Laboratorio 3
Curso: Introducción al Desarrollo de Páginas Web – II Semestre 2024 – Grupo 50
Profesor: Adalberto Cubillo Rojas

Indicaciones

En este laboratorio vamos a desarrollar un sitio web para llevar el control de tareas por hacer (o una lista de "To-Do"). Contará con almacenamiento local de los datos a través de un archivo JSON y cada tarea podría contar con un archivo adjunto (más adelante se menciona cómo funcionaría este apartado).

El sitio web debe desarrollarse utilizando principalmente componentes de React (pero también puede utilizarse en conjunto con HTML, CSS y JavaScript). Con respecto al aspecto visual de la interfaz web queda a criterio del estudiante darle el formato a través de CSS y de la forma que desee mientras cumpla con los requerimientos que se enlistan en las secciones más adelante (además se adjuntan unos bosquejos de interfaz que sí se debe cumplir con la organización de los componentes en el sitio web). Para la interacción con los archivos locales (el JSON con los datos y los adjuntos de las tareas) se espera que se utilice Fetch API.

Parte 1 – Estructura general del proyecto

El directorio y archivos del proyecto deben estar estructurado de la siguiente forma (algunos de estos elementos ya son generados automáticamente cuando se inicializa el proyecto en React):

- **/laboratorio3**
  - /node_modules
  - /public
  - **/src**
    - **/components**
    - **App.js**
    - **App.css**
    - **index.js**
    - **...**
  - **/data**
    - **data.json**
  - **/assets**
    - **attachment-example1.pdf**
    - **attachment-example2.jpg**
    - **...**
  - package-lock.json
  - package.json
  - README.md

Dentro del folder **“src”** deben ir todos los archivos JS con los distintos componentes generados utilizando JSX y React. Así como cualquier archivo CSS para darle formato visual al sitio web. Se espera que haya un archivo por cada componente dentro del folder **“components”**, o al menos varios componentes relacionados entre si en el mismo archivo (además de emplear **import/export** para llamar los distintos componentes en otros archivos según sea necesario).

Dentro del folder **“data”** debe incluirse el archivo JSON que almacenará los datos del sitio web (más adelante se menciona el formato interno del archivo).

Dentro del folder **“assets”** deben incluirse los archivos adjuntos a cada tarea del “To-Do”. Cada archivo debe tener un nombre único (puede ser basado en un UUID o alguna combinación de fecha/id de cada tarea).

Para la entrega del laboratorio 3 debe subirse al TEC Digital un archivo comprimido **ZIP/RAR** con el contenido completo desarrollado en el folder **“laboratorio3”**. **No se permite entregar un link a un Google Drive o gestor de archivos similar**.

Parte 2 – Sitio Web

1. El sitio web contará con: **una barra de navegación sencilla** (título y botón de agregar tareas), una **barra lateral izquierda con la lista de “To-Do”** (título, cantidad total de tareas, campo de busqueda de tareas y un listado de tareas), **un formulario flotante para agregar tareas**, y **un contenedor al lado derecho para visualizar los detalles** de una tarea seleccionada (ver bosquejos adjuntos).

2. A continuación, se muestra el bosquejo (mockup) inicial del sitio web y los distintos componentes principales que se esperan visualizar y generar a través de React (podrían existir más que estos, pero estos son los mínimos requeridos para cumplir con el laboratorio).

