

Consegna progetto finale relativo al corso

"WEB Programming and testing with a modern
framework: AngularJS"

Università degli studi di Catania in collaborazione con
BaxEnergy

Anno accademico 2016/2017

App in AngularJs per la gestione avanzata di note e task

Studente: Alessandro Pruiti Ciarello

Matricola: O55000275

Sommario

- 1 Introduzione 3
- 2 Features 3
 - 2.1 Aggiunta e rimozione nota semplice e task..... 4
 - 2.2 Ricerca 5
 - 2.3 Selezione e “Select All” 6
 - 2.4 Modifica 8
 - 2.5 Ordinamento 8
 - 2.6 Visualizzazione degli elementi, Lista e Griglia 8
- 3 Struttura 10
- 4 Considerazioni Finali..... 11

1 Introduzione

Il progetto parte da una fork della repository “<https://github.com/x00n/lesson4>”, che contiene un App per la gestione dei task sviluppata in AngularJs e rappresenta la base alla quale sono state aggiunte diverse features. Lo sviluppo ha tenuto conto di tutte le nozioni e le best practice acquisite durante il corso.

In pieno stile Single Page Application, tutte le features sviluppate sono a portata di mano e si trovano in una singola pagina, Fig 1.

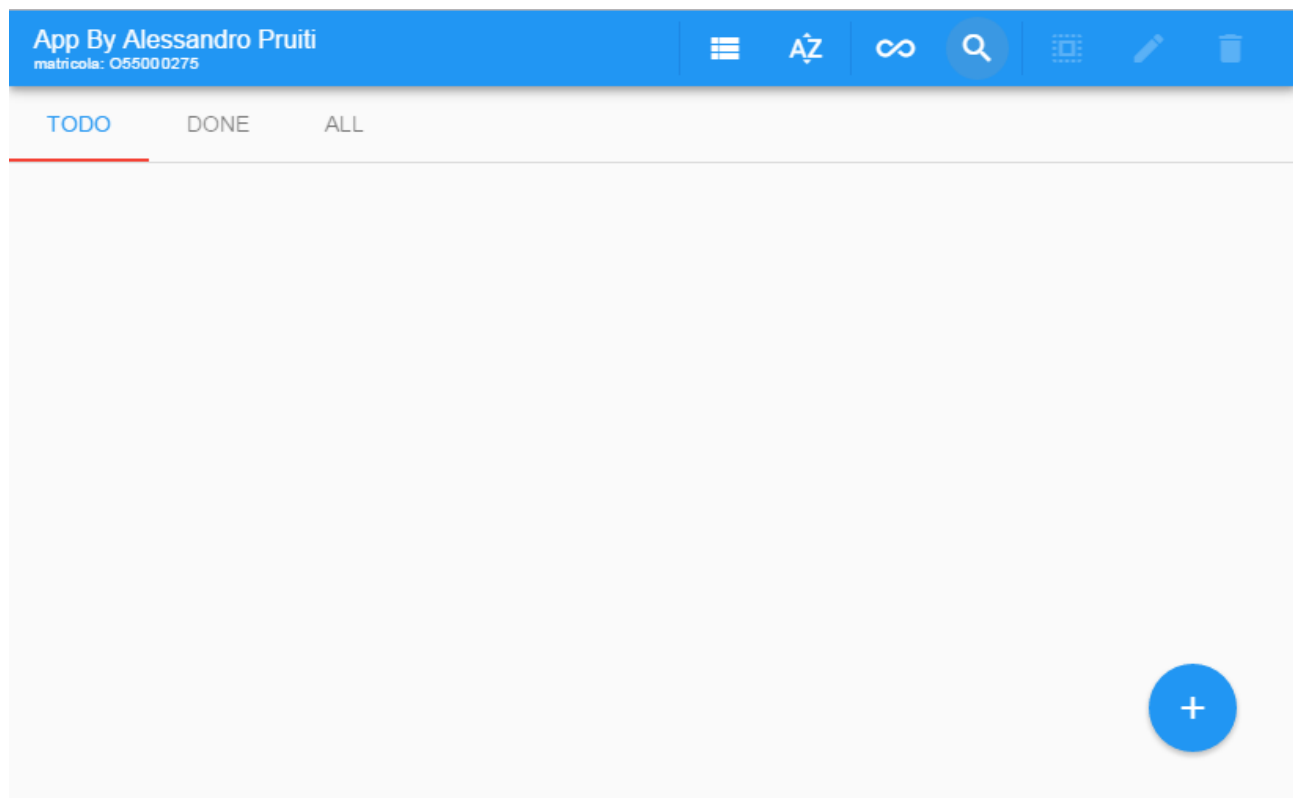


Figura 1 Vista generale App

2 Features

Alla base di partenza sono state aggiunte le seguenti funzionalità


- ✓ Aggiunta e rimozione di due diversi tipi di elementi, nota semplice e task.
- ✓ La nota semplice possiede: Titolo, testo, priorità, data, Todo/Done flag.
