

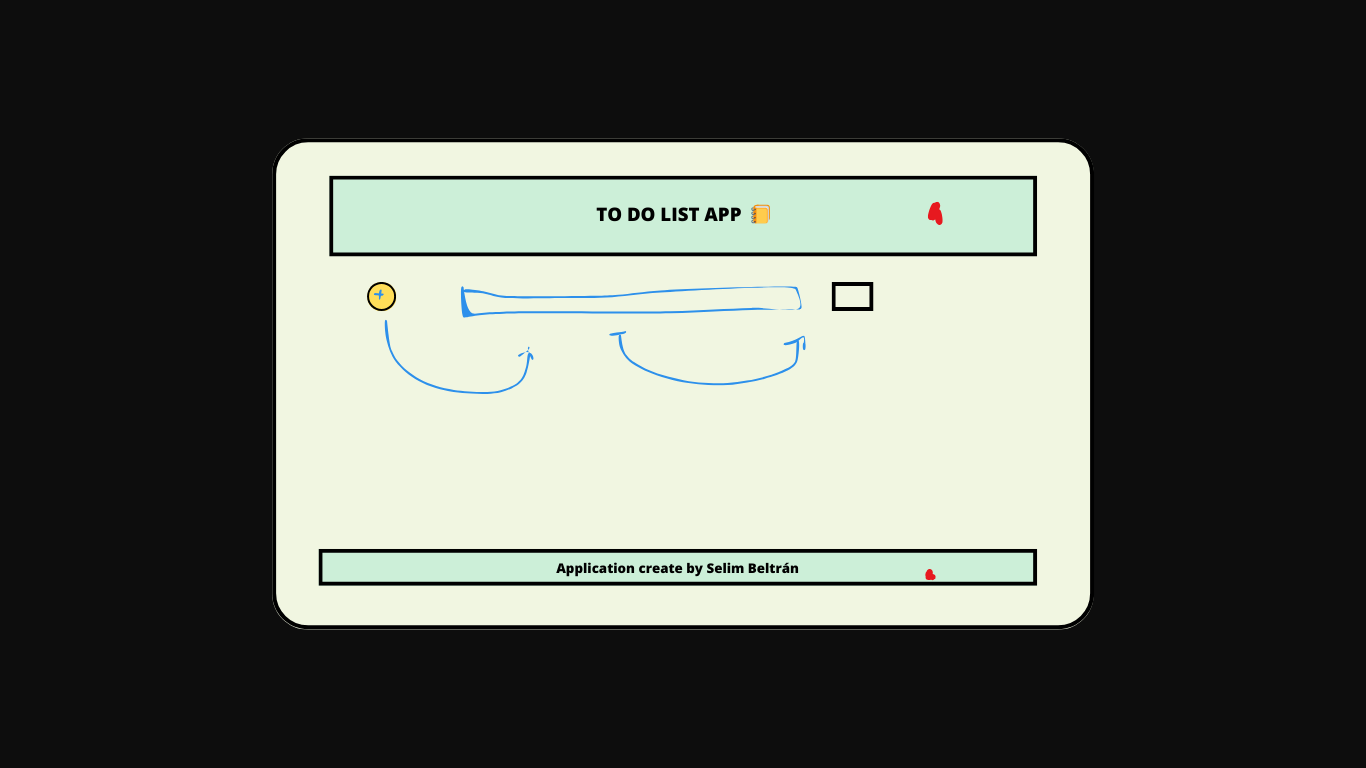
**Project To-Do App**

**Introducción:**

Se creará este programa con un fin educativo, por el cual se buscar aprender a crear una aplicación web usando HTML, CSS y JavaScript. Se realizo un mockup por el cual el proyecto debe regirse y en la documentación se mencionará que hará cada función de la aplicación.

**Diseños:**

Diseño base de la aplicación web. -





En la siguiente imagen se buscará mantener lo que está en rojo como base, es decir, lo que debe estar estático, lo de amarillo debe ser lo dinámico y su función será:

* Añadir la tarea añadiendo la tarea con un checklist button alado suyo.
  + Este debe estar centrado justamente en la mitad de rectángulo.

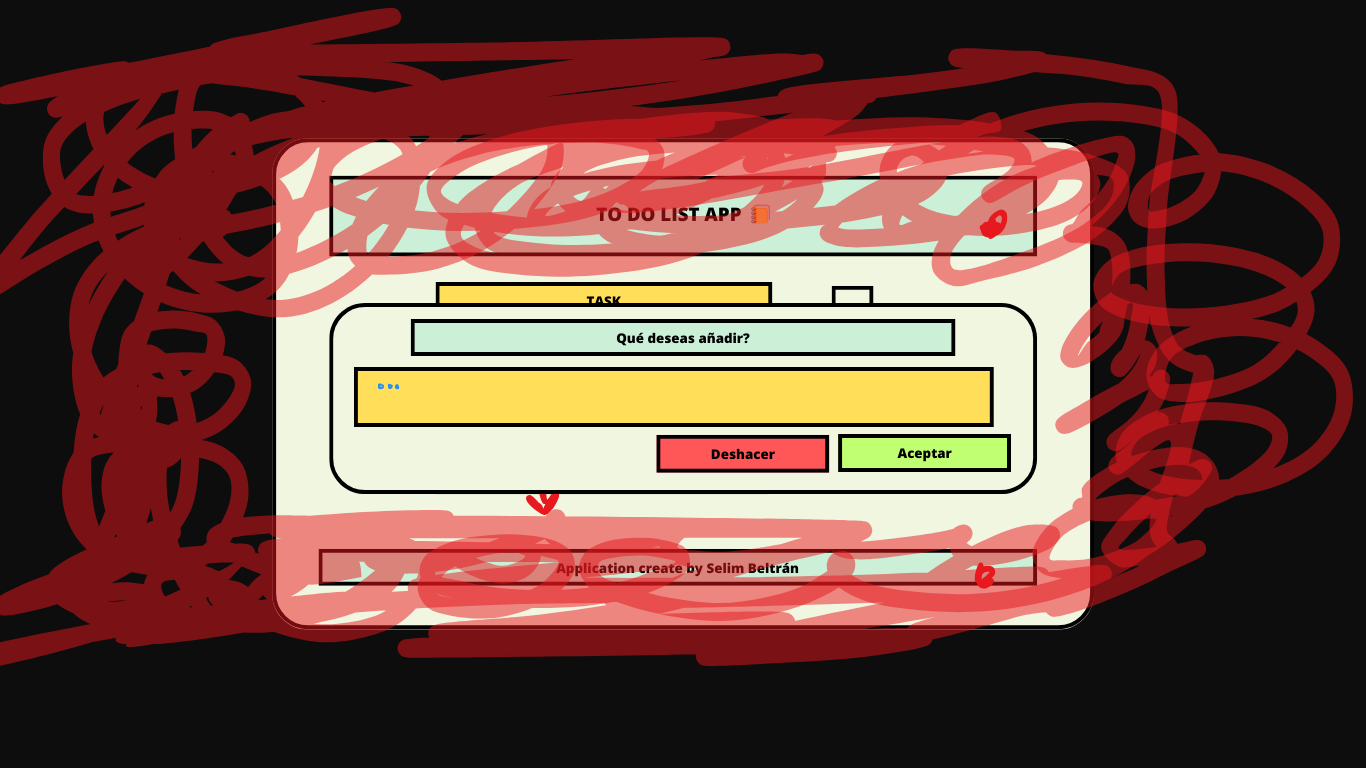
Se recuerda que al momento de agregar tantas tareas no se saturara el programa o habrá límite, sino que simplemente bajara él eslogan.

Como último paso se recuerda que el marco debe estar a la mitad de la página, así como los rectángulos que lo componen.

Texto. -

Al momento de presionar le botón de añadir tarea deberá aparecer una pantalla emergente pregunta que deseas escribir de tarea.

En el cual se buscará opacar todo lo de atrás para que solo se enfoque la tarea.



Funciones de cada botón:

Deshacer: se cerrará la pantalla y no pasará nada.

Aceptar: se añadirá la tarea y seguirá el flujo.

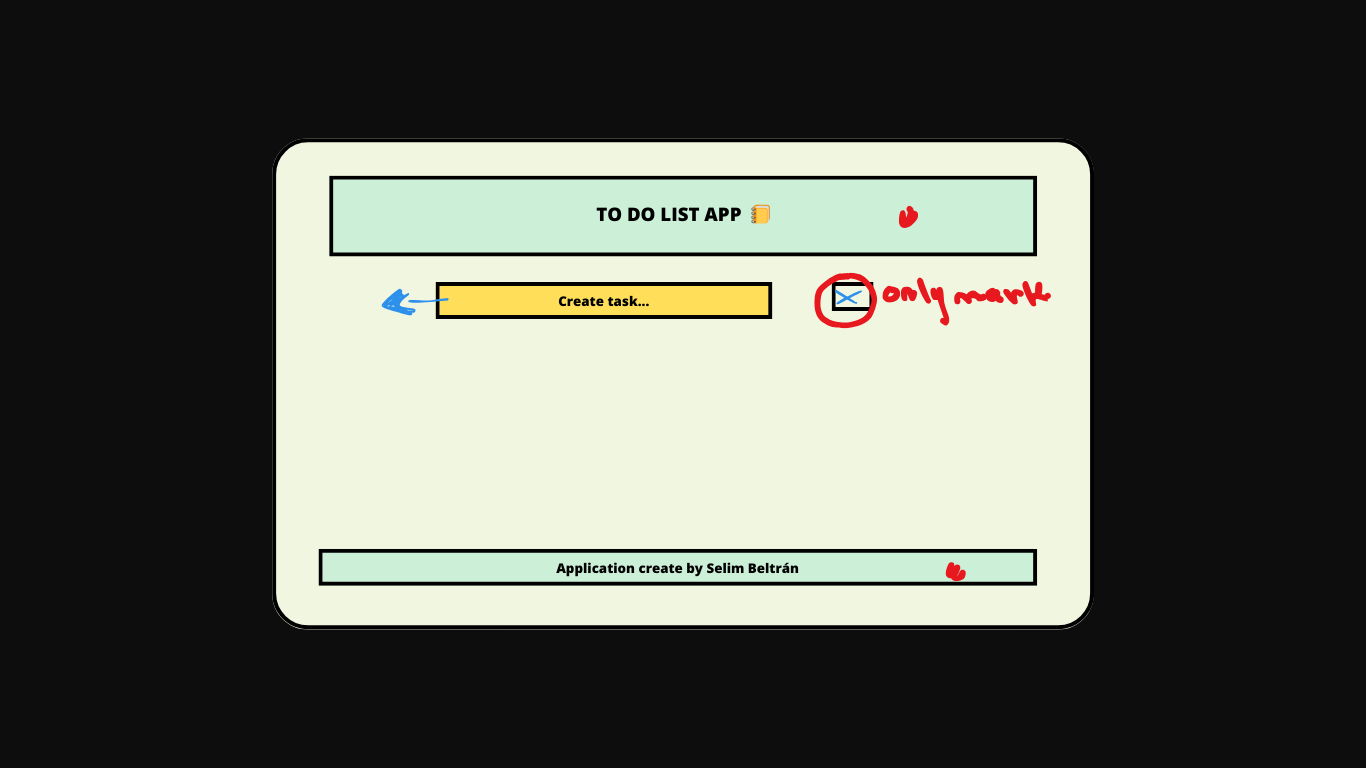
Cuadrado de texto: deberá aparecer un cuadrado en donde aparecen 3 puntos de referencia y se tendrá que hacer click para empezar a escribir.

Texto estático: ¿Qué deseas añadir?

Terminología Opaco: Todo lo de atrás y al momento de 0 o 1 desaparece.

Texto largo: Se hará la pestaña para abajo.

**Imagen #2:**



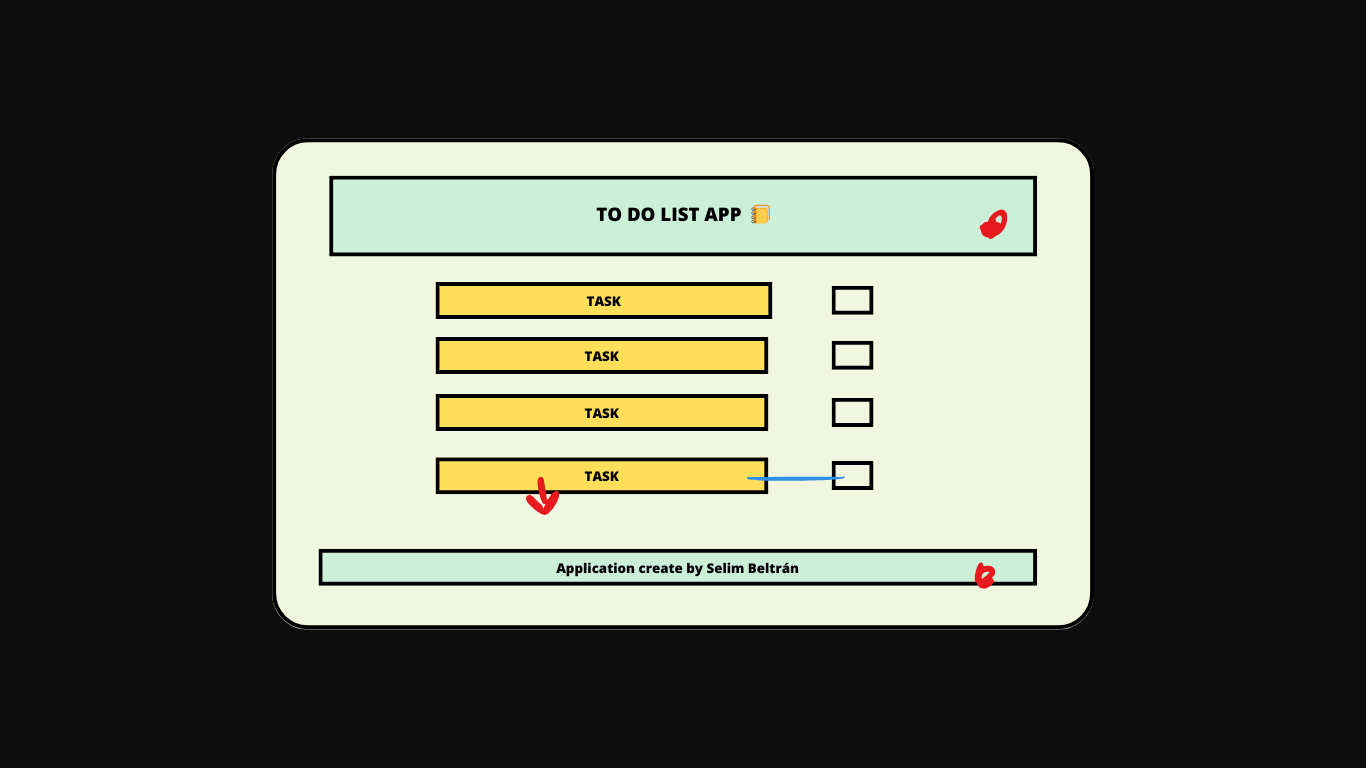


A continuación, se muestra como deberá ser el resultado del botón, aquí se ve el check button, en el cual deberá poder marcarse como una x azul y su función será que al momento de ser marcada deberá subrayar toda la tarea.

Existirá a su medida otro botón a la par que Check que eliminara la tarea, su función será devolver todas las tareas al mismo orden en el que estaban para que no se queden flotando.

Además, se recordare que la tarea en teoría debe ser larga, por ende, la tarea al momento de expandirse, deberá irse para atrás. NO ES EDITABLE LA TAREA.

**Resultado Final:**





Este será el resultado final del proyecto, la flecha para abajo solo demuestra que la tarea deberá ir acorde al eslogan para abajo.

Restricciones:

Al momento de reiniciar la página deberá seguir todo igual. Subrayado y tarea.

Día #1:

Etiquetas base del proyecto en la primera etapa:

* Encabezado tipo 1: ‘TO-DO LIST 📒’ ID: title
* Encabezado tipo 4: ‘Application créate by Selim Beltrán’ ID: credits
* Link de tipo de letra deseada.

Estilo CSS:

Implementación de tipografía:

* Al principio del documento se ha de vincular todo a la tipografía mencionada.

Rectángulo #1:

* Background de color tipo: #0D0D0D
* Se debe implementar con el Z índex: 0 para que así los demás pueda resaltar.
* Position: null.

Rectángulo #2:

* Background color: #F1F6E1
* Debe estar alineada en el centro de la página.
* Se debe implementar con el Z índex: 10 para que así los demás pueda resaltar.
* Tamaño:
  + weidth: 65%
  + height: 380px
  + margin: auto
* Borde: #000000
* ID: rectangleTwo
* Position: relative.