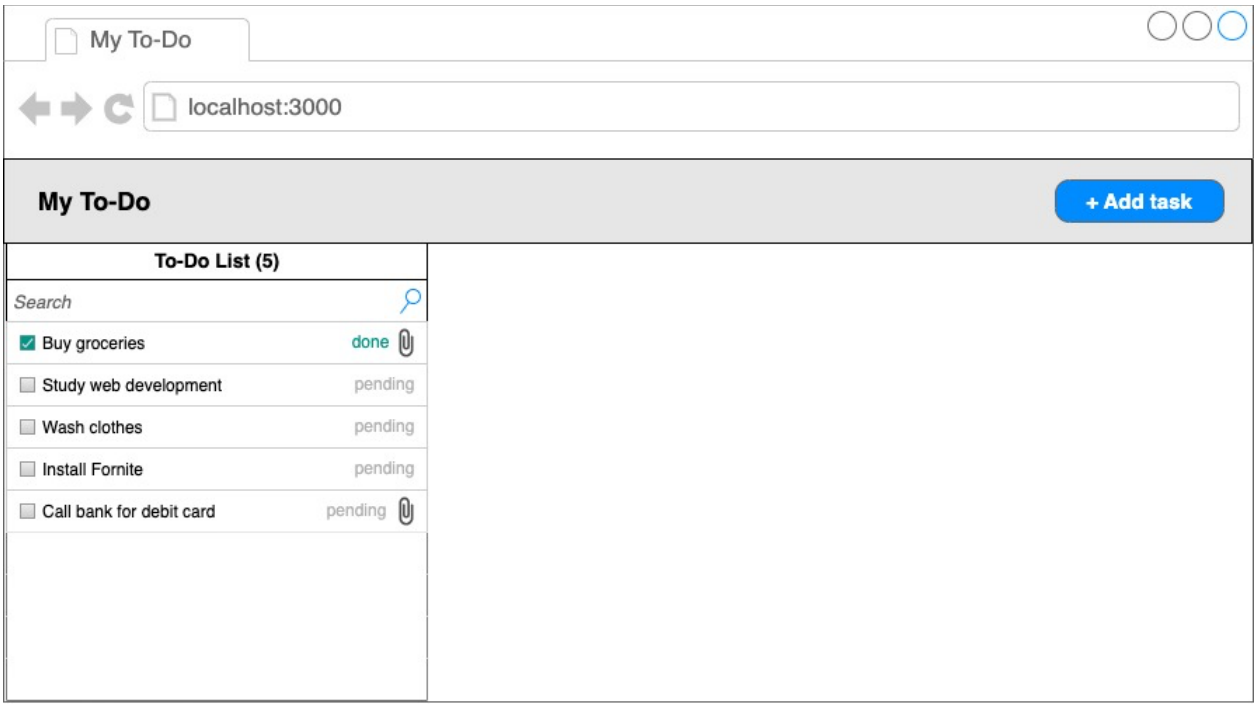

Figura 1. Mockup de la interfaz inicial.

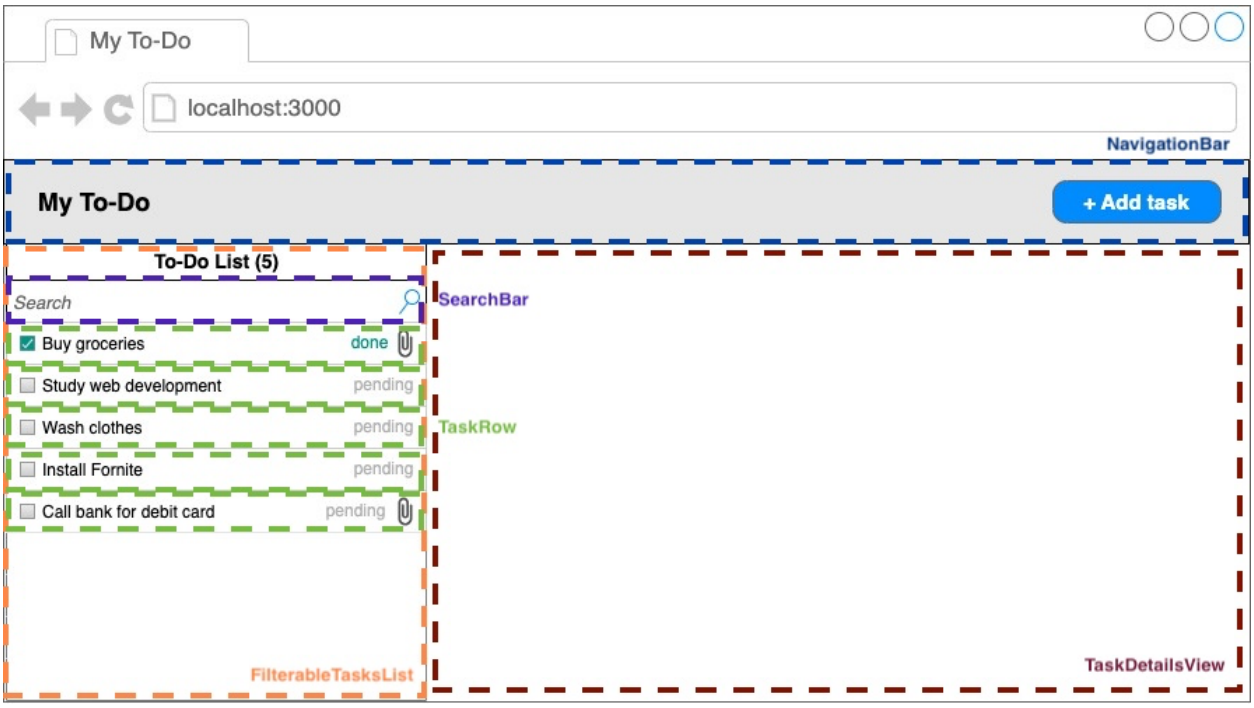

Figura 2. Mockup de la interfaz inicial con los componentes generales resaltados.