- ✓ Il task possiede: Titolo, descrizione, data, ora, durata, priorità, Todo/Done flag.
- ✓ Funzione di ricerca, per titolo o per tutti gli elementi presenti nell'elemento.

- ✓ Selezione singola e multipla degli elementi visualizzati.
- ✓ 'Seleziona tutto' per la selezione o deselection di tutti gli elementi visualizzati
- ✓ Modifica di un singolo elemento selezionato
- ✓ Ordinamento degli elementi visualizzati per testo o per data
- ✓ Possibilità di cambiare visualizzazione degli elementi da lista a griglia

Tutte le funzionalità elencate sono accessibili tramite l'unica toolbar presente nella pagina, basta cliccare sull'icona per attivare la funzionalità o passare da una modalità ad un'altra, che sia di ricerca ordinamento o di visualizzazione degli elementi.

Visivamente le icone sono divise in sezioni, quelle che riguardano la vista, la ricerca e l'interazione con l'elemento.

2.1 Aggiunta e rimozione nota semplice e task.

Tramite l'apposito tasto  si accede alla form di inserimento di un nuovo task o di una nota semplice, dove sarà possibile, nel caso di nota semplice, specificarne titolo, priorità ed un testo (figura 2) e nel caso di task sarà inoltre possibile selezionare una data, un'ora, e la durata del task (figura 3). In caso di nota semplice, verrà memorizzata la data e l'ora del momento di creazione della stessa.