Rectángulo #3:

* Background color: #CCEFD8
* Centrado al rectángulo #2; Ubicación: Parte superior no tan apegado al tope, punto estático.
* Se debe implementar con el Z índex: 20 para que así los demás pueda resaltar.
* Borde: #000000
* ID: rectangleThree
* Position: Static
* Display: flex;
* Align-items: center;
* Justify-content: center;
* Contenido:
  + Debe estar alineado al centro del Rectángulo #3
  + Id: textOne

Rectángulo #4:

* Background color: #CCEFD8
* Centrado al rectángulo #2; Ubicación: Parte inferior no tan apegado al borde, punto estático.
* Se debe implementar con el Z índex: 30 para que así los demás pueda resaltar.
* Borde: #000000
* ID: rectangleFour
* Position: static.
* Display: flex;
* Align-items: center;
* Justify-content: center;
* Contenido:
  + Es un ‘Application créate by Selim Beltrán
  + Debe estar alineado al centro del Rectángulo #4

Círculo Figura #5:

* Background color: #FFDE59
* Borde: #000000
* ID: circle one
* Position: Static
* Display: flex;
* Justify-content: center;
* Align-items: center;
* Se debe implementar con el Z índex: 40 para que así los demás pueda resaltar.
* Centrado en la 2/7 parte del rectángulo #2, que se encuentre estático
* Contenido:
  + Botón con un ‘’+’’. Id=’’buttonCreateTask’’
  + Debe estar en el centro del círculo.

Día #2:

Logro: Se implemento la tipografía deseada al proyecto.

Se implemento el rectángulo #1 y #2 de la posición correctamente.

Día #3:

Day Off

Día #4:

Se implementaron los textos del rectángulo #3 y #4, pero se encontró una falla en su margen, por ende, tendrá que resolverse.

Consultar en: [Diagrama ULM](https://lucid.app/lucidchart/d393cff2-5943-4416-a89c-768d38572151/edit?invitationId=inv_a3294325-1ac3-404a-a863-748402d5855c&page=0_0)

Día #5:

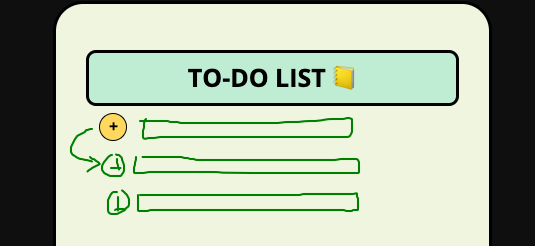
Se encontró una solución temporánea, pero efectiva que parece permanente.

El error consistió en el mal manejo de display debido a que se encontró que las propiedades aplican directamente a la figura y que su tamaño proviene del widht and height.

Hay que denotar el uso de display y su función en alinear textos.

Se denoto muy buenos resultados, pero se encontraron problemas grandes en el proceso, por ejemplo:

* En la creación del icono de agregar más, este se tiene que poner alado de la nueva tarea sub agregada, es decir:



Pero con las propiedades de CSS este proceso se puede complicar a medida que vaya creciendo ya que traerá consigo más problemas que resultado.

Eso es todo el avance del día número 5.

Día #6:

Se encontró una solución al problema en la que consiste en los siguiente:

Llegamos al supuesto que el botón siempre aparecerá alado de una tarea no creada, por lo tanto, el botón de arriba desaparece y continuará el legado del anterior botón.

Pero con esta solución surge otro problema, que consiste en que no se conoce cuando de largo podría ser la tarea y a qué medida se deberá alargar la posición del botón.

Proponemos una solución más intuitiva que consiste en limitar los caracteres de la tarea, mostrando así seriedad con las tareas, ya que una lista de marcaje no debe ser tan larga con su objetivo.

Example:

X ‘Llamar a mamá porque se me quedaron las llaves de la casa y tengo que decirle para que me las traiga’

√ ‘Llamar a mamá’

Procederos a calcular en teoría como avanzara la tarea.

ACCIÓN IMPORTANTE

Se hará un cambio en el mockup por términos de ego y de diseño mal propuesto, a continuación, se mostrará el mismo proyecto, pero con un cambio ligeramente diferente, es IMPORTANTE recalcar que no se hará un cambio completo al diseño sino un cambio al momento de agregar la tarea.

Colores:

Morado débil: #BA99FA

Rojo débil: #F18383

Verde débil: #C1FF72

Amarrillo: #FFDE59

Verde frágil: #D3E8B7

Verde medio: #C1FF72

Imagen número #1



El diseño de la interfaz cambio a un color cálido y amigable el cual busca mejorar el aspecto de la misma. Los aspectos a tomar en cuenta serán los siguientes:

Rectángulo #1:

* Id: cuerpo
* Background de color tipo: #0D0D0D
* Centrar al medio de la página web.

Rectángulo #2:

* Background color: #F1F6E1
* ID: contenedor
* Debe estar alineada en el centro de la página.
* Se debe implementar con el Z índex: 10 para que así los demás pueda resaltar.
* Borde: #000000
* Position: relative.
* Debe contender a los siguientes rectángulos que se formaran.

Rectángulo #3:

* ID: titulo
* Background color: #BA99FA
* Centrado al rectángulo #2; Ubicación: Parte superior no tan apegado al tope, punto estático.
* Se debe implementar con el Z índex: 20 para que así los demás pueda resaltar.
* Borde: #000000
* Contenido:
  + Id: titulo-texto
  + Debe estar alineado al centro del Rectángulo #3

Rectángulo #4:

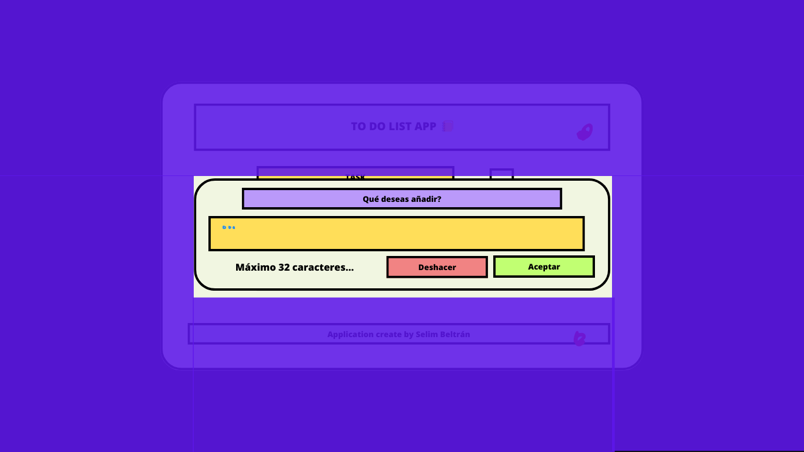
* ID: eslogan
* Background color: #BA99FA
* Centrado al rectángulo #2; Ubicación: Parte inferior no tan apegado al borde, punto estático.
* Se debe implementar con el Z índex: 30 para que así los demás pueda resaltar.
* Borde: #000000
* Contenido:
  + Id: frase
  + “Keep calm and carry on”
  + Debe estar alineado al centro del Rectángulo #4

Rectángulo #5:

* Id: creador-tareas
* Background: #C1FF72
* Su forma es de un rectángulo en el cual se aplicará un **radio de borde** parecido a la imagen mostrada.
* Debe estar centrada en el medio de la **frase**, es decir, a su parte superior.
* Debe ser un botón que accione la funcionalidad de crear una tarea para el usuario, por ende, debe de ser un botón sin marco ni borde.
* Contenido del botón:
  + Id: boton-creador-tareas

Imagen número #2

Día #7:



Se ha concluido con la imagen número #1 en la cual se ha implementado la estructura de cada rectángulo inclusive con el botón para crear tareas, y en la segunda plantilla se trabajará con la ayuda de este botón…

El botón creador de tareas será un activador de pestaña emergente la cual buscará crear una tarea solicitada por el usuario.

Si no ha quedado claro, al presionar el botón este hará saltar una ventana emergente en la cual se podrá crear la tarea del usuario.

Se debe tener en cuenta varios aspectos al momento de crear este tipo de ventanas, ya que debe estar alineada con las reglas y limitaciones de la respectiva imagen.

A continuación, se procederá a detallar cada aspecto importante y de restricción de la siguiente fase del proyecto:

1. El botón llamara a la ventana emergente.
2. La ventana emergente saldrá y opacará toda la visión de atrás.
3. El usuario escribirá la tarea.
4. Aceptará y aparecerá la tarea en el marco/ **v** /Abjurará y esta quedará impune.

A groso modo, estas serían las actividades que se tendrían en cuenta a realizar al momento de la UI/UX.

Se continuará a detalle cada aspecto de la imagen:

Paso #1:

* Vincular el botón con **JS** para que pueda surgir la ventana emergente, la cual no debe interrumpir ningún proceso.
* Esta ventana emergente necesita estar centrada con sus debidas dimensiones, particularidades y elementos.
* **--/**Se cree un posible caso que esta opacara al 100% el ‘’acto principal’’, por ende, si se da dicha acción esta se pondrá en una posición de tal manera que se vea un poco el ‘’acto principal’’, pero de manera opacada. **/--**

¿Cómo se tiene pensado que esta emerja?

* Se buscará aumentar el **Z-índex** de cada figura para que esta sobresalga de las demás, y cada uno contara con su respectivo espacio, pero no hace falta obviar que la figura principal deberá contar con una posición diferente de estática.
* Con respecto a la opacidad del fondo esta se hará al final de esta fase del proyecto ya que no supone una intervención directa al proceso principal; En pocas palabras es el proceso secundario.

**Una etiqueta que debe tener importancia en esta fase del proyecto será: formulario, ya que, en pocas palabras, lo que vamos a crear es un formulario;**

**Id:** formulario-contenedor--tarea

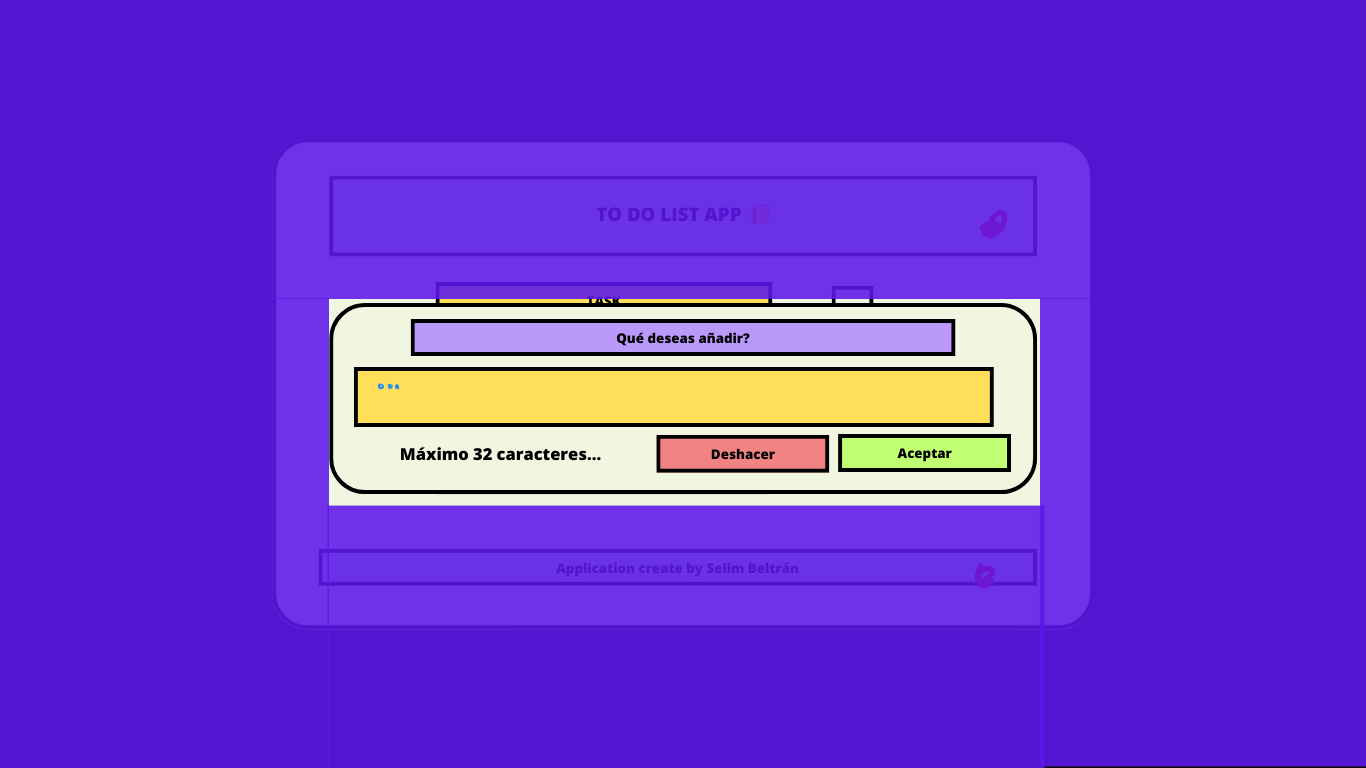
Una vez dadas las expectativas del proyecto, se procederá a mencionar a detalle el proceso que deberá seguir cada elemento y como deber ser involucrado cada proceso de la susodicha fase:

**Proceso #1**



* Id: boton-creador-tareas
* Proceso de **JS** involucrado, el cual se buscará saltar una ventana emergente.
* Se invocará todo el formulario contenedor.

**Proceso #2**





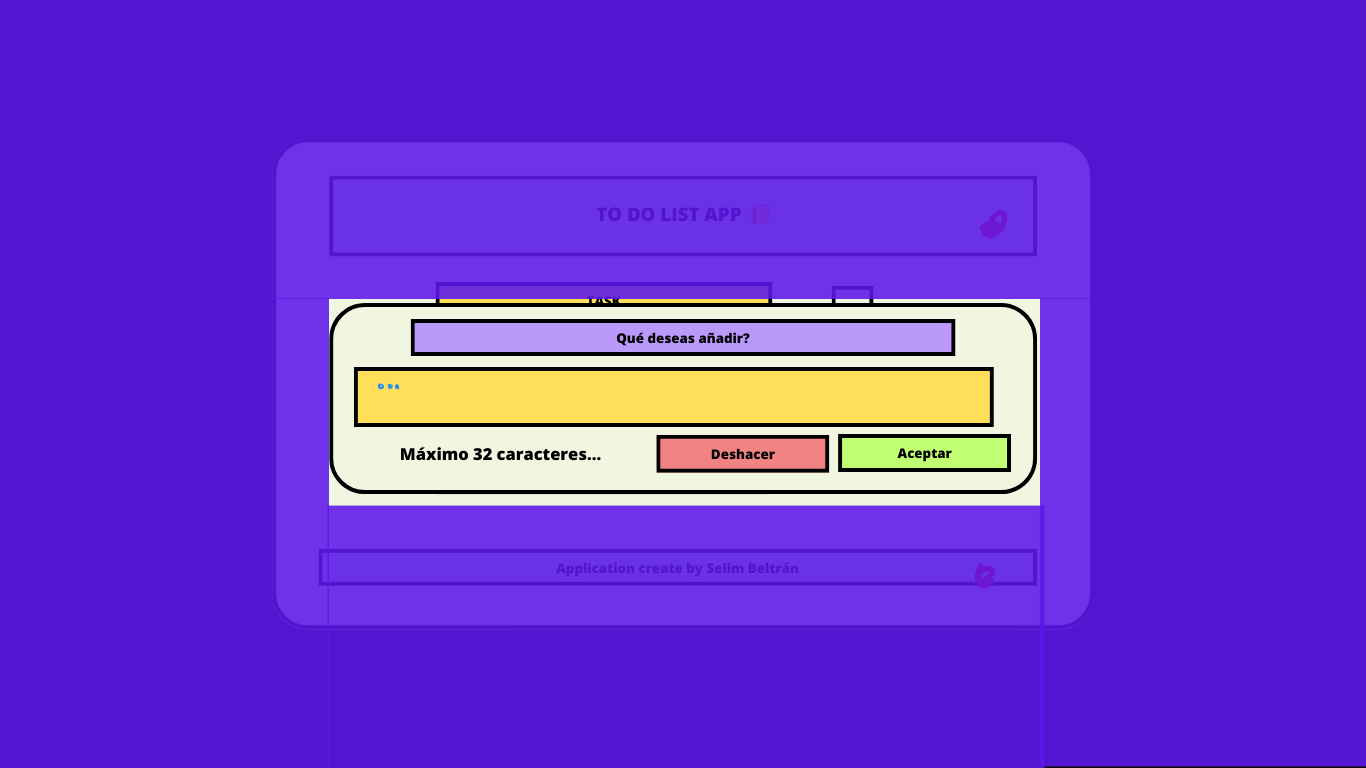
Este proceso es de suma importancia ya que se buscará como hacer surgir la ventana emergente sin afectar a ningún proceso principal:

¿Existe solución?