3. **<u>En el componente “NavigationBar”</u>**, se mostrará el título del sitio web al lado izquierdo y un botón para agregar nuevas tareas al lado derecho.

a. Una vez **presionado el botón de agregar nuevas tareas**, se mostrará un contenedor flotante con un formulario para poder incluir la tarea a la lista de "To-Do" (ver bosquejos adjuntos).

4. **En el componente "FilterableTasksList"**, se mostrará al lado izquierdo del sitio web una lista con las tareas agregadas al "To-Do".
    a. Contará con un título "To-Do List (X)" **donde X representa la cantidad total de elementos en la lista**. Una vez se agreguen o eliminen elementos de la lista este número debe cambiar acordemente (**esto se tiene que hacer a través de Hooks/States de React**).
    b. Además, **contará con un campo para poder filtrar los elementos de la lista** según el título de las tareas. Una vez ingresado texto en este campo la lista debe actualizarse solo con las tareas que el título contenga ese valor (**esto se tiene que hacer preferiblemente a través de Props/Hooks de React**).
    c. Finalmente, se muestra una lista de componentes de tipo **"TaskRow"** para cada una de las tareas en la lista de "To-Do".

5. **El componente "TaskRow"** se comportará de la siguiente manera:
    a. Contará con **un checkbox para poder cambiar el estado de una tarea** entre completada o no.
    b. A su vez tendrá **un título** para poder identificar cada tarea.
    c. Tendrá **un label de estado (pendiente/completado)** para poder diferenciar entre tareas listas o no.
    d. Finalmente, tendrá **un ícono en caso de que la tarea contenga algún archivo adjunto**.

6. **Al presionar un componente de "TaskRow"** se mostrará en **el componente "TaskDetailsView"** los detalles de la tarea de la siguiente forma:
    a. **Un título** con el nombre de la tarea.
    b. Un label **al lado derecho con la fecha de creación** de la tarea.
    c. **Una descripción** de la tarea.
    d. Un enlace para **poder mostrar en otra ventana del navegador el archivo adjunto** de cada tarea (en caso de tener alguno).
    e. **Una imagen para mostrar una vista previa del archivo adjunto** en caso de que este sea de tipo JPG/PNG (de no contar con un archivo de este tipo debe ocultarse).
    f. **Un botón para poder eliminar la tarea**.

7. A continuación, se muestran los bosquejos de los 3 tipos de detalles que se pueden observar en el componente de "TaskDetailsView" según contenga un archivo adjunto, no contenga, o contenga una imagen.