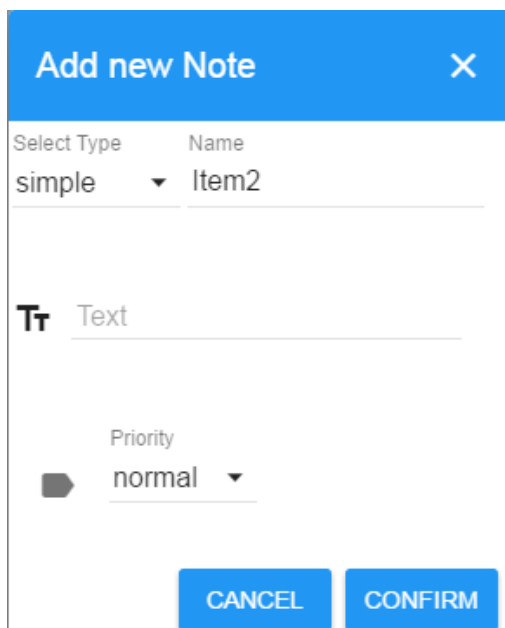


Figura 2 Form inserimento nota semplice

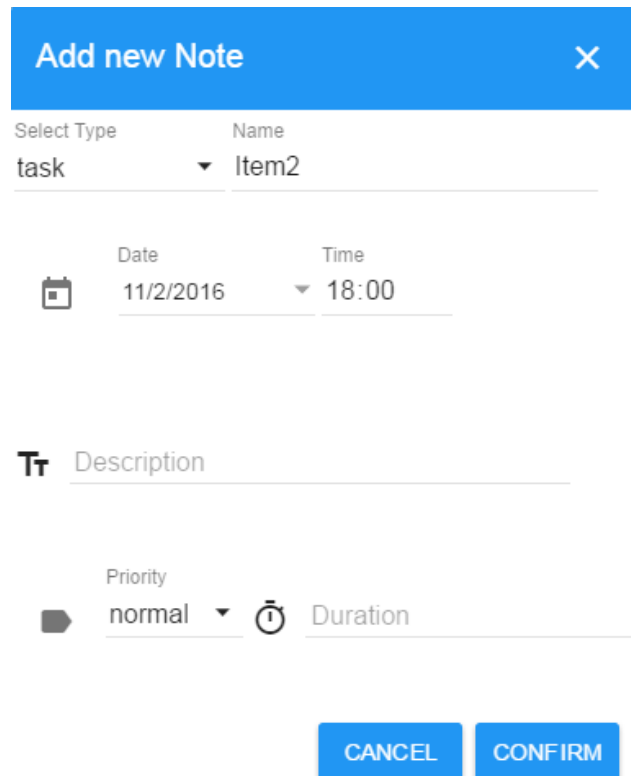



Figura 3 Form inserimento task


La form di inserimento viene richiamata dalla funzione addTask() e viene creata tramite il servizio \$mdDialog, come visibile nel codice che segue. Nel caso di inserimento verrà passato al form anche un elemento vuoto con un titolo già preimpostato che indica il numero del task che si sta per inserire.

```
1. //Add or modify task to the items list
2. function addTask(ev,isAdd,obj) {
3.     var obj1 = angular.copy(obj);
4.     //custom form//
5.     var myFormDialog = {
6.         locals: {item: obj1,formType: isAdd},
7.         bindToController: true,
8.         controller: 'TodoController',
9.         controllerAs: 'ctrl',
10.        templateUrl: 'app/components/addForm.tmpl.html',
11.        parent: angular.element(document.body),
12.        targetEvent: ev,
13.        clickOutsideToClose:false
14.    };
15.    if (isAdd){
16.        $mdDialog.show(myFormDialog).then(function(result) {
17.            vm.createItem(result);
18.        });
19.    }else{
20.        $mdDialog.show(myFormDialog).then(function(result) {
21.            vm.editItem(result);
22.        });
23.    }
24. };
```



Per avere una maggiore riusabilità del codice ho preferito utilizzare il medesimo template sia per l'inserimento che per la modifica del task, nonché la medesima funzione per la creazione del form, la distinzione dei due avviene tramite il parametro isAdd passato come argomento delle funzione addTask().

La rimozione di un oggetto, accessibile con l'apposito tasto , può essere singola o multipla, ciò dipende dal numero di elementi selezionati. Semplicemente alla pressione e alla conferma vengono eliminati dalla lista gli elementi selezionati, una volta implementata la selezione multipla la funzione non ha richiesto grandi modifiche al codice base.

2.2 Ricerca

Una volta inserite le proprie note ed i propri task è possibili filtrarli tramite una funzione di ricerca alla quale si accede tramite una casella di input a scomparsa attivabile tramite l'icona .

Dinamicamente, man mano che il testo viene digitato, gli elementi verranno filtrati. A tal proposito si possono selezionare due tipi di ricerca, per titolo o completa.

A testo inserito si può passare da una modalità di ricerca ad un'altra semplicemente cliccando sull'icona alla sinistra dell'input di ricerca. ( o ).

La ricerca sfrutta il filtro “filter” già presente in AngularJs.

```
1. | filter: customListCtrl.filterSearch()
```

L'implementazione è stata pensata per essere facilmente scalabile e poter aggiungere ricerche su diversi campi dell'elemento. Vi è una funzione che gestisce i toggle delle modalità di ricerca, alla quale ad ognuna di esse viene assegnato un intero. L'intero verrà poi passato alla direttiva che si occupa della visualizzazione degli elementi che grazie alla funzione `filterSearch()` restituisce al filtro il campo dove effettuare la ricerca; sarà quindi possibile aggiungere la ricerca su un nuovo campo semplicemente passando il suo id alla direttiva ed inserirlo nella funzione di ritorno `filterSearch()`, al momento implementata con un `if` e non con un `switch` considerando che sono state implementate solo due ricerche.

```
1. function filterSearch(){
2.     if (vm.search== null) return "";
3.     else if(vm.searchIn== 0) return vm.search;
4.     else return {title: vm.search};
5. }
```

