# **Lab 4: ToDoMVC with Svelte.js**

Prerequisites before starting this lab:

* git pull

Although this was a requirement in previous labs, here is a reminder of the steps to install git:

You can install it from [here](https://git-scm.com/book/en/v2/Getting-Started-Installing-Git).

Let’s check that everything is installed correctly so far. The command to check which version of Git you’re using is the same on both Windows and Mac. To check your Git version, open Command Prompt (Windows), Terminal (Mac), or the Linux terminal.

Once open, run this command:

| git --version |
| --- |

If Git is installed correctly, you should see something like this in your command line:

| git version 2.37.2 |
| --- |

* We are going to need node.js and npm. Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a web browser. NPM – or "Node Package Manager" – is the default package manager for JavaScript's runtime Node.js. You may find detailed instructions to download and install them [here](https://docs.npmjs.com/downloading-and-installing-node-js-and-npm#using-a-node-installer-to-install-nodejs-and-npm).

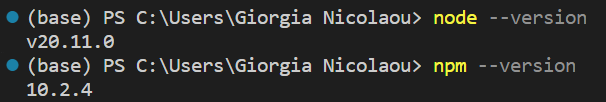
To check the Node version, open a command prompt and type:

| node --version |
| --- |

To check the npm version, type in:

| npm --version |
| --- |

If everything is installed correctly, you should see their respective version numbers below each command, e.g. in my terminal I can see this:



The following versions will be ok to use in the context of this assignment:

1. Node.js 18.0.0 or higher
2. npm 9.0.0 or higher

Today’s objectives are:

* Create a to-do app using Svelte

**Preparation**

Using your terminal, create a folder for lab 4 in your directory on your laptop. You can do this in linux with the command **mkdir folder\_name**

Actionables will be **bolded** to signify steps that you should take, in order to help differentiate between **changes you should make** and examples provided for learning. The version of the final code files are at the end of this doc, *for reference only*.

# **0. Intro to NPM**

npm is the world's largest Software Registry. With over 800,000 code packages, open-source developers use npm to share software.The name npm (Node Package Manager) stems from when npm first was created as a package manager for Node.js, and now all npm packages are defined in files called package.json.

If you run into any problems with npm, be sure to try looking the error up online, posting on edstem, or attending OH :)

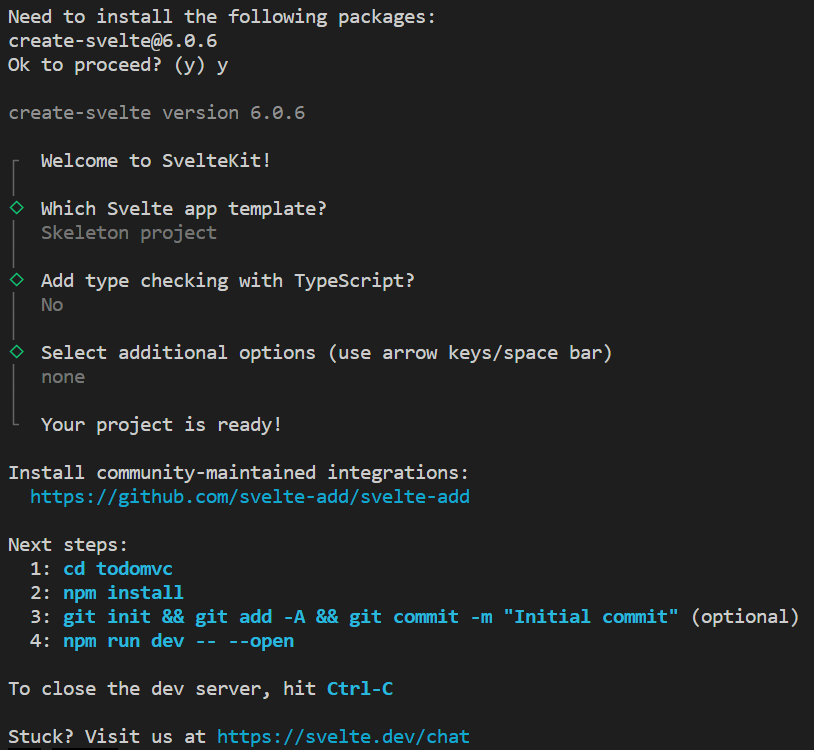
# **1. Intro and starting a Svelte App**

In earlier weeks, we have learned how to use HTML, CSS, and Javascript to build websites from scratch. Now, we will learn how to use Svelte, a framework/compiler that eases the process of building sleek, data-driven apps using the tools we’ve learned.