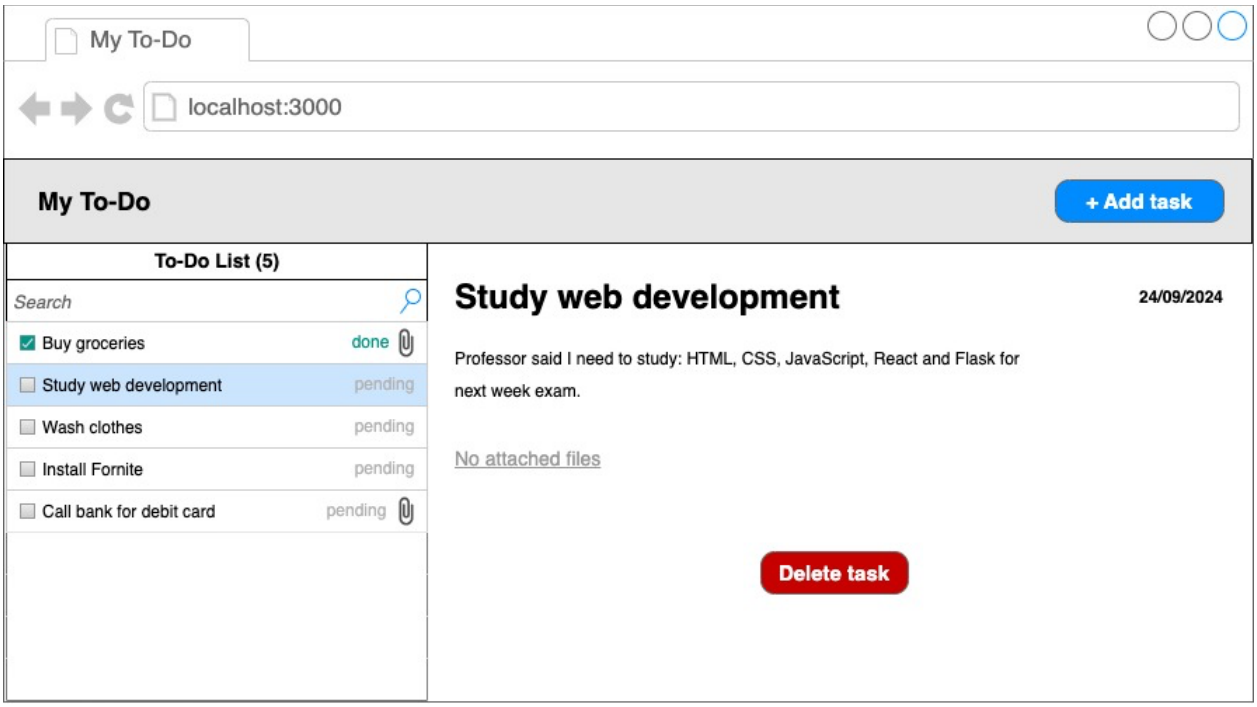

Figura 3. Mockup de la selección de una tarea sin archivos adjuntos.

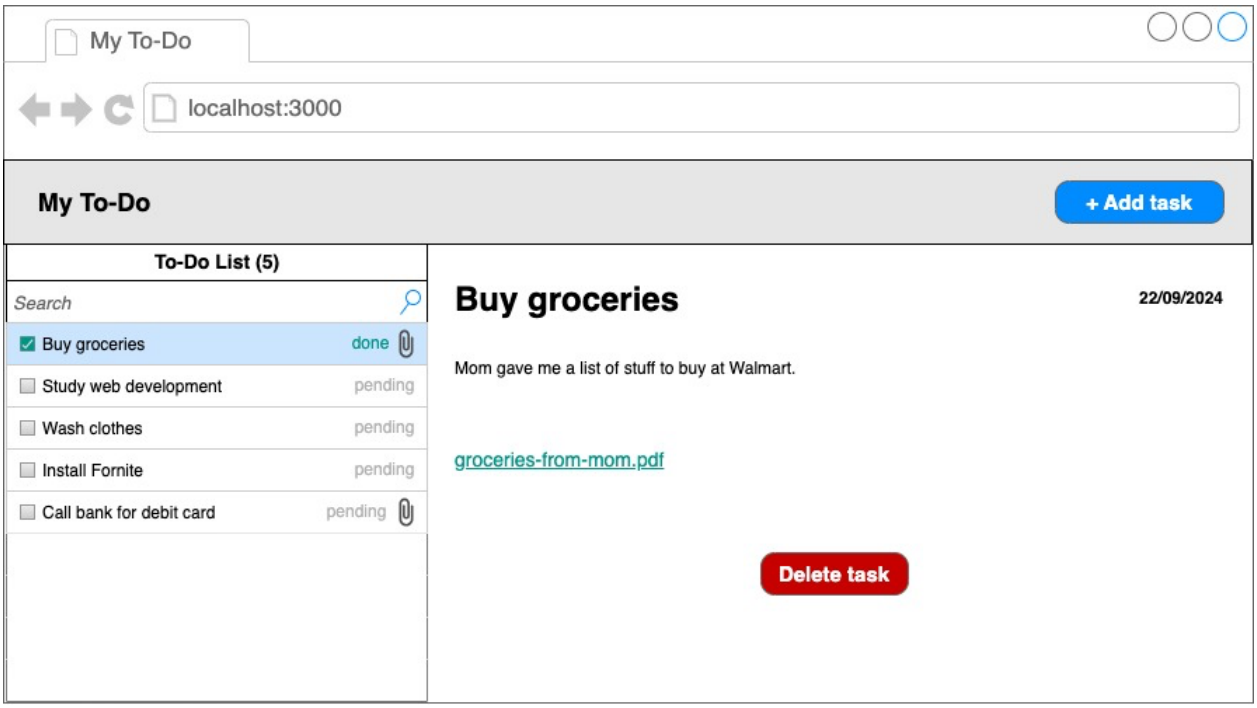

Figura 4. Mockup de la selección de una tarea con un archivo adjunto (no imagen).