2.3 Selezione e “Select All”

Gli elementi visualizzati possono essere selezionati semplicemente cliccando su di essi ed un click successivo implica la deselectazione dello stesso. Questa funzionalità è gestita dalla funzione `toggleSelection()`.

```
1. function toggleSelection(item) {
2.     var index= vm.selectedItem.selected.indexOf(item);
3.     if (index == -1){
4.         vm.selectedItem.selected.push(item);
5.     }
6.     else
7.         vm.selectedItem.selected.splice(index, 1);
8.
9. }
```

Per implementare la selezione multipla è stato creato un vettore che contenga dinamicamente gli oggetti selezionati e, tale vettore, verrà utilizzato inoltre all'interno della funzione `backgroundColor()` per restituire alla direttiva “ng-style” il colore da dare all'elemento selezionato o deselectato.

```
1. ng-style="{background: customListCtrl.backgroundColor($index,item)}"
```

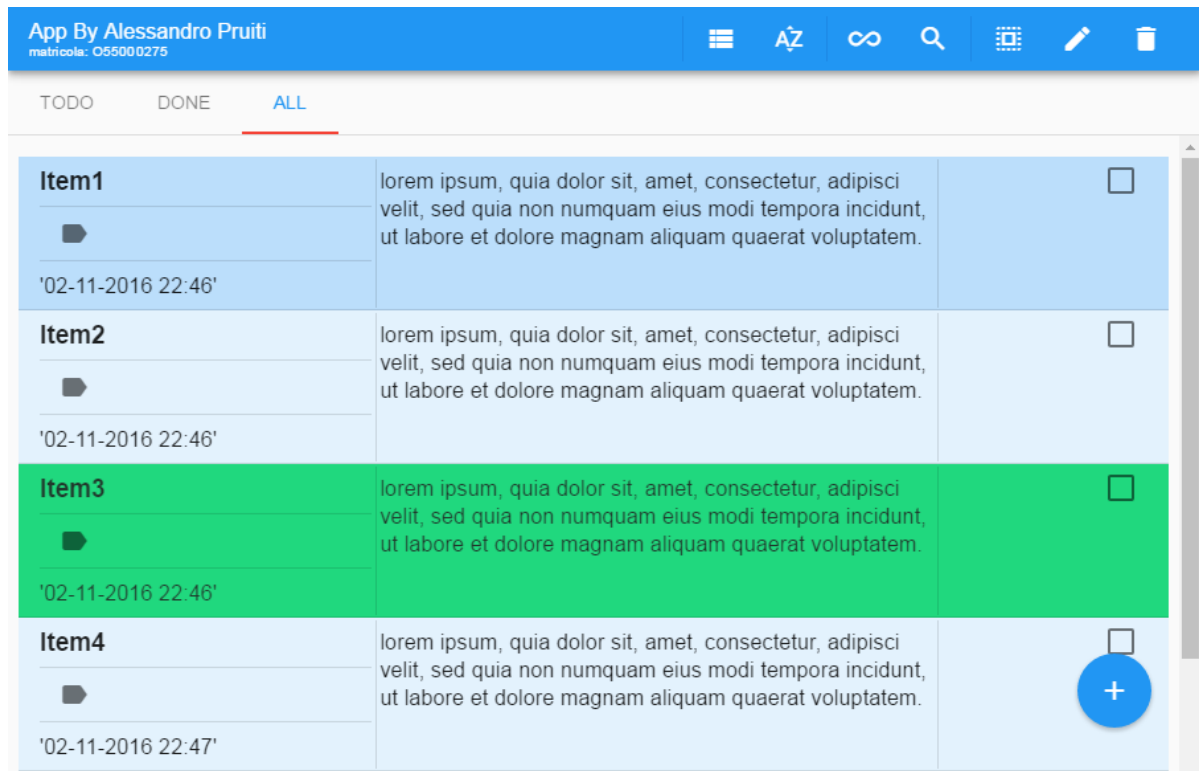



Figura 4 Schermata con un task selezionato (in verde)