To get started, we’ll first create a Svelte app through the command line. **Run the following command**:

**npm create svelte@latest todomvc**

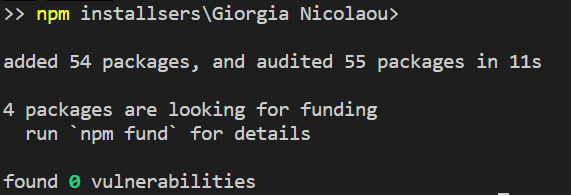
The output should look as follows. Type **y** to confirm installation of any necessary packages. Select **Skeleton project** using your arrow keys for the app template, and hit **Enter/Return**. Select **No** for all of the rest of options given:



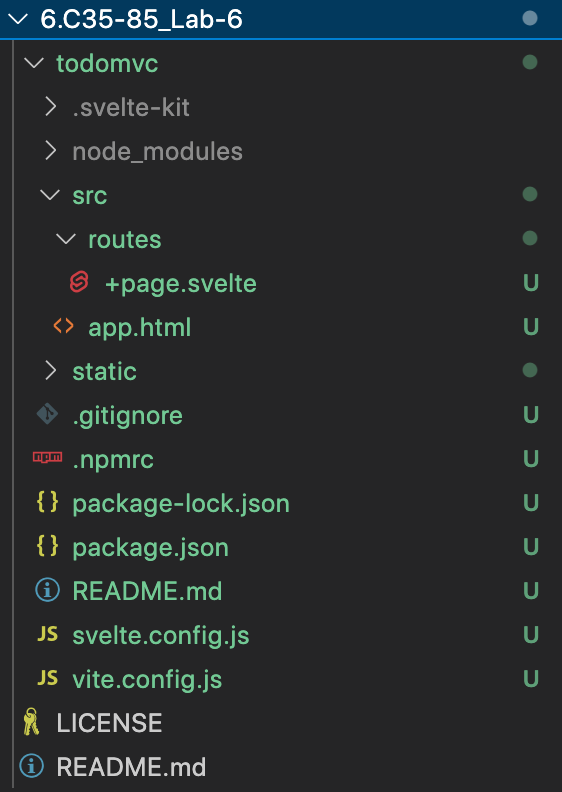
Follow the instructions given under Next steps, and **run the following commands** (example outputs given under each if there are any):

**cd todomvc**

**npm install**

****

After that step, your file structure should look like this:



**npm run dev -- --open**

| > todomvc@0.0.1 dev  > vite dev --open  Forced re-optimization of dependencies  VITE v4.0.4 ready in 1016 ms  ➜ Local: http://localhost:5173/  ➜ Network: use --host to expose  ➜ press h to show help  11:32:11 AM [vite-plugin-svelte] ssr compile done.  package files time avg  todomvc 3 95.7ms 31.9ms | |
| --- | --- |

# **Navigate to** [**http://localhost:5173**](http://localhost:5173) - you should see a basic template application that looks like this:



# **2. Building Components**

First, let’s create a new folder to house our components. Currently, our file structure looks like this:

| todomvc  ├── .svelte-kit  ├── node\_modules  ├── src  │ ├── app.html  │ └── routes  │ └── +page.svelte  ├── static  │ └── favicon.png  ├── .gitignore  ├── .npmrc  ├── package-lock.json  ├── package.json  ├── README.md  ├── svelte.config.js  └── vite.config.js | |
| --- | --- |

**Create a new folder under src called components using vscode, and a file within it called App.svelte**, so the file structure looks like this (edits highlighted):

These are the commands in unix:

**mkdir components**

**cd components**

**touch App.svelte**

| todomvc  ├── .svelte-kit  ├── node\_modules  ├── src  │ ├── app.html  │ ├── components  │ └── App.svelte  │ └── routes  │ └── +page.svelte  ├── static  │ └── favicon.png  ├── .gitignore  ├── .npmrc  ├── package-lock.json  ├── package.json  ├── README.md  ├── svelte.config.js  └── vite.config.js | |
| --- | --- |

## Our home page will be the +page.svelte file under the root route, which is in green above. Let’s **add our new App component** to our home page. Add the following to the +page.svelte file in the routes folder:

| **./todomvc/src/routes/+page.svelte** | | |
| --- | --- | --- |
| <script>  import App from '../components/App.svelte';  </script>  <h1>Welcome to SvelteKit</h1>  <p>Visit <a href="https://kit.svelte.dev">kit.svelte.dev</a> to read the documentation</p>  <App /> | | |

Here, we import our App component by pointing the page to the location within the project, relative to the file.