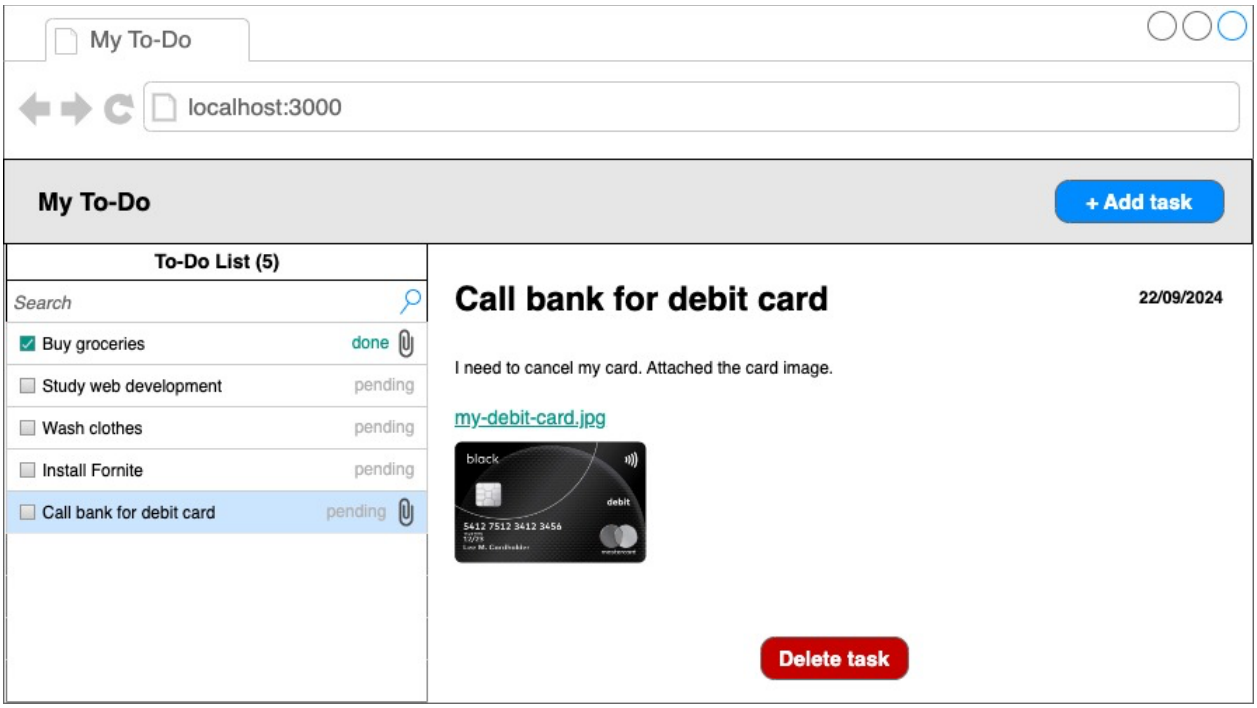

Figura 5. Mockup de la selección de una tarea con un archivo adjunto (imagen).

8. Para el formulario flotante, a la hora de agregar tareas a la lista de "To-Do" se utilizarán 2 componentes generales (pueden existir otros adicionales según la necesidad): "AddTaskBackground" y "AddTaskForm".
    a. **El componente "AddTaskBackground"**, se utilizará para mostrar un contenedor de color negro con opacidad al 40% que cubrirá la totalidad del tamaño de la ventana del sitio web en el navegador y que contendrá centrado al componente "AddTaskForm".
    b. **El componente "AddTaskForm"**, contendrá los siguientes elementos (o pueden ser componentes):
        i. Un título "Add task to To-Do list".
        ii. **Un campo de texto para agregar el título de la tarea** (obligatorio).
        iii. **Un campo de texto para agregar la descripción o detalles** de la tarea (opcional).
        iv. **Un input de tipo "file" para poder adjuntar un documento o imagen a la tarea** (opcional, tamaño máximo 5 MB).
        v. 2 botones, **uno para cancelar el proceso de agregar una tarea** (oculta el formulario y vuelve a la vista inicial del sitio web), **y otro para agregar la tarea a la lista** (agrega los datos y archivos y refresca la lista del "To-Do").