Per evitare che al cambio di un Tab l'oggetto rimanga selezionato senza essere però visualizzato, durante la transazione il vettore viene svuotato e in tal modo, se si prova a cancellare un elemento o più di uno, non si rischia di eliminare elementi non visualizzati come selezionati e non presenti in lista.

Oltre alla selezione singola e multipla è stata aggiunta una funzionalità di “Select All”, accessibile tramite l'icona , per poter in modo più rapido gestire la selezione e deselezione di tutti gli elementi. Questa funzionalità ha richiesto la necessità di assegnare ad ogni Tab un proprio vettore dove si andranno a memorizzare tutti gli elementi filtrati e visualizzati in quel momento; inoltre si è dovuto tenere traccia dell'attuale Tab visualizzato dall'utente tramite la variabile “currentTab”, che varia in conseguenza della selezione di un Tab, per poter poi copiare il vettore attualmente visualizzato all'interno del vettore degli elementi selezionati. Tale operazione viene eseguita dalla funzione selectAll();


```

1. function selectAll(){
2.     if(vm.selectedItem.selected.length == vm.selectedItem.currentTab[vm.currentTab
3.         ].length)
4.         vm.selectedItem.selected = [];
5.     else
6.         vm.selectedItem.selected = vm.selectedItem.currentTab[vm.currentTab];

```



Così facendo la selezione va ad evidenziare gli elementi effettivamente visualizzati dall'utente e non tutti gli elementi presenti nello storage.

2.4 Modifica

Nel momento in cui viene selezionato un singolo elemento verrà abilitata l'icona  che ci permette di andare a modificare tutti i campi dell'elemento selezionato.

Come già accennato nell'inserimento di un nuovo task per la modifica, è stato riutilizzato lo stesso template dell'inserimento passando alla funzione del nuovo task non solo l'oggetto attualmente selezionato ma anche un parametro "isAdd" settato a false; quest'ultimo sta ad indicare che si tratta di una modifica e non della creazione di un elemento. L'oggetto passato al form sarà comunque una copia dell'elemento selezionato e non l'elemento stesso, questo per evitare che l'utente, dopo aver modificato alcuni campi, decida di annullare la procedura e quindi di rendere inefficaci le modifiche.



2.5 Ordinamento

Funzionalità accessibile tramite le icone  e  è l'ordinamento rispettivamente per titolo e per data.

Analogamente a quanto fatto per la ricerca, ad ogni tipologia di ordinamento viene assegnato un intero, quindi l'ordinamento attualmente selezionato viene poi passato alla direttiva che, in modo del tutto analogo alla ricerca, richiama una funzione che sfruttando tale intero ritornerà al filtro 'SortBy', già insito in AngularJs, l'elemento su cui basare l'ordinamento.

2.6 Visualizzazione degli elementi, Lista e Griglia

Di default tutti gli elementi inseriti dall'utente vengono visualizzati in una lista avente un elemento su di ogni riga.

Tramite le icone  e  è possibile passare dalla visualizzazione a lista a quella a griglia e viceversa.

La visualizzazione degli elementi viene eseguita da parte di una direttiva personalizzata, già citata precedentemente nella ricerca e nell'ordinamento. Tale direttiva richiede poi come attributi diversi parametri, come appunto il tipo di ricerca e di ordinamento, ma anche gli elementi stessi da visualizzare o ancora il Tab corrente in cui ci troviamo per poter gestire al meglio la selezione.

