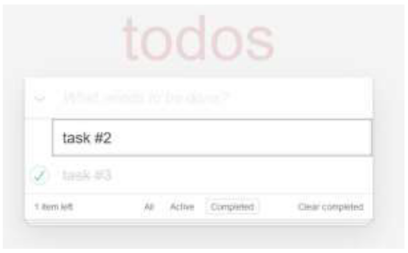
1.4 Item

A todo item has three possible interactions:

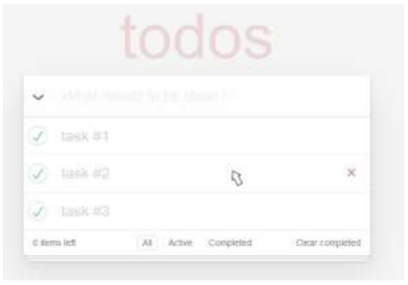
1. Clicking the checkbox marks the todo as complete by updating its completed property value .



2. Double-clicking on the todo item label activates editing mode.



3. Hovering over the todo shows the remove button.



1.5 Editing

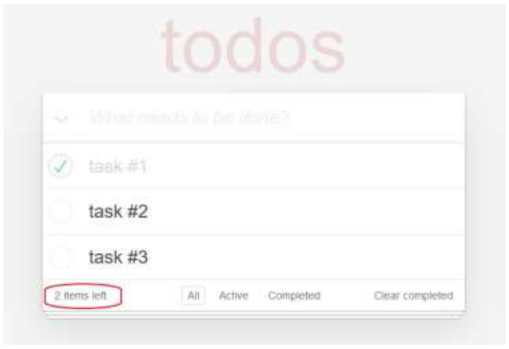
When editing mode is activated it will hide the other controls and bring forward an input that contains the todo title. The edit should be saved on: blur event and enter key press.

Check if the input is empty. If it's empty the todo should instead be destroyed. If escape is pressed during the edit, the edit state should be left and any changes be discarded.

1.6 Counter

Displays the number of active todos in a pluralized form. Make sure the number is wrapped by a `<strong>` tag. Also make sure to pluralize the `item` word correctly: `0 items`, `1 item`, `2 items`.

Example: **2** items left



1.7 Clear completed button

Removes completed todo's when clicked. Should be hidden when there are no completed todos.

1.8 State management

Your app should persist the todos state. It can be implemented using any of the state management libs: Vuex, Pinia, Redux, RxJs,...

1.9 Routing

Routing is required for all implementations. The following routes should be implemented:

```
#/(all - default)
#/active
#/completed
```

When the route changes, the ***todo*** list should be filtered on a model level and the `selected` class on the filter links should be toggled. When an item is updated while in a filtered state, it should be updated accordingly. E.g. if the filter is `Active` and the item is checked, it should be hidden. Make sure the active filter is persisted on reload.

1.10 API Rest services (Optional)

Implement CRUD services for **create**, **update** and **delete** the todo's actions above described.
Http client such as axios is allowed. It is not necessary to implement a backend for the services.
It is sufficient to mock the services using a mock server (Ej: json-server).

Evaluation Requirements

- The exercise must be performed under the SPA paradigm.
- The CSS is provided in a single file that must be split up to provide the components their corresponding styling.
- The source code must be provided with a README.txt inside with the execution instructions.

Is evaluated positively:
- The implementation with the Vue library (in case the candidate does not have experience with this framework, ReactJS or Angular are also allowed).
- Typescript integration.
- The use of native javascript ES6 functions.
- A correct hierarchy of components. Proposed hierarchy:
  - Pages (layout components)
  - Containers (regions of de UI with state management)
  - Base Components (shared)

Unit tests included.
- Recommendation: *Jest* or *Testing Library*

GOAL
SYSTEMS
optimizing your world

goal@goalsystems.com
www.goalsystems.com