Si, consiste en usar un Z-índex que hará tener mayor jerarquía a cada uno de las figuras para lograr una mejor visión. Como no es necesario que todas las figuras sobresalgan del recuadro tendrán un z-índex iguales, pero mayores a los iniciales.

Es importante entender que el proceso de aparición de la ventana se hará por medio del botón de llamado.

**Proceso #3**





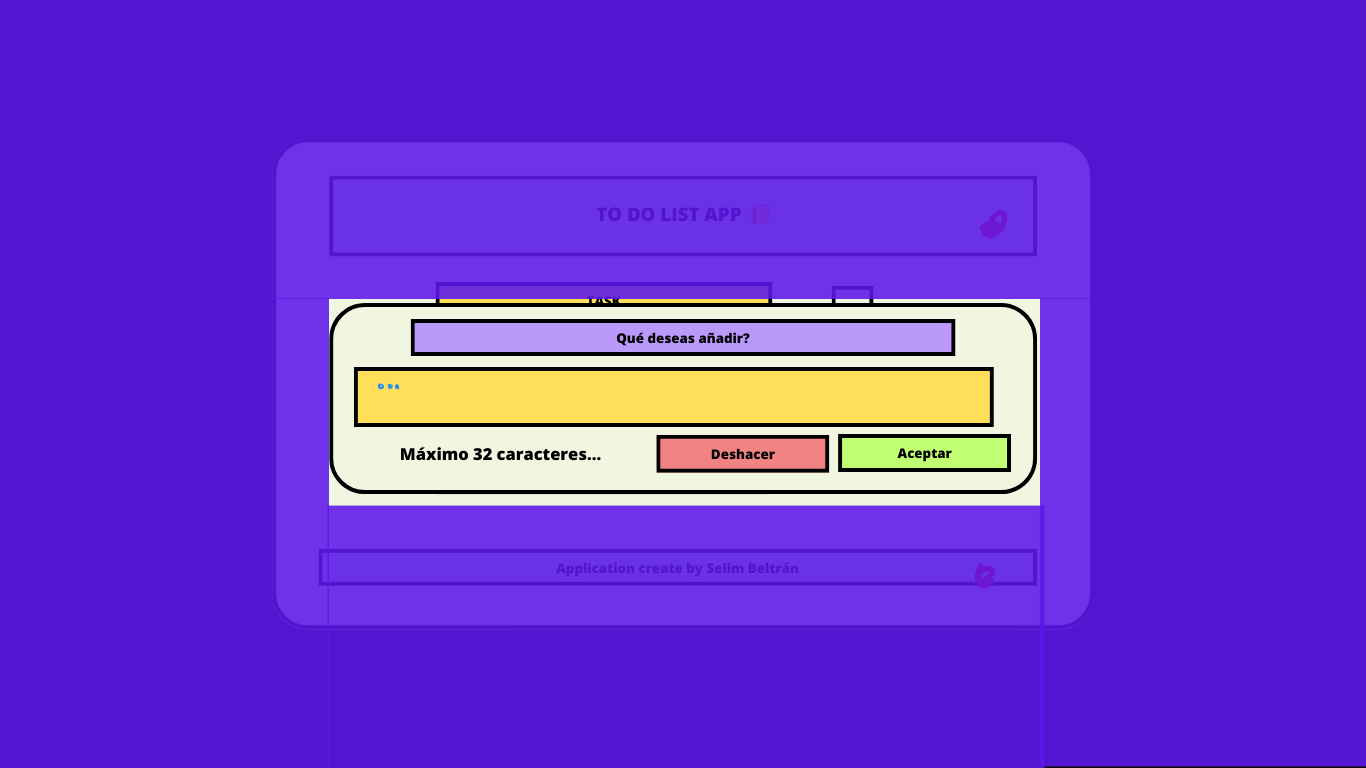
En este proceso se hará la primera aparición del rectángulo contenedor, el cual será el primer ‘’experimento’’ del proyecto. A continuación, se redactará como deben ser sus propiedades:

* Id: ‘’contenedor--tarea’’
* Color de fondo: #F1F6E1
* Tanto su altura como anchura serán definidos de manera arbitraria sin alejarse mucho de lo que se busca. Deberá ser lo suficientemente grande para que todos los objetos se encuentren en armonía.
* Desempeñará su función como contenedor, por ende, su posición relativa es necesaria en este caso.
* Con existen elementos que no se deben centrar no se hará el uso de display u otros efectos innecesarios.
* **Recuerda que** esta debe colocarse en el medio de la página web, por lo tanto, el uso del margen, display o padding será necesario en este caso.
* Por su forma obtendrá un rol de relleno en su radio por que provoca un efecto de armonía en el diseño, su color será: #000000.
* Su Z-índex será igual a 40.

/\*El proceso se logró por medio de dos elementos, escuchar de elementos y estilos de visibilidad\*/

/\*Se encontró elementos importantes en la investigación: display none y block\*/

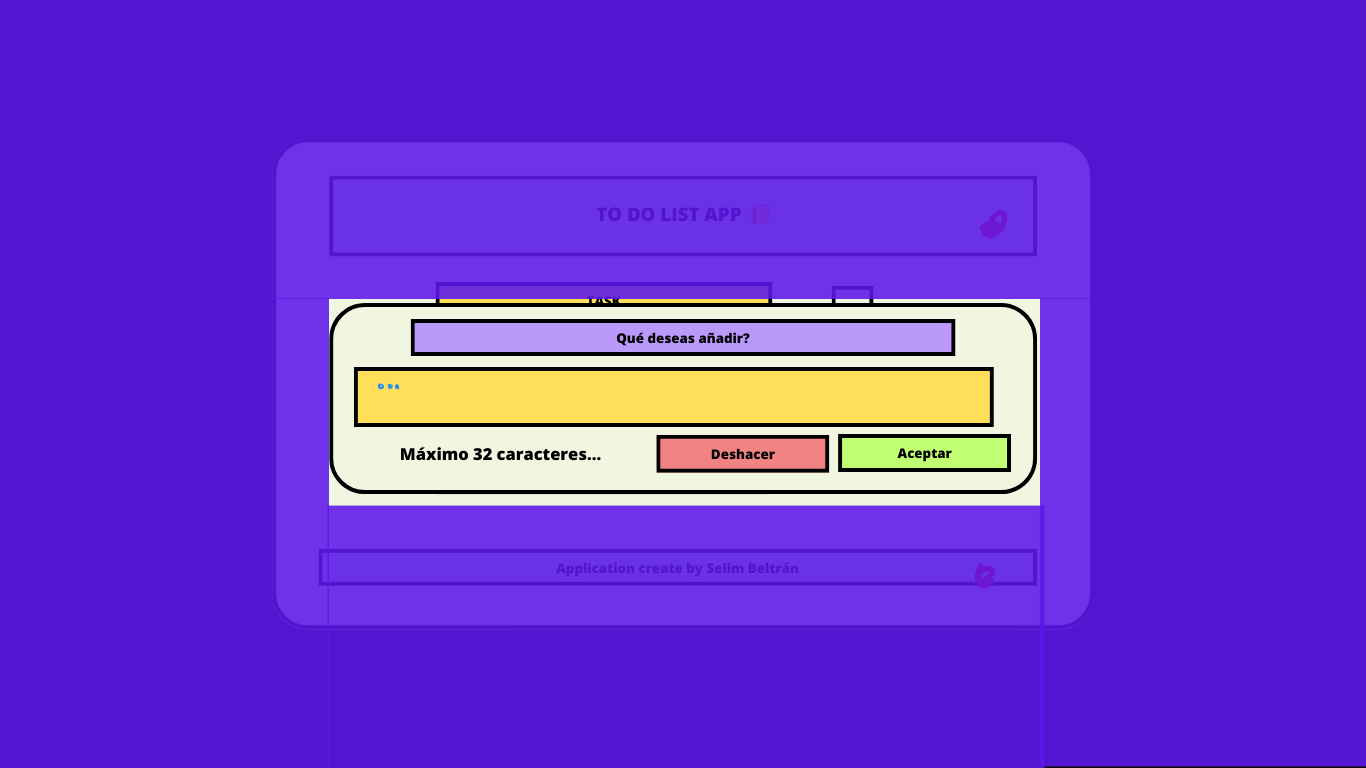
**Proceso #4**





* Sus propiedades serán igual a las primeras figuras, pero lo única que varía serán los tamaños de su grosor y alto.
* A groso modo se hará un copiar y pegar de los mismos elementos:
* ID: titulo--tarea
* Background color: #BA99FA
* Centrado al contenedor; ubicación de acuerdo a la imagen.
* Su Z-índex será igual a 40.
* Por su forma obtendrá un rol de relleno en su radio por que provoca un efecto de armonía en el diseño, su color será: #000000.
* Contenido:
  + Tipo: párrafo
  + Id: frase-titulo--tarea
  + ¿Qué tarea deseas añadir?
  + Su tamaño de letra será arbitrario.

**Proceso #5**





Sin duda el inicio de todo fin, dejando las bromas de lado, un proceso riguroso, pero fácil de manejar… En esta figura deberá contar:

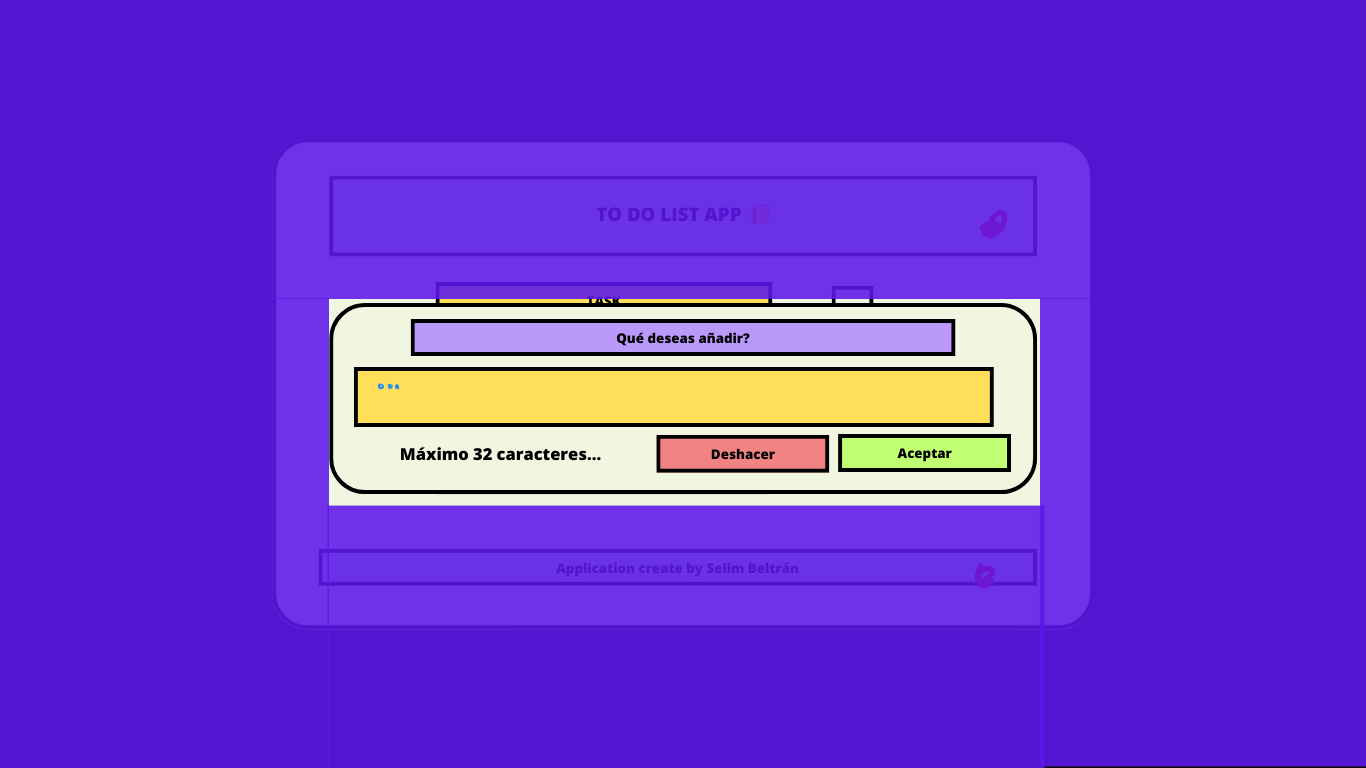
* Un input que permita la entrada de escritura de un usuario. Se tendrá que moldear y ocultar con el fin de obtener solo en marco amarrillo y que se pueda escribir y enviar de la misma manera.
* Input id: usuario-texto--tarea
* Id: escribir-texto—tarea
* Color de fondo: #FFDE59
* Por su forma obtendrá un rol de relleno en su radio por que provoca un efecto de armonía en el diseño, su color será: #000000.
* Centrado al contenedor; ubicación de acuerdo a la imagen.
* Su Z-índex será igual a 40.
* Se conoce la idea de que debe ir texto dentro del cuadrado, sin embargo, el uso de display o margen no será ocupado ya que el usuario será el que escriba y no es obligatorio que el texto empiece desde el centro.
  + Un hecho a tomar en cuenta en conjunto con la idea primordial es que esta debe contar como máximo con 32 caracteres a escribir, si no, presentara fallas al momento de su optimización.

¿Cómo se pretende logar esta hazaña?

* + Se hará una búsqueda la cual nos permitirá colocar un limitador de caracteres en el input.
  + Aquí se escribirá lo que se obtuvo de dicha búsqueda:

Se uso la propiedad maxlength en la cual el input nos permite interactuar con el máximo caracteres. Y se usará un minlength para que el usuario no ponga payasadas.

**Proceso #6**



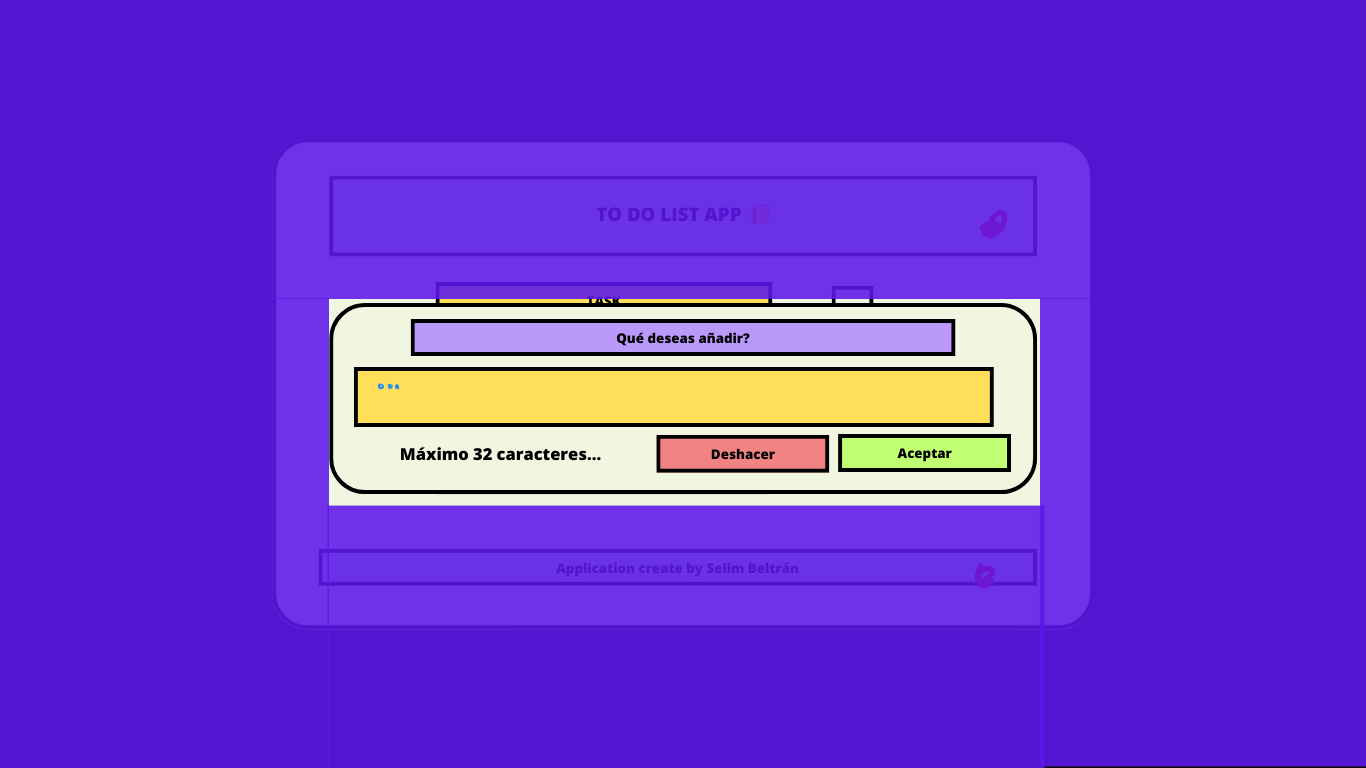


Se colocará una etiqueta de párrafo la cual **informara al usuario** cuantos caracteres están permitidos en el proceso #5.