Avendo implementato due viste differenti era possibile immaginare la presenza di due direttive differenti, una per ogni vista. Essendo la logica di funzionamento del controller per la gestione degli elementi, nonché per il loro filtraggio, praticamente identica per entrambe le viste, ho pensato di usare la medesima direttiva e poter quindi riutilizzare al meglio il codice già scritto, passando come parametro il template relativo alla vista selezionata.

```
1.      <custom-list  template=ctrl.customTemplate[ctrl.viewType]
2.                      items="ctrl.items"
3.                      selected-item="ctrl.selectedItem"
4.                      filter-function="ctrl.notDone"
5.                      search="ctrl.searchInput"
6.                      tab="0"
7.                      sort-by =ctrl.sortBy
8.                      search-in = ctrl.searchIn >
9.      </custom-list>
```

Un po' come avveniva per la ricerca e l'ordinamento, al toggle dell'icona della vista verrà associato un intero che mi permette di selezionare all'interno di un vettore di template quello corrispondente e di passarlo alla direttiva. Tale template verrà poi richiamato grazie alla direttiva di AngularJs ng-include.

```
1.  function directive() {
2.      return {
3.          scope: {
4.              template: '=',
5.          },
6.          bindToController: {
7.              items: '=',
8.              selectedItem: '=',
9.              filterFunction: '=',
10.             search: '=',
11.             tab: '=',
12.             sortBy: '=',
13.             searchIn: '=',
14.             template: '=',
15.          },
16.          controller: customListController,
17.          controllerAs: 'customListCtrl',
18.          transclude: true,
19.          restrict: 'E',
20.          template: '<ng-include src="customListCtrl.template" ></ng-include>',
21.      };
22.  }
```

Così facendo è possibile aggiungere velocemente un nuovo template all'applicazione ed estendere le possibili viste.

NB: la vista a griglie presenta un layout responsive che adatta il numero di colonne dei tiles in base alla risoluzione di visualizzazione dell'App. Ho impostato a mio

piacimento tre fasce di risoluzione, passando dalla visualizzazione di 1 colonna per la risoluzione minima, di 3 per quella intermedia ed infine 5 colonne per la massima risoluzione.