9. A continuación, se muestran los bosquejos del formulario y los distintos componentes principales que se esperan visualizar y generar a través de React (podrían existir más que estos, pero estos son los mínimos requeridos para cumplir con el laboratorio).

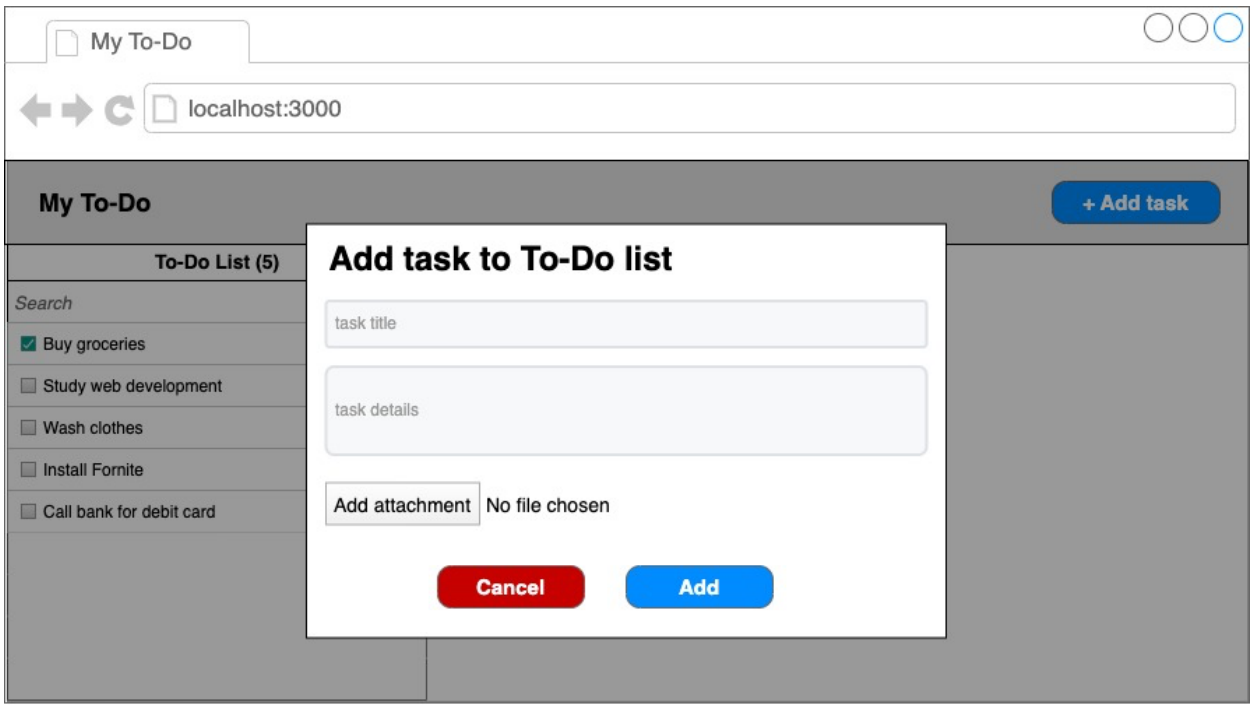

Figura 6. Mockup del formulario para agregar nuevas tareas al “To-do”.