## **2.1 App.svelte**

Let’s add some elements into the App component. First, **add the basic tags**:

| **./todomvc/src/components/App.svelte** | |
| --- | --- |
| <script>  // Add Javascript here  </script>  <main>  <!-- Add HTML elements here -->  </main>  <style>  main {  // Add CSS styling here, scoped to only work on App.svelte  }  </style> | |

Next, add in some text elements into the App, with some added data within curly brackets. First, **add a name variable to your script tag**. Then, we can **refer to name in the markup**. After you save, you can **check the** [**http://localhost:5173**](http://localhost:5173) **page for updates**. Inside the curly braces, we can put any JavaScript we want. Try changing name to name.toUpperCase() in the curly braces for a shoutier greeting (try to figure that one out yourself!).

| **./todomvc/src/components/App.svelte** | |
| --- | --- |
| <script>  export let name = "World";  </script>  <main>  <h1>Hello {name}!</h1>  <p>Visit <a href="https://svelte.dev/tutorial"> Svelte tutorial</a> to learn how to build Svelte apps.</p>  </main>  <style>  main {  text-align: center;  }  </style> | |

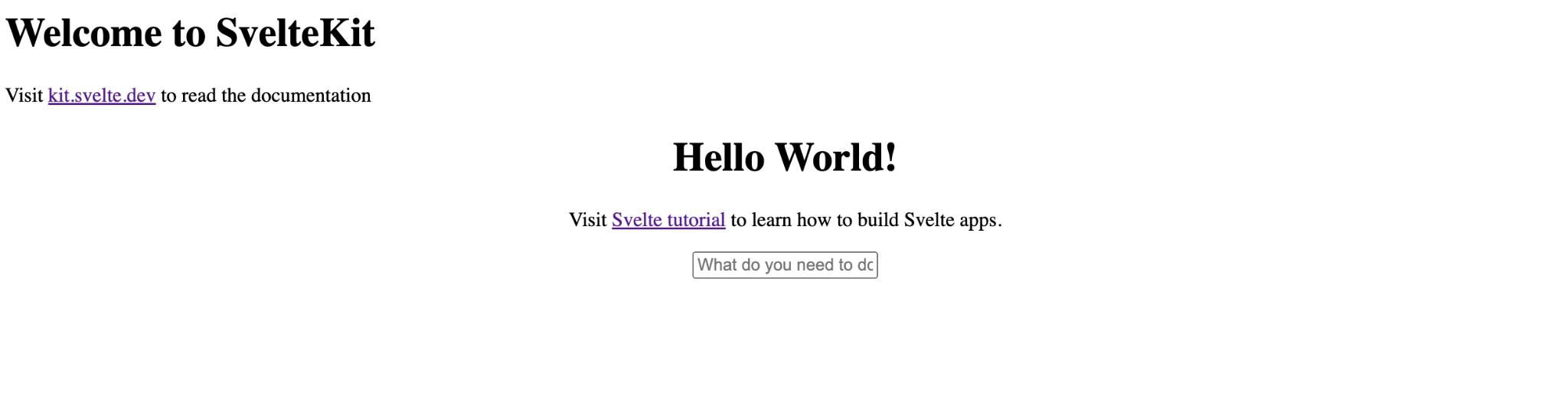
Feel free to play around with making new variables and adjusting them in various ways.

## **2.2 Adding new elements**

Let’s extend our App.svelte with some new features for our to-do app. We can **add an input field**:

| **./todomvc/src/components/App.svelte** | |
| --- | --- |
| <main>  <h1>Hello {name}!</h1>  <p>Visit <a href="https://svelte.dev/tutorial"> Svelte tutorial</a> to learn how to build Svelte apps.</p>  <input placeholder="What do you need to do?" value="" />  </main> | |

If you **check the** [**http://localhost:5173**](http://localhost:5173) **page for updates**, your page should look like:



We can use curly braces to control element attributes, just like we used them to control text. Let’s **adjust our input’s** placeholder value and have it be controlled by a variable in our script tag. We can also **change our header** to better reflect our app’s purpose, and add some more styling.