Contará con las siguientes propiedades:

* Id: maximo-aviso--tarea
* Su posición será un poco alejada de la esquina inferior izquierda.
* Su Z-índex será igual a 40.
* Contenido:
  + Máximo 32 caracteres…

[**Proceso #7**](Día%20#8)





El siguiente proceso será una etiqueta de botón y su función será cerrar la pestaña de creador de tareas sin ejecutar ningún proceso, en pocas palabras, un botón de cancelación:

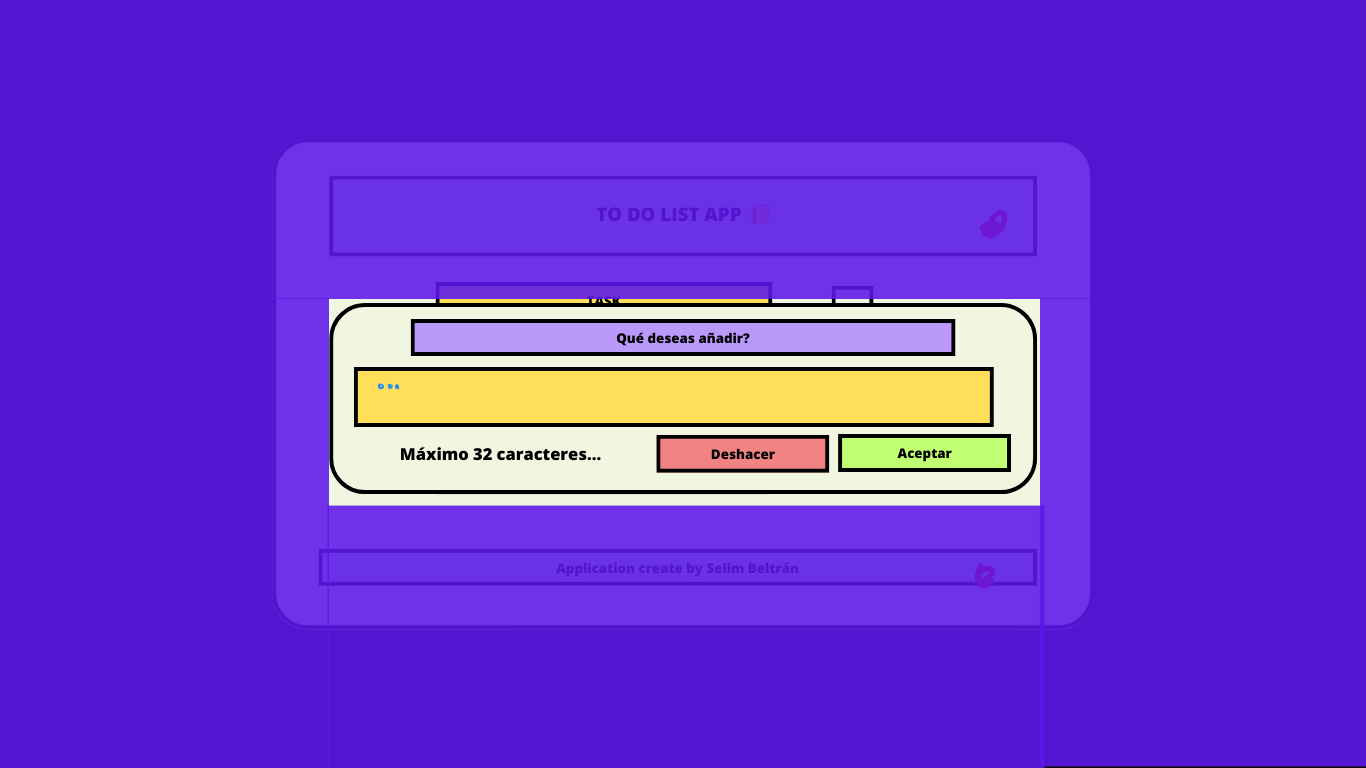
Las propiedades de la caja son:

* Id: cerrar-creador--tarea
* Color de fondo: #F18383
* Por su forma obtendrá un rol de relleno en su radio por que provoca un efecto de armonía en el diseño, su color será: #000000.
* Contará con un display **flex** para el botón, ya que, este si necesita de un texto coordinado y firme, además de controlar el uso de su margen, ya que no puede ser de bloque.
* Su Z-índex será igual a 40.
* Ubicación: como se indica en la imagen… apegado al botón ‘’aceptar’’.

Las propiedades del input son:

* Contenido del botón:
  + Id: input-deshacer--tarea
  + ‘’Deshacer’’
  + Ocular el color de fondo del input, marco y resalte.
  + Tipo: botón

**Proceso #8**

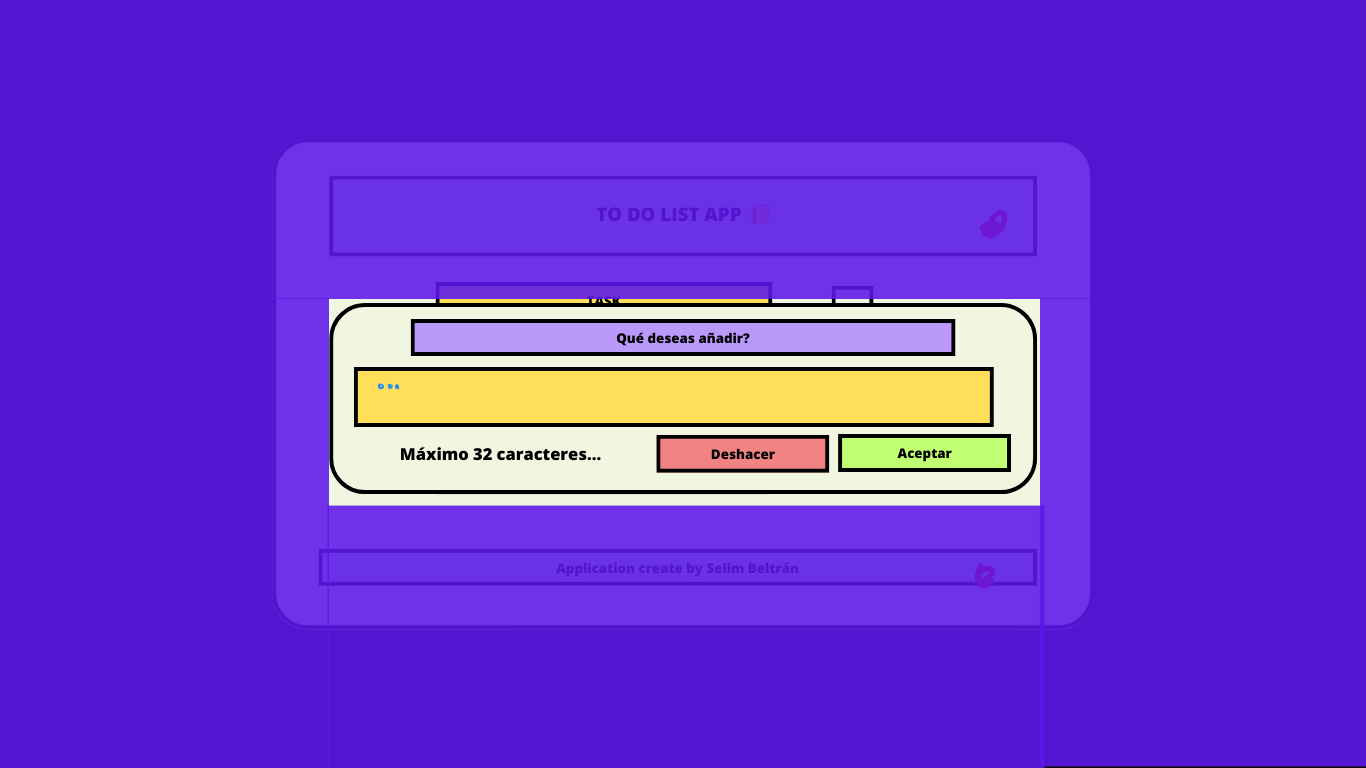




El penúltimo proceso deberá ser un botón, el cual cerrara la ventana emergente, pero no eliminara el mensaje que escribió el usuario, si no que la guardara, sin embargo, este proceso mencionado queda en descarte por el momento, ya que la función de este botón será ligada a otra, pero será responsabilidad de la próxima acción.

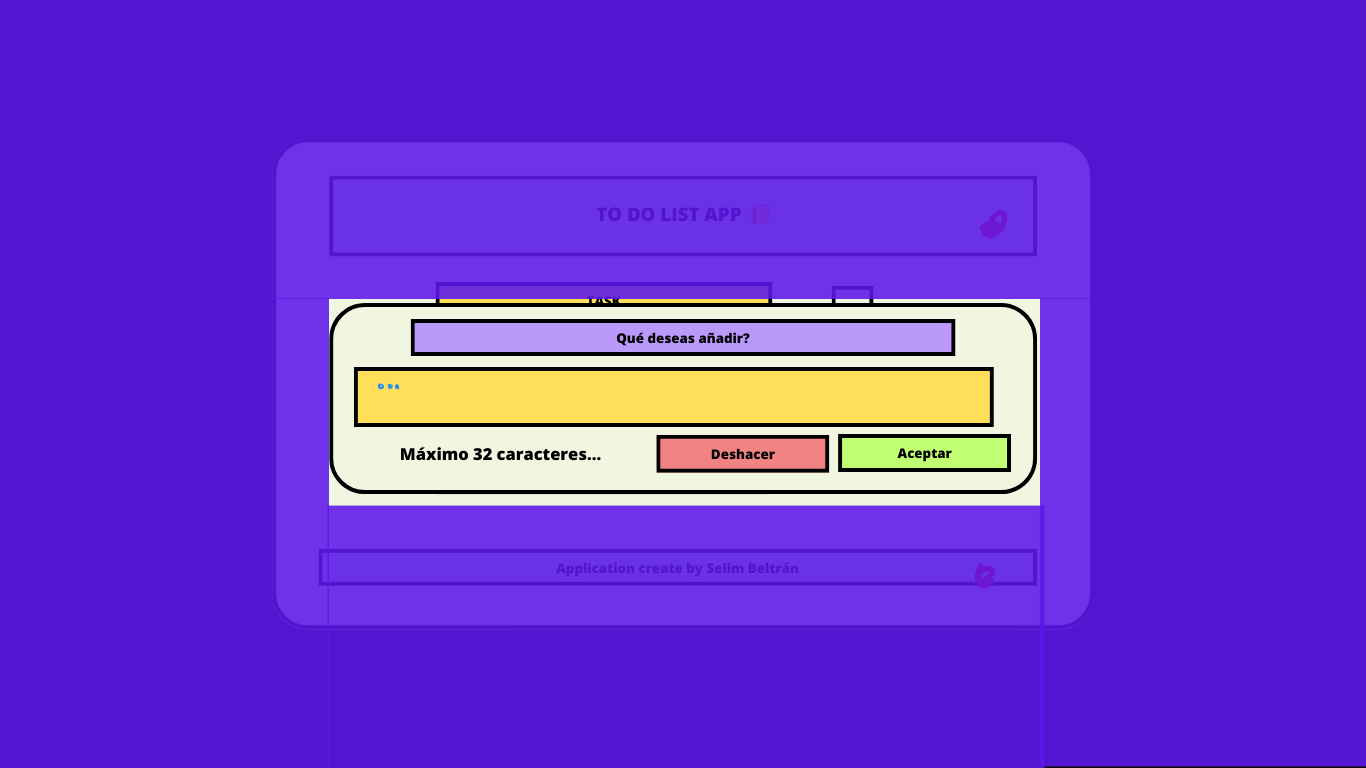
Por el momento deberá contar con las mismas propiedades que su vecino, así que, a groso modo se copia y pega el proceso con sus respectivos cambios:

Las propiedades de la caja son:

* Id: crear-creador--tarea
* Color de fondo: #C1FF72
* Por su forma obtendrá un rol de relleno en su radio por que provoca un efecto de armonía en el diseño, su color será: #000000.
* Contará con un display **flex** para el botón, ya que, este si necesita de un texto coordinado y firme, además de controlar el uso de su margen, ya que no puede ser de bloque.
* Su Z-índex será igual a 40.
* Ubicación: como se indica en la imagen… apegado al botón ‘‘deshacer’’.

Las propiedades del botón son:

* Contenido del input:
  + Id: input-crear--tarea
  + ‘‘Aceptar’’
  + Ocular el color de fondo del botón y su marco.

**Proceso #9:**



Como paso final se agregará la opacidad correspondiente al cuadrado, este proceso fue llamado proceso secundario, que fue explicada en páginas anteriores, pero se procederá ahora con más detalles.

Sus propiedades serán:

* Su fondo deberá ser opaco de manera blanquinosa, es decir, como se muestra en la siguiente figura:



Lo que se espera es que el fondo este atenué a los marcos de atrás, por lo tanto

se considerará el uso de un fondo no muy desenfocado color blanco.

* Color de desenfoque: #FFFFFF
* Aún no sabemos cómo actuará el fondo, pero se espera que sea un rectángulo grande con un **Z-índex de 35** para así tener la suficiente jerarquía para tapar el fondo, pero no lo que se debe mostrar.
* Su posición será en todo el archivo HTML y **se accionará cuando el creador de tareas se abra**.
* Id: desenfoque-creador--tarea

Esto es todo por hacer en el proceso de la creación del iniciador de tarea, si surgen problemas se comunicará por medio del documento.

**Descubrimiento del día #12…** Se descubrió que no existe una figura la cual su centro se encuentra vacío, por ende, se utilizó cuatro rectángulos equiláteros que cubran la necesidad del recuadro a crear.

A continuación, se darán más detalles de sus propiedades:

* Su posición será absoluta y se utilizaran etiquetas aparte para cada uno de ellas, además de utilizar una clase, la cual englobara cada una de ellas y esta será utilizada en **JS**;
* Su fondo será de color: #ffffff;
* Su forma será la que se expresa en la figura [#n](images/n.png).
* Contaran todas con un efecto blur(), además se da por medido que al ser 4 etiquetas de diferentes Id’s, pero igual clase se buscara en la medida de lo posible utilizar una propiedad para todas, por ejemplo, el color de los rectángulos.
* En este apartado se darán los siguientes identificadores y clases:
  + Clase: imgBlur
  + Identificadores:
    - Rectángulo #1: rBlur1
    - Rectángulo #2: rBlur2
    - Rectángulo #3: rBlur3
    - Rectángulo #4: rBlur4
* Se utilizo la propiedad ‘’opacity’’ para así lograr un efecto armonioso con el blur… En definitiva, el efecto blur con figuras llanas presenta un mejor funcionalidad con opacity, pero en imágenes funciona de manera individual.

Día #9:

Se descubrió huecos en las entradas de los div las cuales no permitían el flujo correcto de las etiquetas lo cual dificultaba mucho las tareas que seguían, la mayoría de estos problemas se resolvieron por medio de lógica de display y espacio a seguir.

Por ejemplo, se desenvolvió el padding que había como espacio en el body y se descubrió cómo funciona en viewport en relación al body y HTML.

Por último, se reajusto la posición de unas etiquetas y se mejoró su flujo, en el caso del formulario se movió de su contenedor original y en el caso del contenedor secundario se volvió absoluto.

Se descubrió la causa de un posible error futuro incierto que no se sabe si afectara el flujo de las entradas del cuadro HTML, sin embargo, es una especulación y no se espera nada de ella, ya que, no se presenta molestia por el momento.

Día #10:

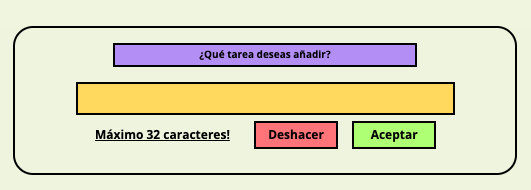
Se encontró un diseño des armónico en el id ‘’ usuario-texto--tarea’’, se arreglará al final del proyecto, ya que es un error de diseño no importante.