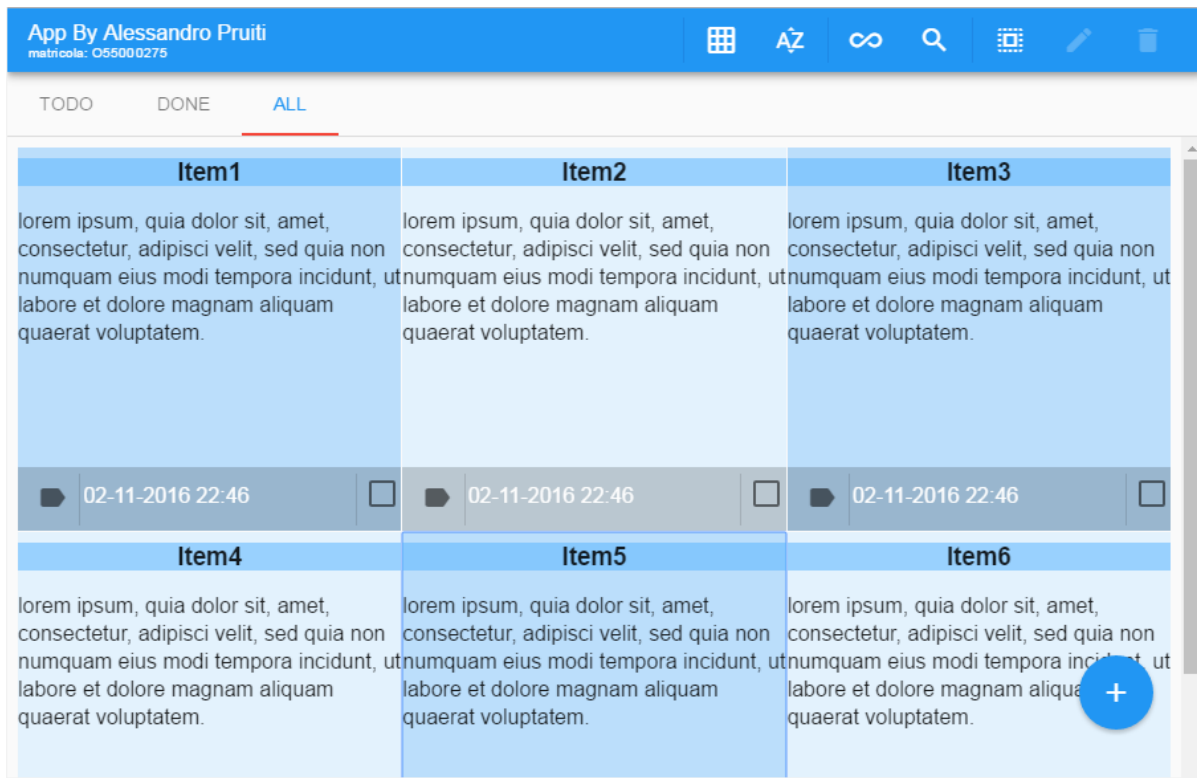


Figura 5 visualizzazione a griglia (3 colonne)

3 Struttura

Essendo l'applicazione semplice, avendo seguito il pattern Single Page Application ed avendo fatto riuso del codice per diverse funzionalità dell'applicazione, la struttura si presenta poco articolata (Figura 4). Avendo inoltre rispettato le best practices per le quali sarebbe consono non possedere file con più di 400 righe di codice e cartelle con non più di 7 file, non vi è stata la necessità di aggiungere ulteriori controller o direttive che avrebbero, più che alleggerito il codice, complicato la struttura dell'app.

Vi sono tre cartelle principali: una contenente i file di stile e le icone, un'altra contenente le api di AngularJs e infine la cartella App che contiene tutta la logica dell'applicazione.

All'interno della cartella App vi è il modulo principale (todoApp.module.js) ed il controller principale ad esso associato (todo.controller.js). Troviamo poi una cartella contenente tutti i componenti di supporto al controller principale, quali il servizio per la gestione dello storage locale offerto dal browser (storage.service.js), la direttiva già

brevemente citata per la gestione della visualizzazione degli elementi e infine i template relativi alle due viste implementate e al form di inserimento e modifica.

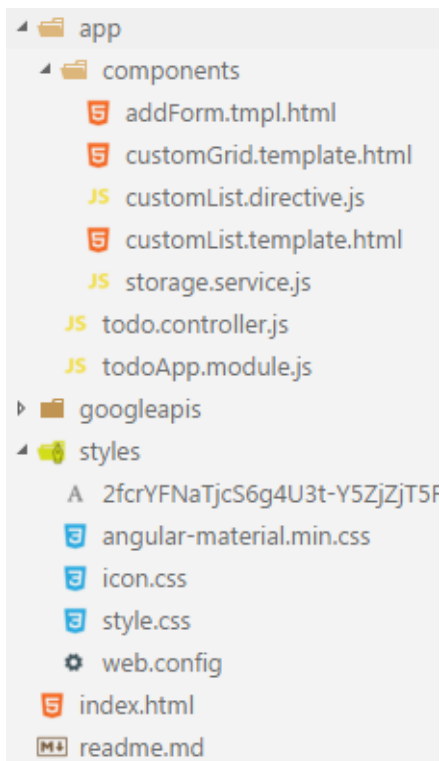


Figura 6 struttura dell'App

4 Considerazioni Finali

Per tutta la durata dello sviluppo, come già accennato, si è cercato di tener conto di tutte le best practices apprese e di rendere il codice leggibile e poco complesso. La logica ha tenuto conto di una possibile espansione e ha reso un eventuale aggiunta di funzionalità semplice e veloce senza aggiungere però ulteriore complessità al progetto. È stato tenuto conto nella vista a griglie, ma anche in altri template utilizzando l'attributo flex, della flessibilità nella visualizzazione dell'App. Lo stile è stato implementato anche a scopo didattico sia tramite foglio di stile css sia utilizzando direttive AngularJs come ng-class ed ng-style o ancora direttamente come attributo style nei tag HTML.

È possibile visionare una demo del progetto all'indirizzo:

<http://pru.altervista.org/index.html>