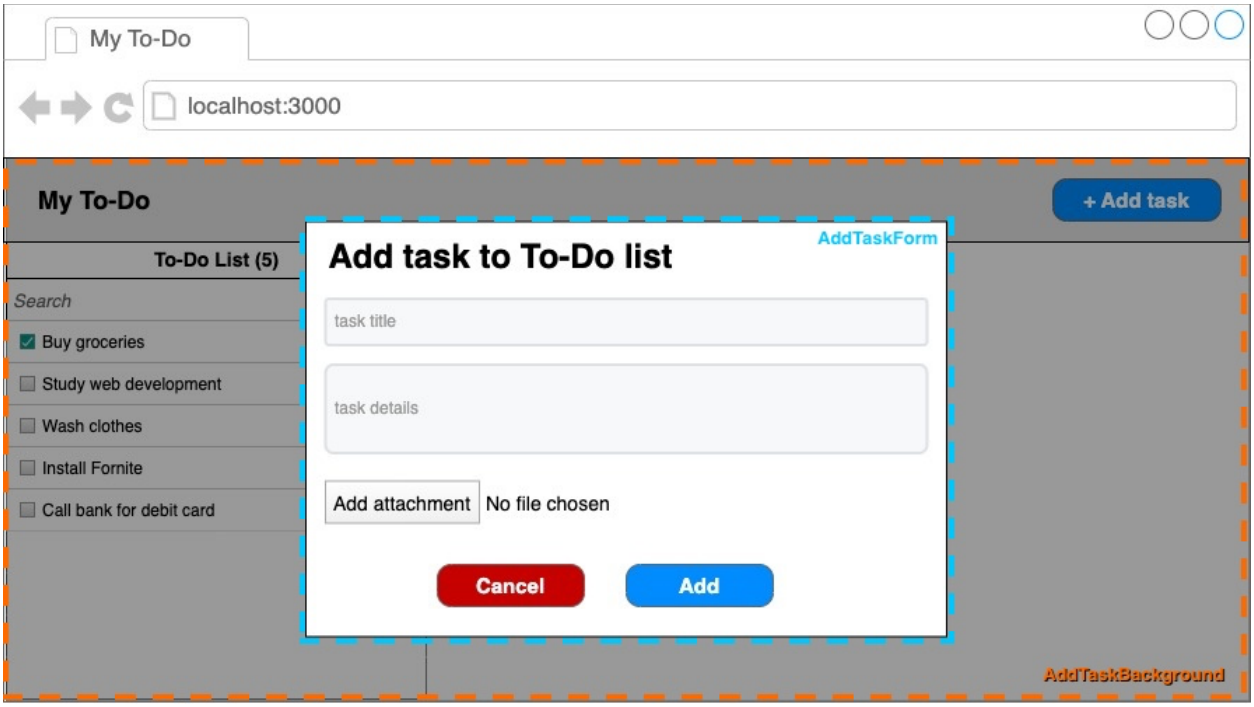

Figura 7. Mockup del formulario para agregar nuevas tareas con los componentes generales resaltados.

Parte 3 – Manejo de los datos y archivos adjuntos

- Como se mencionó en la Parte 1, para el manejo de los datos del sitio web se utilizará un archivo JSON local para almacenar la lista de tareas del "To-Do". Este archivo tendrá el siguiente formato.


```
[
  {
    "id":0,
    "title":"This is a task",
    "created_at":"24/09/2024",
    "completed":false,
    "details":"This task has some details over here.",
    "attachment_url":null
  },
  {
    "id":1,
    "title":"This is another task",
    "created_at":"22/09/2024",
    "completed":true,
    "details":"This task has some details over here.",
    "attachment_url":"../assets/attachment_example.pdf"
  },
  {
    "id":2,
    "title":"Again another task",
    "created_at":"20/09/2024",
    "completed":false,
    "details":null,
    "attachment_url":"../assets/attachment_example2.jpg"
  }
]
```


Figura 8. Ejemplo del formato para el archivo JSON con los datos del sitio web.

- **Se espera que el contenido se cargue en el sitio web al cargar la primera vez en un objeto de JavaScript. Y luego, cada vez que se agreguen/actualicen/eliminen tareas se escriba la totalidad de la lista de vuelta al archivo JSON.**
- Los tipos de datos que se observan en la figura 8 es una recomendación, pero se podría cambiar según la necesidad (p. ej. "created_at" podría ser un UNIX epoch representando un datetime – No lo recomiendo, más fácil con string).
- **Para el key "attachment_url"**, la idea es guardar el valor de la ubicación de los archivos adjuntos. El valor en la figura 8 es representativo, preferiblemente debería almacenarse un URL que pueda utilizar React para acceder al archivo desde el navegador de forma correcta (e.g. "http://localhost:3000/assets/..." o similar).

- Toda interacción con el archivo de los datos y los archivos adjuntos debe hacerse a través de Fetch API (excepto el enlace que carga otra ventana del navegador para mostrar el adjunto).
- **No hace falta eliminar archivos adjuntos huérfanos** (aquellos que quedaron almacenados en el sitio web, pero que la tarea fue eliminada de la lista).