Se produjo el conocimiento de una nueva alternativa para la armonía del diseño y resulto en un buen estilo, además, cabe recalcar que al final de este proceso, eliminando la armonía del diseño se tendrá que hacer los toques realistas.

Día #11:

Antes de indagar en la diferencia entre entrada y botón y él porque funcionan de manera diferente en el formulario y sin este, no sabe duda que un input permite rellenar opciones por medio de señales variadas y el botón en un formulario sirve para subir el total de la información dada en el formulario.

Se encontró un diseño estable y armónico, el cual es aceptado por su simplicidad y armonía.



Diferencia entre ENTRADA y BOTÓN:

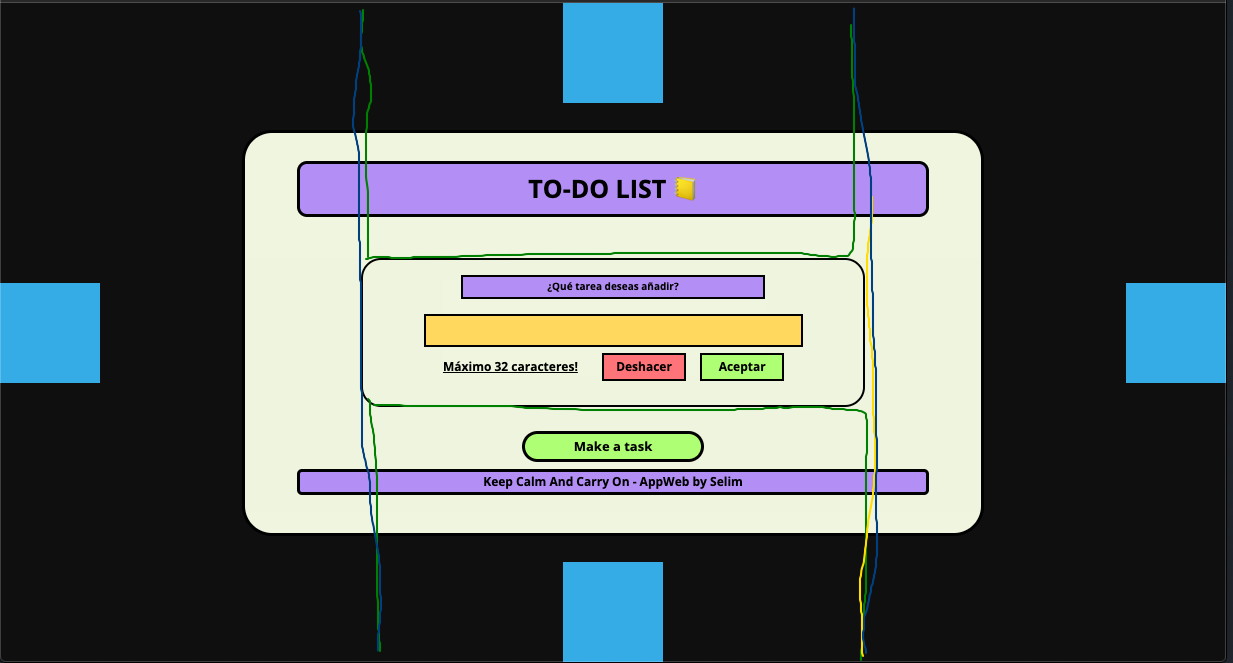
* La diferencia reside en su uso en formularios y contenido, por ejemplo, en términos de formulario el input debe tener un tipo SUBIR para que pueda subirse un formulario, mientras que, el botón tiene el tipo SUBIR adherido.
* También, existe otra propiedad que permite reiniciar al botón todos los inputs, el cual es RESET.
* Por último, la propiedad que más diferencia marca entre el input y el botón es su contenido… En el input se puede poner solo un tipo, mientras que en el BOTÓN se puede colocar imágenes y texto sumándole el SUBIR.

Por el momento encontramos dificultades en los procesos como {formulario: method, post y get} además de {input y su valor al hacer intro}.

Se indagarán el día 26 de septiembre de 2023…

Día #12:

Se encontró una solución para no sobrepasar el efecto del blur () que consiste en dividir de manera equitativa los rectángulos y la separación de sus partes:



Además, se eliminó el div padre que se había creado el día anterior ya que la clase funciona como nombre común para todos los div, ahora se tendrá que ver como llamar a una etiqueta clase en **JS**.

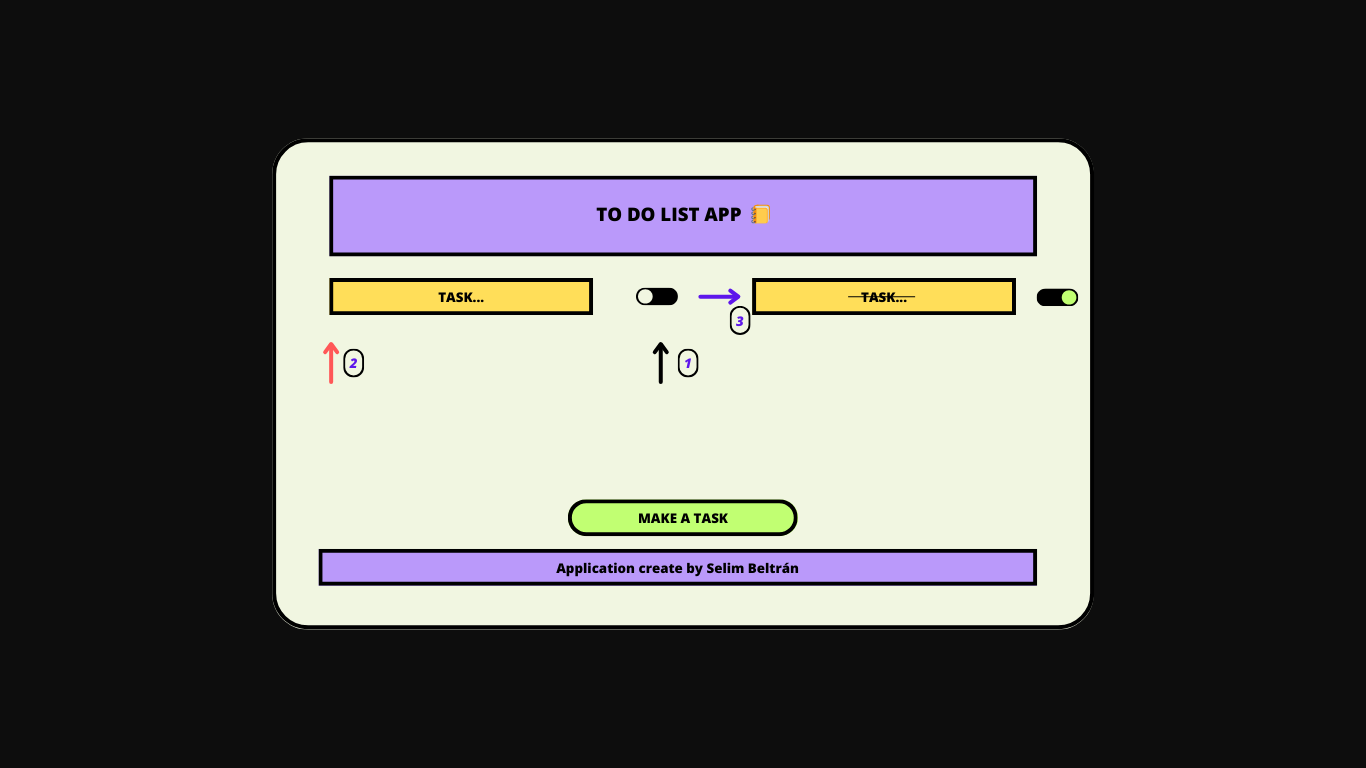
Día #13:

Un problema quedo pendiente en día número 12 el cual consiste en que la clase común entre todos sería: ‘’imgBlur’’, esta iba a ser llamada en **JS** y luego seguir el proceso para ocultarse:”.style.display:none”, no obstante surgió un súmmum, al momento de llamar la función en **JS** con la clase mencionada esta NO responde de manera adecuado como si de un div se tratase…

Entonces se busca solucionar este problema que consiste en ver de qué manera llamar a las clases y que se oculten todas al mismo tiempo.

Día #14:

Se termino por completo la fase número 3, ahora seguirá la fase número cuatro que consiste en conectar y recrear la tarea que introduzca el usuario en la “consulta”.

Veamos la siguiente imagen:

Como podemos observar se presentan varios aspectos a tomar en cuenta a la hora de que el usuario cree una tarea, dicho esto se procederá a explicar de manera clara lo que se quiere lograr en el proceso #4.

Fase #4:

Como la fase cuatro proviene de varios comandos y partes extensas estas se comprimirán en diversos procesos para así lograr un mejor ambiente de trabajo y facilitación en el manejo de tareas.

Como tema global, se debe intentar englobar todo en un div, es decir, si el proyecto lo permite independientemente de su grado de dificultad, el esquema será el siguiente:

<div id="englobardor-Tareas">

<div id="marco-Tarea1"> </div>

<input type="checkbox" id="visto-Tarea1">

<div id="marco-Tarea2"> </div>

<input type="checkbox" id="visto-Tarea2">

</div>

Como se puede observar en la medida de lo posible que las tareas esten siempre alado de sus padres, recordando que siempre estarán englobadas en un div mayor, el cual dara un limite de marco en css de acuerdo al límite del rectángulo #2 preestablecido.

A continuación, el desglose de los procesos:

Proceso #1;

La tarea que cree el usuario se guarda en el ‘’Query’’ y luego aparece con un marco de color amarillo como el marco #2 en la pestaña principal. Es decir, el usuario entra al apartado de Crear tarea, luego escribe su tarea y presiona aceptar… Luego la tarea que agregue el usuario deberá aparecer en el marco #2 sin modificar.

Se conoce que se necesita de varios aspectos como llamadas de **JS**, div’s y css. Por el momento, solo se procederá a crear una solo tarea, ya que se conoce que se deberán agregar varias, pero al momento de comprimir primero en unidades, se conoce que su resultado será el mismo.

Entonces, ¿que se necesita para que aparezca tal cual la tarea que ingresa el usuario y además centrado dentro del marco #2?

Sabemos por el momento que se necesita propiedades de CSS y un div para lograr guardar lo que escriba el usuario y obtener las propiedades del marco… Además de escribir un párrafo vacío que junto con **JS** cambie el DOM TREE y se guarde la tarea.

También se necesita llamar al id que tiene al input que ingresa el usuario, para así obtener la tarea en **JS**.

A continuación, se procederá a detallar las propiedades del marco de tarea:

* Se hará un div padre que contenga a todas las tareas, para facilitar deberes futuros y agilizar propiedades, además de ponerle un marco final a las tareas, para que estas funcionen mejor.
* **Id div padre**: ‘‘marco-tareas’’
* Color de fondo del **div hijo**: #FFDE59
* **Id div hijo**: “marco-tarea1”
* Por su forma obtendrá un rol de relleno en su radio por que provoca un efecto de armonía en el diseño, su color será: #000000.
* Contará con un display inline-**flex** o flex para el texto, ya que, este si necesita de un texto coordinado y firme, además de controlar el uso de su margen, ya que no puede ser de bloque.
* Su Z-índex será igual a 50.
* Contenido:
  + ID: ‘’usuario-tarea1’’
  + Input del usuario

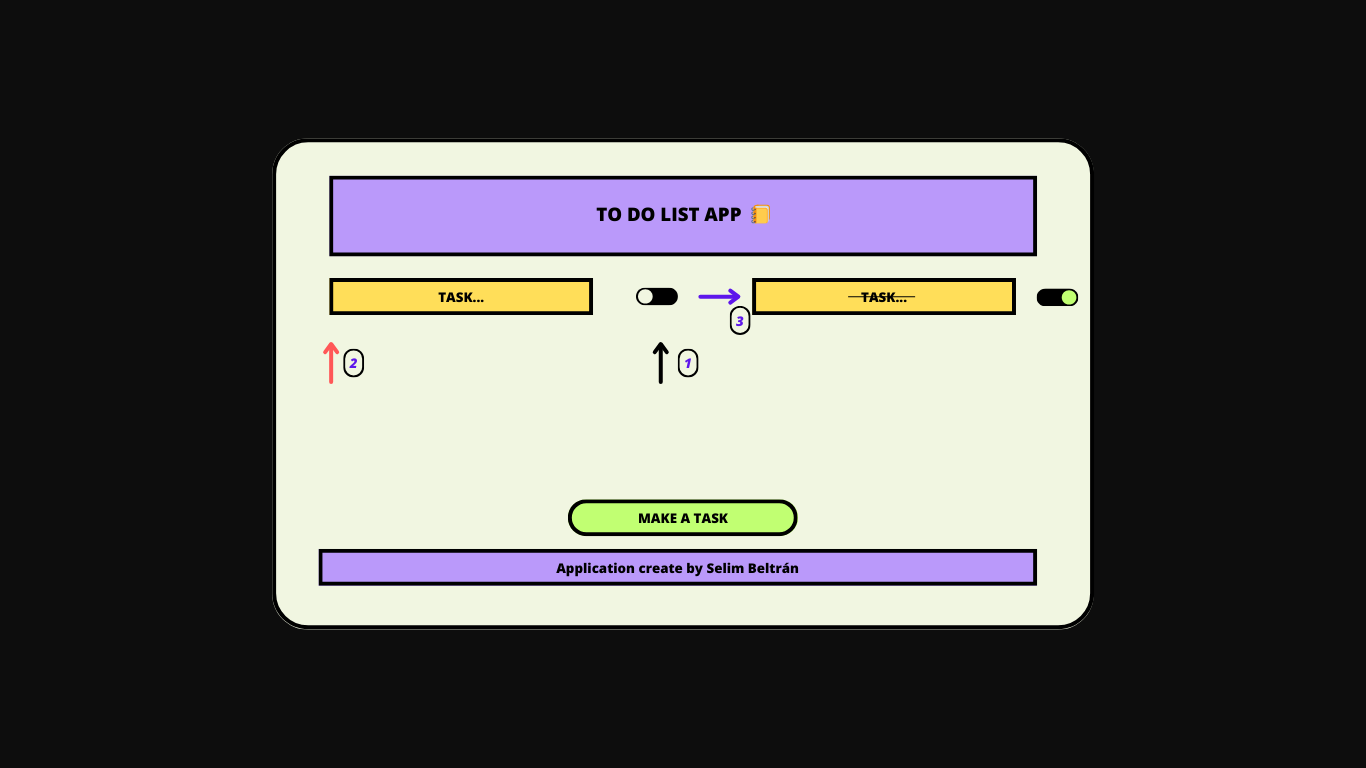
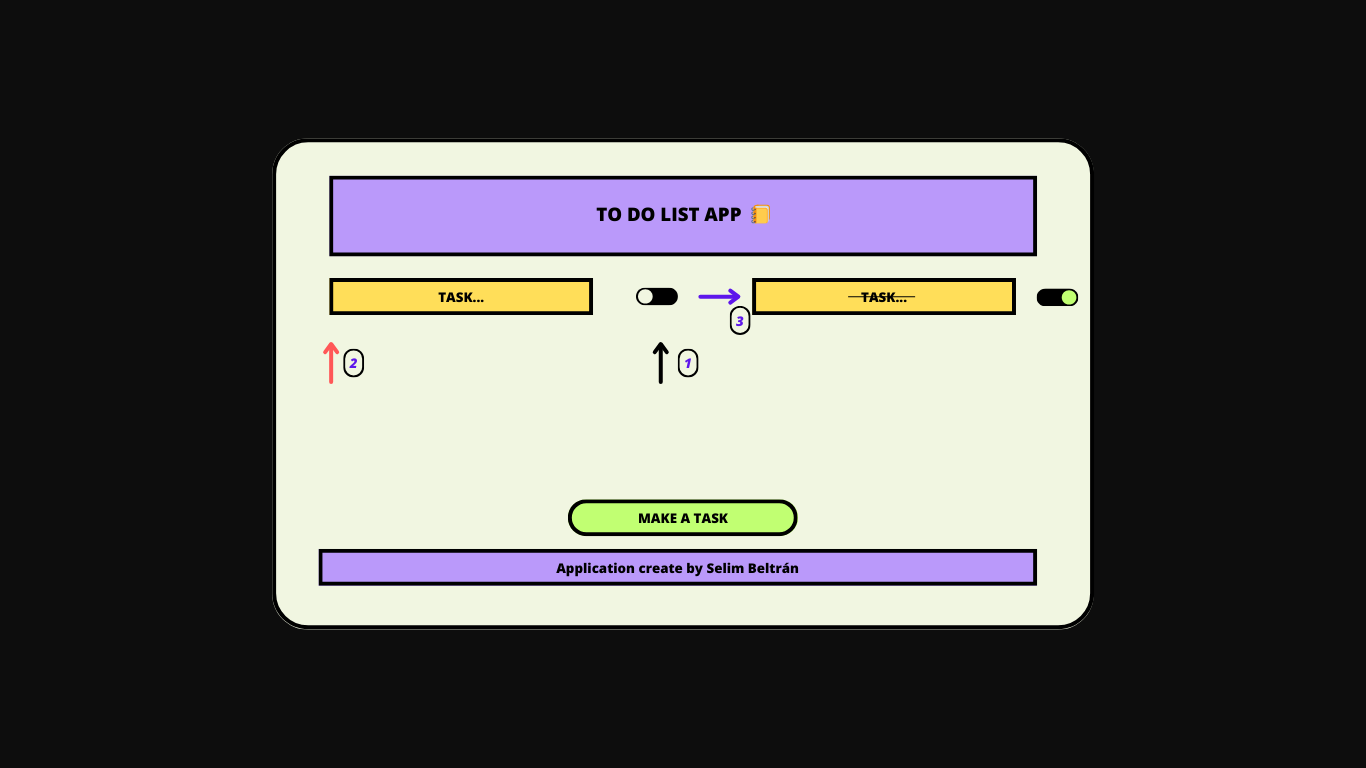
Por el momento se conoce que esto es lo necesario para el correcto funcionamiento del marco de tarea.