| **./todomvc/src/components/App.svelte** | |
| --- | --- |
| <script>  let placeholder = "What do you need to do?";  </script>  <main>  <h1>todos</h1>  <input  placeholder={placeholder}  value=""  >  </main>  <style>  @import url('https://fonts.googleapis.com/css2?family=Nunito:wght@300;400;700&display=swap');  main {  text-align: center;  font-family: 'Nunito', sans-serif;  }  h1 {  font-size: 72px;  font-weight: 300;  }  </style> | |

If we check our page again, we’ll see our changes reflected. Notably, our styling on the App.svelte component does not leak out of the component, and the headers and text on the rest of the page remains the same.



## Go ahead and **delete the first two lines (Welcome to SvelteKit** and **Visit kit.svelte.dev…)** - as a reminder, they’re located in +page.svelte. Your +page.svelte file should look like this:

| **./todomvc/src/routes/+page.svelte** | | |
| --- | --- | --- |
| <script>  import App from '../components/App.svelte';  </script>  <App /> | | |

Now, we have a way to write text into the page, but we need to be able to access what the user types in. We can accomplish this by **binding the value to a variable**. This is allowed in Svelte by adding bind: to the element attribute. This means that not only will editing the todo\_text variable edit what is in the input box, but adding and editing the text in the input box will directly change the todo\_text variable.

| **./todomvc/src/components/App.svelte** | | |
| --- | --- | --- |
| <script>  let placeholder = "What do you need to do?";  let todo\_text = "";  </script>  <main>  <h1>todos</h1>  <input  placeholder={placeholder}  bind:value={todo\_text}  >  <p>{todo\_text}</p>  </main> | | |

On your newly created webpage, if we now **add text to the input box**, we see the corresponding text appear below.



## **2.3 Adding functionality**

Another element we can **add is a button** to let Svelte know when to add the to-do. Let’s **add some variables to support storing** our previous to-dos.

| **./todomvc/src/components/App.svelte** | | |
| --- | --- | --- |
| <script>  let placeholder = "What do you need to do?";  let todo\_text = "";  let todos = [  'Learn Svelte',  'Finish Lab'  ];  function add() {  todos = todos.concat(todo\_text);  todo\_text = "";  }  </script>  <main>  <h1>todos</h1>  <input  placeholder={placeholder}  bind:value={todo\_text}  >  <button on:click={add}>  Add new  </button>  <p>{todo\_text}</p>  <p>{todos}</p>  </main> | | |

Finally, let’s improve the general aesthetics of the app. We’ll first **encapsulate the main to-do panel within its own section as a todos class**, which allows us to add styles specifically to it. Next, we will **convert the input-button format of the to-do input area into a form**, whose submit action adds the to-do. This allows us to **remove the “Add new” button**, cleaning up the interface. We **add preventDefault** to stop the default HTML action for a form submit from occurring.

Finally, we **add a lengthy list of styles** to pretty the UI. Add your own colors and styles! We define a set list of variables to encode various colors for the panel - **add in whatever colors you like** and additional styles and see how the app changes! To check if your color contrast is compliant with web accessibility guidelines, **use this tool**: <https://webaim.org/resources/contrastchecker/>.