**Proceso #2;**

La fase número dos se conoce que cuenta con un ingente de propiedades, por lo tanto, el uso de **JS** será eludible; En la siguiente fase se busca crear un checkbox de desplazamiento por clic, el cual provocará que la tarea se marque con una raya mostrándole al usuario que esta fue completada…

Pero… ¿Qué se busca en concreto en esta fase?

Se busca crear el siguiente checkbox con sus debidas propiedades:



————— >

Aún se desconoce que se debe utilizar para crear un checkbox que funcione a tal magnitud, pero se averiguará en esta fase del proyecto, por el momento se conocen algunas propiedades llanas que van a ser aplicadas.

* El mismo checkbox puede lograr el efecto de transitividad así que solo se utilizará un solo checkbox, no obstante, habrá varias tareas, por lo cual su id será de 1.
* El input tipo checkbox contará con un el siguiente id: “visto-Tarea1”
* Su Z-índex será igual a 40.

El input contará con más propiedades, pero hasta el momento estas serán las más importantes.

Proceso #3:

Como último proceso se buscará combinar el visto del input con la tarea, es decir, se quiere lograr que, al momento de darle visto a la tarea, está se “subrayara” como si ya estuviera terminada…

Simplemente es eso… además de que el uso de **JS** será inexorable en este proceso, a continuación, se dará un breve detalle de lo que se espera utilizar…

* Llamadas a Id’s.
* Modificaciones globales al DOM TREE.
* CSS o en su posibilidad HTML.

Fuera de todas estas aplicaciones también si el programa lo requiere se puede crear una clase, nombre, id, etc… especial con **JS** que al momento de agregarse estás previamente agregadas en CSS se pongan subrayadas.

**Apartado de opiniones:**

Al momento de realizar el [proceso #2](#Proceso2) se deben hacer dos tareas, crear el marco en donde se guardará la tarea que solo aparecerá si y solo si hay algo escrito en el input y transcribir lo que hay en el input…

En este día solo indagaremos a como guardar el input y mostrarlo por medio de modificar el DOM.

Día #14:

[Documento de input](../../helps/callInput.docx)

Se encontró de manera exitosa como encontrar la llamada un Input, para más detalles leer el archivo mencionado.

Día #15:

Se hizo cambios en las etiquetas del proyecto, debido a que no ejemplificaban lo que se buscaba en estos, por ejemplo, había un botón el cual no contrastaba con los otros inputs, por ende, se cambió el botón a un input de tipo SUBIR.

Cambios:

<input type="text" id="usuario-texto--tarea" maxlength="32" value="" <class="inputs-tarea”>

<input type="button" value="Deshacer" id="input-deshacer--tarea"> <class="inputs-tarea”>

<input id="input-crear--tarea" type="submit" value="Aceptar"> <class="inputs-tarea”>

Se hicieron sus respectivos cambios a sus elementos interconectados en **JS** y **CSS**.

También cabe denotar que se agregaron clases Inter vinculadas a los inputs.

Class=”inputs-tarea”

Día #16:

Hemos creado otro documento acerca de las diferencias entre input y button con respectos a sus tipos reset, submit y button.

Más información aquí: [Documento](../../helps/Diferencias%20entre%20submit%20y%20button.docx).

‘’Las variables se utilizan de manera no arbitraria.’’

Se pensó en la idea de cómo aparecerán los párrafos al momento de crear varias tareas y se encontró una duda la cual consiste en como aparecerán las tareas mientras que el usuario va creándolas.

Podríamos utilizar la propiedad display none para ir apareciendo de poco a poco las tareas, pero no sería tan rentable, debido a que serán como máximo 6 tareas, es decir habrá 12 comandos de none y block…

La otra función consta en que el usuario hará clic en la tarea, eso en consecuencia arrastrara una funcionalidad del botón a estar en propiedad #2, la cual invocara la siguiente tarea, aunque su desventaja es que se deberá contar con sus 12 display…

No obstante, se la conoce como la opción más vocacional, ya que al final de cuentas estas harán resurgir las tareas, ya que deberán de estar ocultas de igual modo.

En el día 17 se comenzará un resumen detallado acerca de la estructura que perpetrará la base de las tareas.

PD: No se busca que los cuadros ya estén posicionados, sino que vayan agarrando posición.

Día #18:

Se creo un documento acerca de las funcionalidades utilizadas para conectar el input y que este aparezca en modo título.

[JavascriptFunctions](../../helps/newFunctions.docx).

Se creo un documento acerca de las funcionalidades utilizadas para conectar el input y que este aparezca en modo título.

Día #19:

Se logro la invención de la función input la cual constaba con muchas variables y funciones eludibles. A continuación, se presenta como se encontró la resolución de dicho proceso:

[Documento.](../../helps/Ifi.docx)

Una vez realizada esta tarea, es importante denotar la importancia del orden de los div’s en la aparición del documento, para así tener una estrategia predefinida de cómo se planea manejar estos problemas.

Vamos a partir por ideas, de las cuales la mejor será la que guiará el rumbo del proyecto. Pero antes de comenzar, pongamos en nuestro cerebro cuál es el objetivo de estos elementos:

En primer lugar, se busca que el usuario al momento de crear una tarea esta aparezca automáticamente por debajo de la primera tarea y en orden.

Por esto una de las ideas propuestas es tener creado cada input con su orden, previamente oculto con display y mostrado al momento de escuchar un segundo clic u alguna condicional.

Otra idea planteada es que los Id’s y los párrafos se creen solos, pero en base a la funcionalidad del programa esto será perjudicial para el mismo, ya que, si bien deben crear, estos también deben tener una posición exacta. {Aquí puede entrar en juego un campo oculto de conocimiento}

Como última propuesta planteada es que los Id’s y los párrafos se creen solos (JS), pero con etiquetas ya preestablecidas y elementos conectados.

La idea con más apreciación fue la tercera, la cual consta que a medida que el usuario crea tareas estas van apareciendo, ya que, agiliza el proceso el cual la computadora registra datos.

También se hace mención de un botón o input “eliminar” el cual ejecutará una serie de comandos que juntos, moverán partes del documento, pero este es un tema aparte.

Entonces, a partir de este **momento** se guiará el proyecto en base de:

* JS Y CSS
* Padres e hijos en HTML, y
* JS

Las etiquetas e Id’s en JS ya estarán predefinidas en CSS.

Por si no ha quedado claro la idea, esta consta de lo siguiente:

* Se deberán crear div’s y atributos predefinidos en CSS que al momento de crearlos estos ya estén posicionados.
* Se creará un id predefinido para cada tarea y una clase para todos (límite -> 6).
* Deberán estar uno debajo del otro, de modo que el diseño quede en armonía.
* **Existe** una funcionalidad la cual al hacer clic escuchar una condicional, pero esto queda para otro apartado. [Idea: Condicional que verifique que sí la tarea[x] esta creada]

¿Por dónde empezar?

1. **~~Crear los nombres de cada div a usar.~~**
2. **~~Crear div’s falsos y darles las posiciones correctas del elemento.~~**
3. **~~Anotar rigurosamente cada posición.~~**
4. **~~Aprendizaje de padres e hijos en etiquetas.~~**
5. [**~~Aprender conceptos de creación de etiquetas.~~**](https://developer.mozilla.org/en-US/docs/Web/API/Document/createElement)
6. **~~Acción y verificación de etiquetas en su respectivo orden.~~**

Pasos por dar:

* Quitar funcionalidad del Input “Enter” -> submit.
* **Aparición** de condicional de cada tarea con respecto al clic.

Paso #1:

* El to do list contará con máximo seis tareas, por ende, existen solo seis div’s a crear y sus párrafos:

Para toda la recreación del marco contará con una clase general tanto en el div como en su párrafo, la cual será:

**Class en el div** = “marcos-texto-tarea”

**Class en el p** = “volúmenes-texto-tarea”

Se hará con el fin de simplificar procesos en donde todos los elementos ocupen las mismas propiedades.

**Marco uno:** “marco-tarea-uno”

**Párrafo uno:** “usuario-tarea-uno”

**Marco dos:** “marco-tarea-dos”

**Párrafo dos:** “usuario-tarea-dos”

**Marco tres:** “marco-tarea-tres”

**Párrafo tres:** “usuario-tarea-tres”

**Marco cuatro:** “marco-tarea-cuatro”

**Párrafo cuatro:** “usuario-tarea-cuatro”

**Marco cinco:** “marco-tarea-cinco”

**Párrafo cinco:** “usuario-tarea-cinco”

**Marco seis:** “marco-tarea-seis”

**Párrafo seis:** “usuario-tarea-seis”

Resultado:

<div *id*="*marco-tarea-uno*" *class*="*marcos-texto-tarea*">

<p *id*="*usuario-tarea-uno*" *class*="*volumenes-texto-tarea*"></p>

</div>

<div *id*="*marco-tarea-dos*" *class*="*marcos-texto-tarea*">

<p *id*="*usuario-tarea-dos*" *class*="*volumenes-texto-tarea*"></p>

</div>

<div *id*="*marco-tarea-tres*" *class*="*marcos-texto-tarea*">

<p *id*="*usuario-tarea-tres*" *class*="*volumenes-texto-tarea*"></p>

</div>

<div *id*="*marco-tarea-cuatro*" *class*="*marcos-texto-tarea*">

<p *id*="*usuario-tarea-cuatro*" *class*="*volumenes-texto-tarea*"></p>

</div>

<div *id*="*marco-tarea-cinco*" *class*="*marcos-texto-tarea*">

<p *id*="*usuario-tarea-cinco*" *class*="*volumenes-texto-tarea*"></p>

</div>

<div *id*="*marco-tarea-seis*" *class*="*marcos-texto-tarea*">

<p *id*="*usuario-tarea-seis*" *class*="*volumenes-texto-tarea*"></p>

</div>

**Posiciones:**

#*marco-tarea-uno*, #*marco-tarea-dos*, #*marco-tarea-tres* {

*left*: *7%*;

}

#*marco-tarea-cuatro*, #*marco-tarea-cinco*, #*marco-tarea-seis* {

*left*: *50%*;

}

#*marco-tarea-uno*, #*marco-tarea-cuatro* {

*top*: *26%*;

}

#*marco-tarea-dos*, #*marco-tarea-cinco* {

*top*: *41%*;

}

#*marco-tarea-tres*, #*marco-tarea-seis* {

*top*: *56%*;

}

Día #19:

Se investigo a detalle acerca de las diferencias y definiciones de conceptos importantes al momento de modificar el DOM TREE, ya que, por medio de JS estos serán relevantes en futuros proyectos.

Detalles aquí: [Documento](../../helps/pAndc.docx).

Día #20:

Se realizo una investigación exhaustiva de la utilidad de JS al momento de insertar etiquetas, se hizo una buena comprensión del tema dado:

Detalles del documento: [Documento](../../helps/creaciónDeElementos.docx).

Ahora, nos encontramos en el mini proceso 6 del proyecto. Recordemos que en el siguiente mini proceso se deberá crear los div’s y etiquetas a partir del conjunto que las contiene, en este caso:

<div *id*="*marco-tareas*">

Día #21:

Se escribió un documento acerca de la interacción con el teclado del agente y como conocer que teclas está presionando.

Más detalles aquí: [Documento](../../helps/metodoKC.docx).

Para el día #22 se propone a analizar con más profundidad como se hará el manejo de la aparición de las tareas, pero se tiene por superstición que será un condicional que verifique que una tarea esta subida y que si no lo está simplemente no crea la tarea.

Día #22:

La mayoría de la infraestructura de las tareas ha sido creada. Desde su aparición hasta su estilo, sin embargo, aún queda una tarea pendiente, la cual es la aparición en orden y de en uno en uno cada tarea.

Es decir, uno vez aparecido el marco uno, invoque al marco número 2 y sucesivamente.

Se planteó una idea la cual consiste en que por medio de un booleano esta verifique si dicho elemento ya ha aparecido y por ende busque aparecer en el siguiente texto.

Aunque aún quedan varios huecos por resolver:

1. ~~Reiniciar input a escribir aceptar.~~
2. ~~Aparición en el documento.~~
3. ~~Orden de eliminación futura.~~

Para el paso #2 pondremos todos los elementos en display none, luego hacemos que la primera opción salga apenas se ejecute el primer input. Una vez realizada se realizada se llamará a la clase en común y un loop verificará si el elemento está disponible y desbloqueará el siguiente elemento. Pondremos que entre en estado inline-flex si se cumple haciendo “i+1”. Así logramos que uno por uno emerja de cada verificación del documento.

Como último paso, haremos que el input que escriba el usuario también se mueve en la otra casilla, utilización el mismo loop, es decir, una implementación al cuadrado.

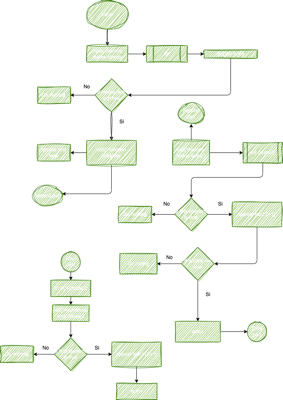
Vamos a utilizar las siguientes etiquetas para denominar cada llamado:

var booleanoMarcos.

var clasesParrafos.

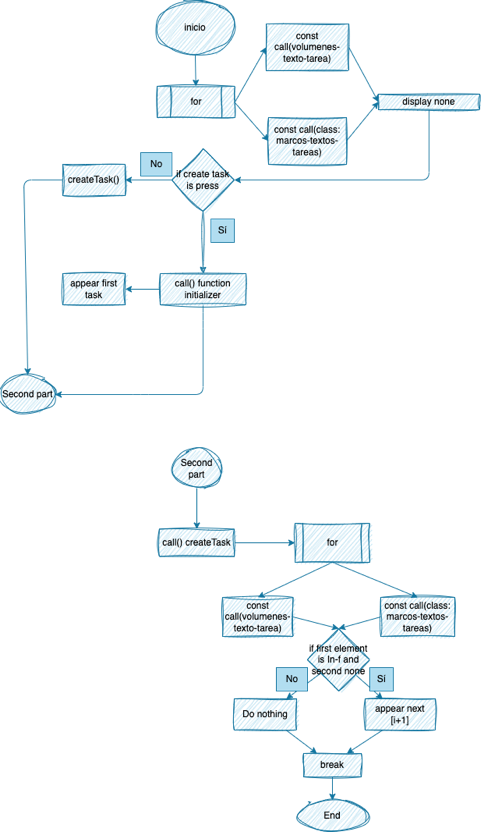
Día #23:

Se mejoró de manera significativa el diagrama por el cual se busca implementar uno de los mini procesos del proyecto.



Día #24:

Se ha culminado con el diagrama UML el cual buscaba representar de manera visual el proceso el cual se llevó a cabo en la aparición de las tareas.



Después de todo el proceso finalizado, se plantea la siguiente duda… ¿Cuál será el proceso a realizar si el usuario elimina una tarea? Se conoce que:

Primero: Habrá un botón que permita al usuario eliminar una tarea reciente o antigua.

Segundo: Las tareas se mantendrán en su posición, aunque no presente un diseño armonioso es presentable y cumple su función.

Una vez visto los pasos dos primeros pasos, podemos decir que, la solución es correcta y que tal como está, no presentará ningún problema en el futuro.

Idea:



Pero no somos conformistas, por lo cual, haremos que las tareas retrocedan y obtengan un nuevo orden, aprovechando al máximo ambas funcionalidades.

Se dio “masa” a una idea y es hacer todo lo contrario a la función “create task”, es decir, el condicional if será lo inverso lo cual significa que esta comprobará si es none y luego flex, en consecuencia, moverá el párrafo que se tiene presente al anterior y poniendo al anteriormente presente como none, provocando así que los demás retrocedan junto con los demás.

Cómo se mencionó aún la idea no tiene ni fundamentos y diagramas lo cual complica la interpretación de la misma, pero se clarifica que tiene total argumento funcional.

Interpretación a detalle del micro proceso #3:

¿Qué tenemos para crear la idea?

Contamos con la creación de etiquetas en las cuales tienen sus respectivas clases e Id’s, podemos jugar con estas para suplantar su valor por el siguiente y borrar su estado actual. Por ejemplo:



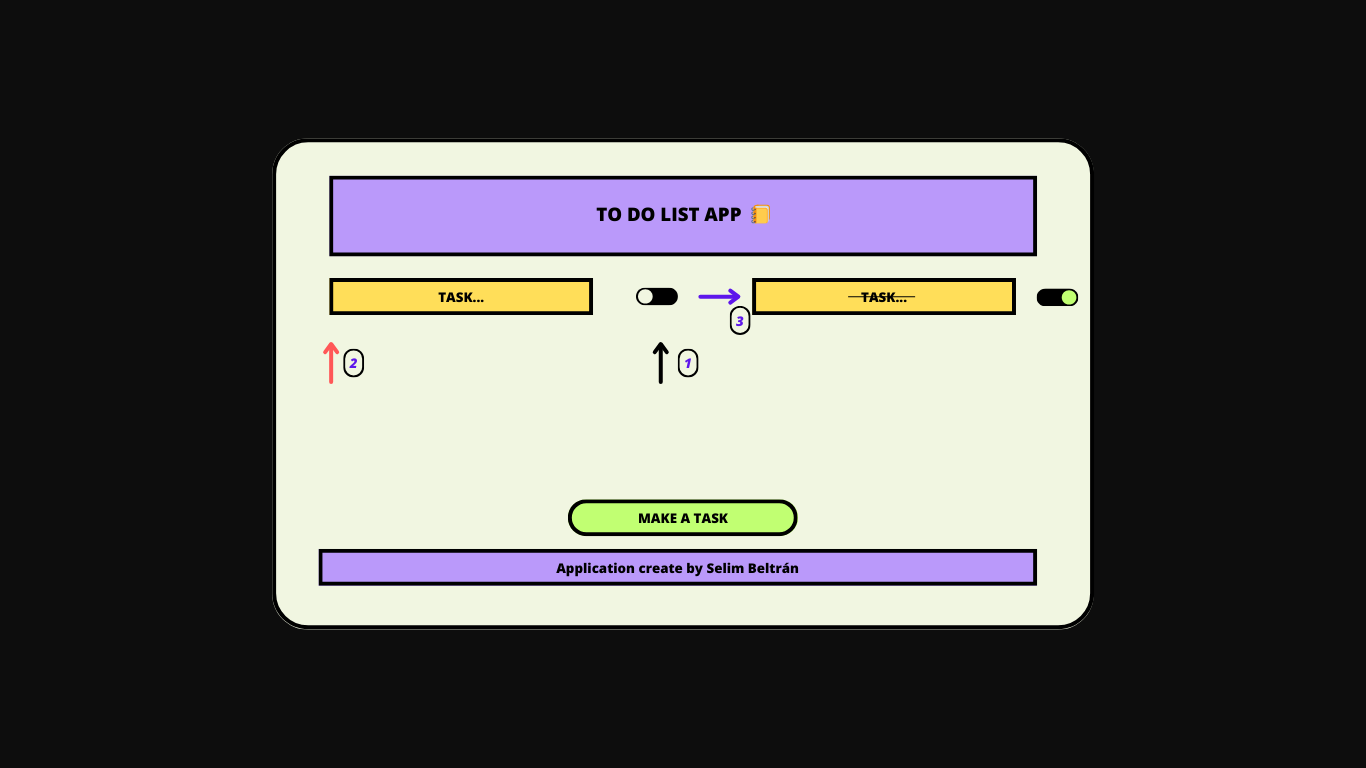
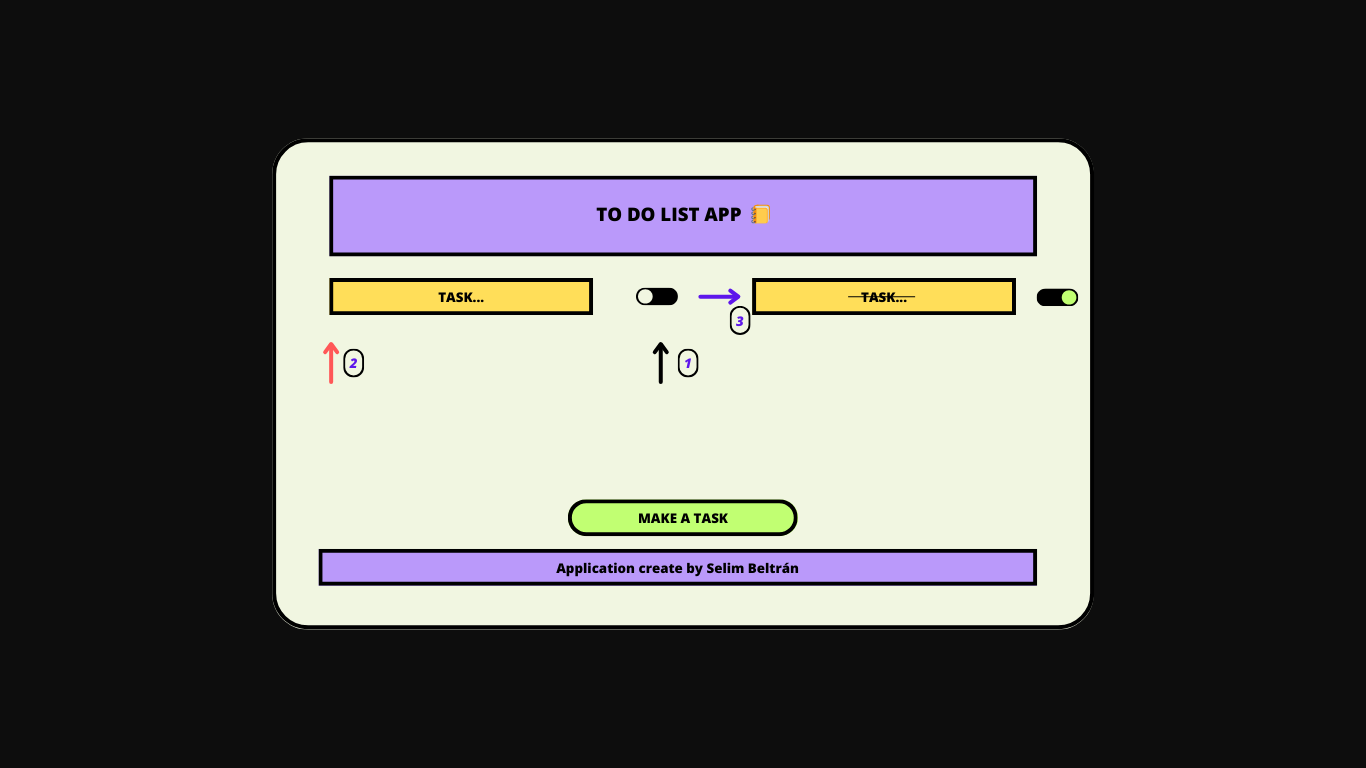
Como podemos observar, es un bucle él cual es activado si y solo si, mínimo 2 tareas activadas.

Lo que se tiene por ver es la aparición de estos trasladar cada nuevo valor a su nuevo lote.

Día #25:

Se empezó en el proceso de la fase número dos, por lo cual es importante retomar el proceso de CSS, ya que se utilizará como herramienta base.

Se recuerda que la creación del checkbox será el siguiente:



🡪

🡪

🡪

Se realizará el mismo proceso de creación de elementos como el proceso #1, pero esta vez con inputs, ya que se busca implementar la unión entre la caja con su respectivo checkbox.

Día #26:

Como nuestra pericia en el uso de CSS es muy limitada, este día se utilizará para la retroalimentación de este “lenguaje”. Como prueba se anexarán documentos en donde se hablará sobre lo aprendido.

Más detalles aquí -> [Documento](../../helps/CSSTips.docx);

Día #27:

Se descubrieron nuevas metodologías para el uso de CSS el cual simplifica el proceso por el cual es cargado el sitio web, además permite ajustar de manera sencilla todo el contenido de la página web, por ende, se hará un reajuste a todo el sitio, permitiendo una mejor comprensión del mismo. En el siguiente día se mostrarán los diferentes cambios que presentarán cada etiqueta.

#1:

\* {

font-family:'Futura';

}

#2:

We eliminate the selector body and HTML

.cuerpo{

background-color: #0D0D0D;

height: 100vh;

display: flex;

justify-content: center;

align-items: center;

}

And we get all in the principal DIV who we change it into a class with the same characteristic.

#3:

For the moment we change a lot of properties, so, it is difficult to put a lot of change to it, and instead of put a lot of selects without any sense, we will put only the important changes in the document. For example, the change number two is no more available.

Day #28:

Today, we are going to continue with the changes.

Day #29:

We are going to continue with the change in JS and Css, because we reach the point between them.

In the important changes we have the simplify of the colors using a dictionary:

:root{

--coloring-p: #BA99FA;

--coloring-g: #C1FF72;

--coloring-w: #f1f6e1;

--coloring-y: #FFDE59;

--coloring-r: #F18383;

--coloring-b: #0D0D0D;

--bordering: 3px solid #000000;

--bordering-less: 2px solid #000000;

}

**We also have the change of the unimportant id and we change into class. Also, we change all pixel and percentage values into rem values with the objective to get a better performance in the app.**

*Examples*:

<div class="cuerpo">

<div class="contenedor">

<div class="titulo">

<div class="eslogan">

<div class="creador-tareas">

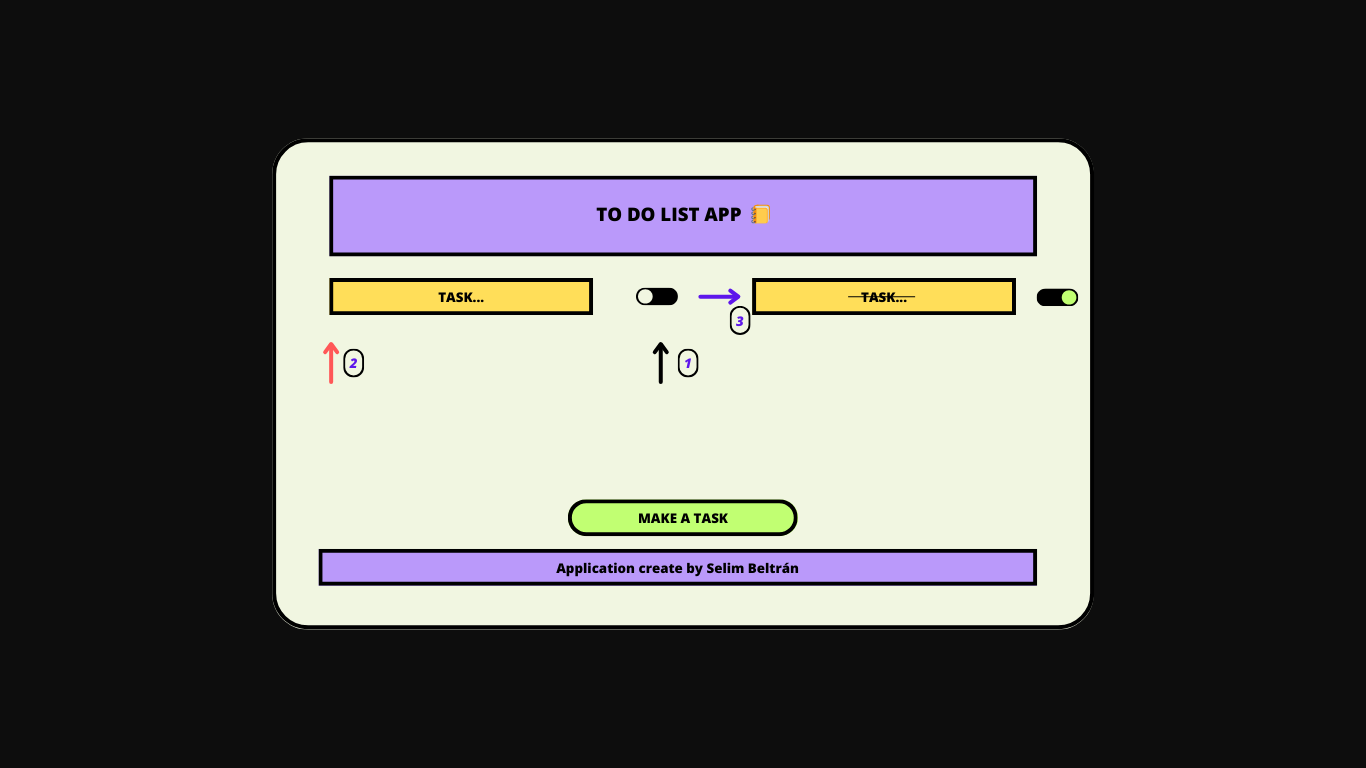
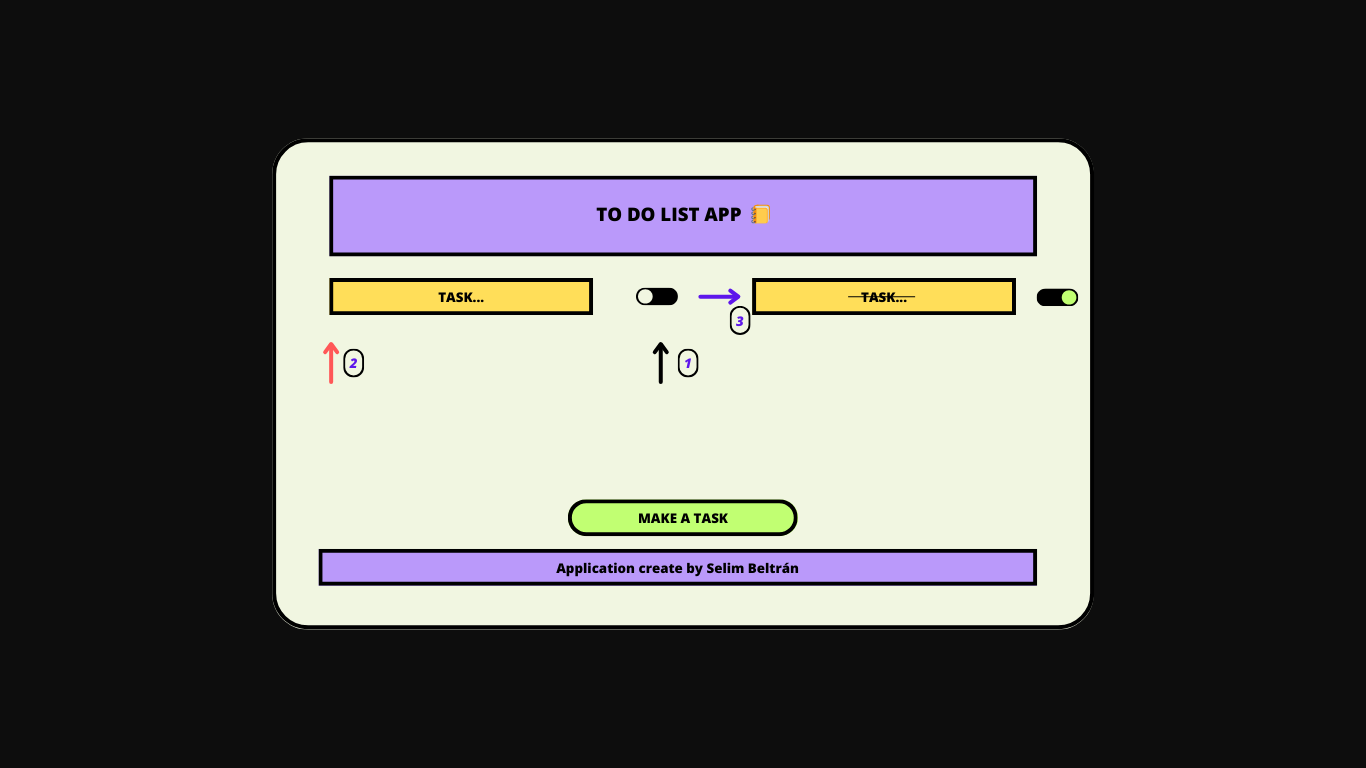
**And so on, the only properties we do not change are Inputs, buttons or functions that are import in JS.**

*Example*:

<div id="rBlur"></div>

**And finally, we compact some values in css that are repeatable in the document.**

**Now we are going to think about the properties to get the button, we are going to put some favorites properties that could make a good flow:**



#1:

We know that we are going to use a check box, because the function of the “icon” act like one.

Type: checkbox;

Id: check-Button;

Class: check-Button;

We can use the function TRANSITION and we can try to call the checkbox in Css to see how it works.

Day #30:

We are going to decompose the parts of button:

#1 Shapes

We suspicious that it is necessary two shapes:



As you can see, the rectangle is the “base” shape and the circle is going to be “transition” shape. Why? Because first it is going to be in “white” and if you press is going to be green.

#2 Properties

We see that the **rectangle** has the next properties:

1. border radius

2. background-color

3. z-index

4. width

5. height

6. position

And the **circle** has the following properties (could be changed):

1. color w => 1.1 color g

2. transition

4. width

5. height

6. position

For instance, the **checkbox**:

1. opacity

2. z-index

3. The same width and height of the rectangle

4. position

5. width

6. height

#3 Functionality

1. If you press the **hide** checkbox it will [transition](../../helps/transitionMethod.docx) the color white into green.

First, we are going to put into practice the order of the divs into shapes.

**Content** div: class: “container-shapes--task”;

**Rectangle** div: class= “rectangle--task”;

**Checkbox**: div = “checkbox-task”;

**Circle** div: class= “circle--task”;

We also have the problem of hiding the checkbox in the rectangle, but we have an idea, and it could be to put the same width and height of the rectangle in the checkbox and then hide it in CSS. Besides, we can also move the checkbox with position.

We hide the checkbox with opacity, because it seems like it doesn’t have the same characteristics as input type text or button.

Day #31:

We did the simulator of two things… the place of the button and the functionality of it.





But now we have a problem, and it is good to know what we can do in this situation. We only need to know how to restart all functions in JS, because when we reload the page, some functions don’t reload and he stuck in their places, but WE RELOAD THE PAGE.

Day #1/01:

The day before, we finished the function about underline the text. Now, we need to know how to do these processes into each paragraph and if the function “eliminate” is create, how you face up with the values.

First:

1. Create six “buttons” for each paragraph.

Second:

2. Appear it when we create a function.

Function(i):

i => parameter

Third:

Check it all is okay.

--First--

1. Put all the “buttons “with their paragraphs.

**Content** div: id: “container-shapes--task”;

**Rectangle** div: class= “rectangle—task0 => 5”;

**Checkbox**: class= “checkbox-task0 => 5”;

**Circle** div: class: class= “circle--task0 => 5”;

Blocked Code:

<div class="container-shapes--task">

<div class= "rectangle--task">

<input type="checkbox" id="checkbox-task" autocomplete="off">

<div class= "circle--task"></div>

</div>

</div>

We have done a lot of part of the project and we prefer to put the things in HTML than JS, because it had a lot of properties that could be down it.

We are going to put all changes we do:

<div id="container-shapes--task0" class="container-shapes--task">

<div class= "rectangle--task">

<input type="checkbox" class="checkbox-task" autocomplete="off">

<div class= "circle--task"></div>

</div>

</div>

<div id="container-shapes--task1" class="container-shapes--task">

<div class= "rectangle--task">

<input type="checkbox" class="checkbox-task" autocomplete="off">

<div class= "circle--task"></div>

</div>

</div>

<div id="container-shapes--task2" class="container-shapes--task">

<div class= "rectangle--task">

<input type="checkbox" class="checkbox-task" autocomplete="off">

<div class= "circle--task"></div>

</div>

</div>

<div id="container-shapes--task3" class="container-shapes--task">

<div class= "rectangle--task">

<input type="checkbox" class="checkbox-task" autocomplete="off">

<div class= "circle--task"></div>

</div>

</div>

<div id="container-shapes--task4" class="container-shapes--task">

<div class= "rectangle--task">

<input type="checkbox" class="checkbox-task" autocomplete="off">

<div class= "circle--task"></div>

</div>

</div>

<div id="container-shapes--task5" class="container-shapes--task">

<div class= "rectangle--task">

<input type="checkbox" class="checkbox-task" autocomplete="off">

<div class= "circle--task"></div>

</div>

</div>

Day #01/2:

Now, we need to separate one part of the code with this problem, because we can show the check buttons with the function “create task”, but how do we see the lines?

Idea One:

We can create a function that calls the input with the id and takes a parameter that verifies the function.

Steps:

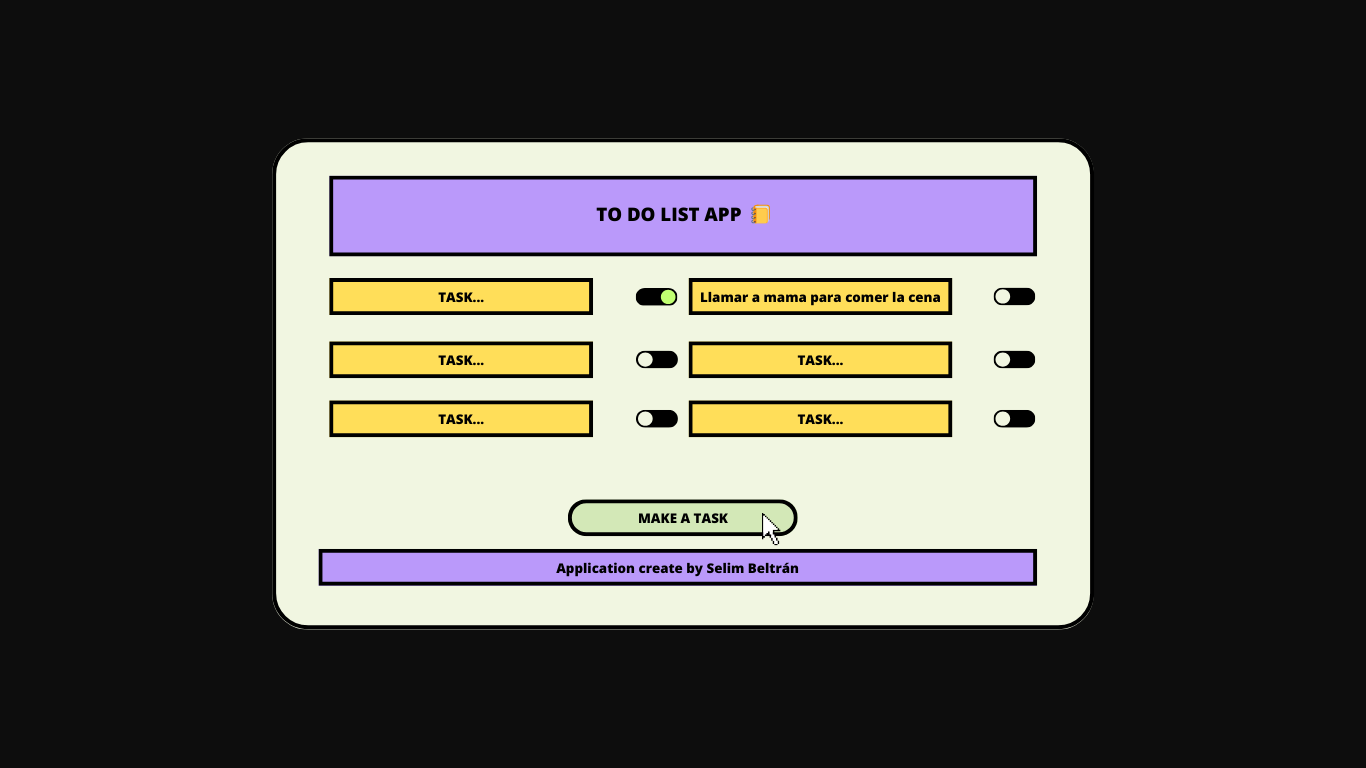
#1: Create a call for each input in JS.

#2: Put the context of the style in JS.

Day #01/3:

The function Checked only works with a Checkbox…

Now:



We need to disable the create task to make sense to the program, so we know a property in HTML that can help us with this problematic.

We are going to use attribute DISABLE = “”, and we are going to call it in JS and invoked when the container five appeared.

To create an overflow program, we’re going to put a global function that disable the button like a conditional and if it is false, then the program flow normal but when it is true this button will be disabled.

Example:

function disableButton(){

if (contenedores[5].style.display == "flex"){

boton\_creador\_tareas.setAttribute("disabled","")

const colorsTask = document.getElementById("creador-tareas");

colorsTask.style.color = "#000000";

colorsTask.style.backgroundColor = "#a8e5c7";

} else {

formEvent();

appearEffect();

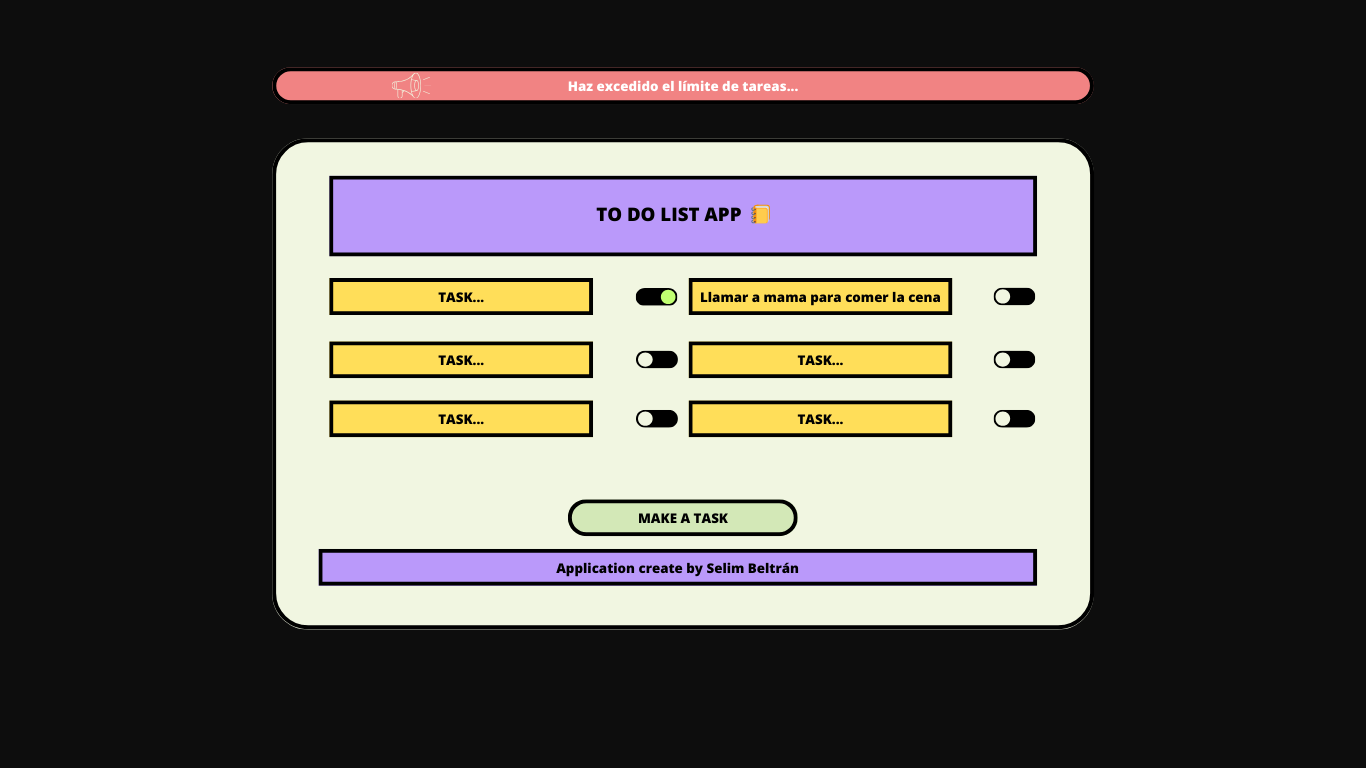
}

We discovered a super important property and it is:

Object.style.cssText: (write like in css);

Day #01/4:

We put the message to the user when he/she puts the button when whole task is created.



Day #01/5:

We create a document explained some properties that is good to the client knows: [document](../../helps/PropertiesWellKnow.docx).

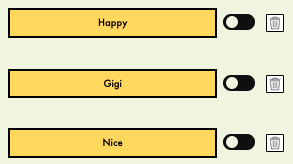
Day #01/6:

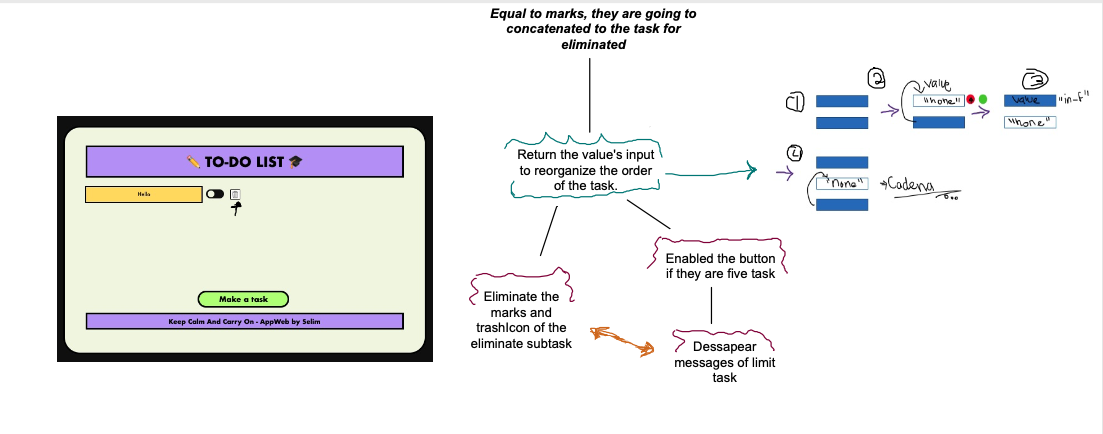
We finish two important processes, the first is the way of the client sees the limited task and the other refreshing the input value to restart the insert text.

Now, we are going to create the “back function” that permits the user eliminate one homework to introduce another. First, we need to make a “eliminate task” with essentials tools.

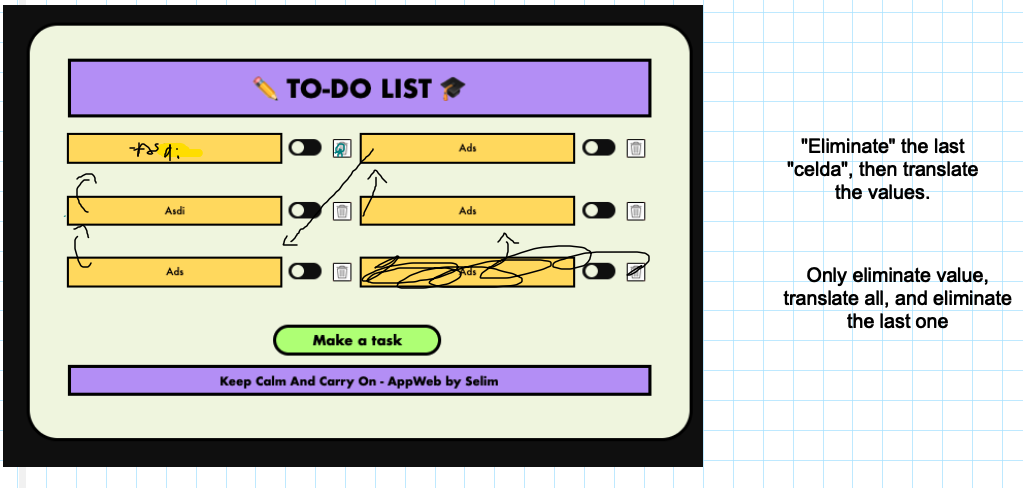
Day #01/7:

We create the icons to eliminate the task that the user doesn’t like… Next, we are going to explain what it will do the button to exterminate some functions.



What is going to do the button?

They are going to be the steps to get a satisfactory performance.



Day #01/8:

Today, we did a lot of things, we transition the task and marks and also, we improve some properties.



We are so quite to finish the project, next, we need to clarify some properties like the underline task and clean some of code.

Day #01/9:

We done the principal part of the project, so we can say that the project is finished, but we are going to implement more functions with JavaScript and they are going to be:

* Refresh the page and stay the data.
* Signup and login.

As you can see, we are going to implement a “server” to stay in touch with the data of the user, but before this GOOD STEP, we are going to stay the files and all data in GTHB.