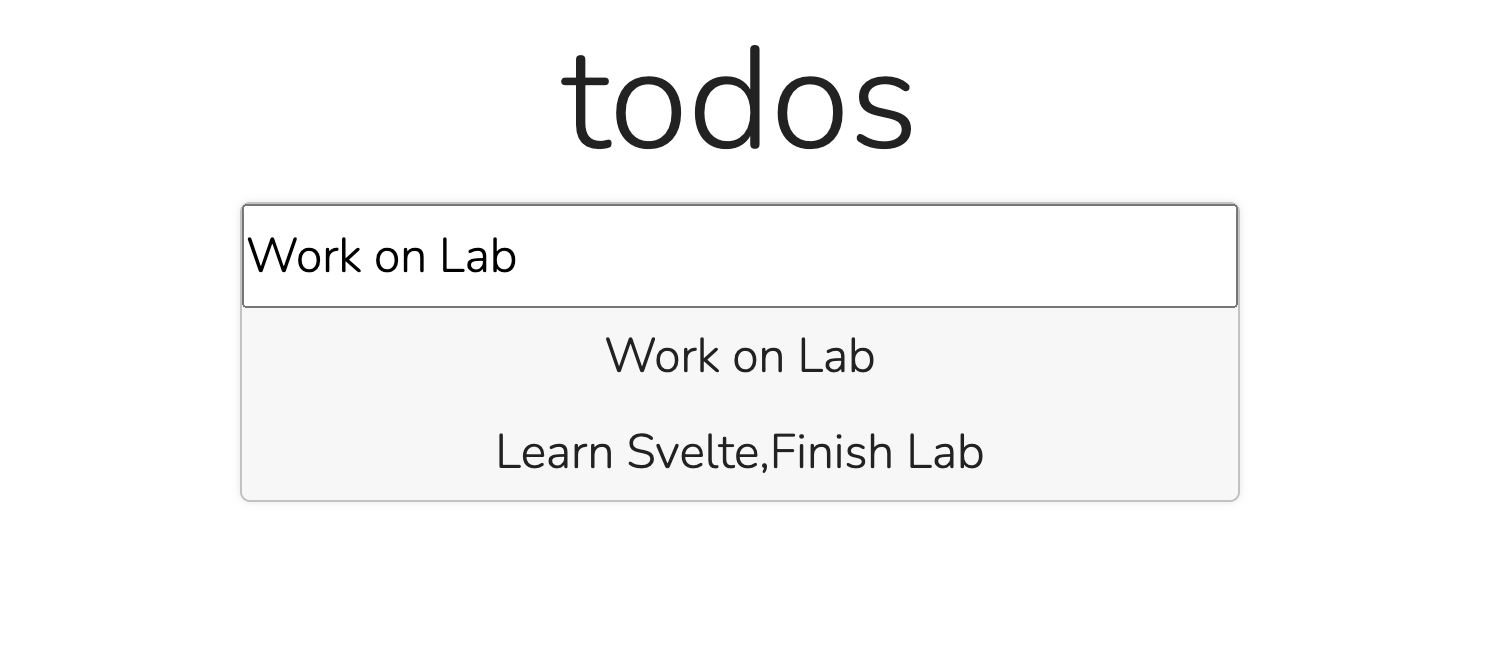
**! To recap the previous two paragraphs, here are your next steps in the following section:**

* Add a todos class
* Remove the button element
* Add a form element to the new todos class, including the attribute of running the add() function on a form submit and preventing the default action, as shown below
* Put the existing input element into the form element
* Copy and paste the entire style section from below, from <style> to </style>, into the bottom of App.svelte
* Under the <style> tag, in the :root {} block, add your own favorite colors to each of the variables

Here is some modifications to App.svelte that implements the above (you have to decide and input your own color scheme):

| **./todomvc/src/components/App.svelte** | | |
| --- | --- | --- |
| <script>  ...  </script>  <main>  <h1>todos</h1>  <section class="todos">  <form on:submit|preventDefault={add}>  <input  placeholder={placeholder}  bind:value={todo\_text}  >  </form>  <p>{todo\_text}</p>  <p>{todos}</p>  </section>  </main>  <style>  @import url('https://fonts.googleapis.com/css2?family=Nunito:wght@300;400;700&display=swap');  :root {  --color-bg: /\* Add a background color for the to-do app! \*/ ex: #f7f7f7;  --color-outline: /\* Add an outline color for the to-do app! \*/ ex: #c2c2c2;  --color-shadow: /\* Add a shadow for the to-do app! \*/ ex: hsl(0, 0%, 0%, 0.1);  --color-text: /\* Add a color for the text! \*/ ex: #222222;  }  \*,  \*::before,  \*::after {  margin: 0;  padding: 0;  box-sizing: border-box;  }  main {  height: 100%;  display: grid;  place-content: center;  text-align: center;  font-family: 'Nunito', sans-serif;  font-weight: 300;  line-height: 2;  font-size: 24px;  color: var(--color-text);  }  input,  button {  font-family: inherit;  font-weight: inherit;  line-height: inherit;  font-size: 24px;  width: 100%;  }  h1 {  font-size: 72px;  font-weight: 300;  line-height: 2;  }  .todos {  width: 500px;  background-color: var(--color-bg);  border-radius: 5px;  border: 1px solid var(--color-outline);  box-shadow: 0 0 4px var(--color-shadow);  }  </style> | | |

Let’s see what the app looks like now! We can enter a new to-do…



And once you hit Enter/Return…

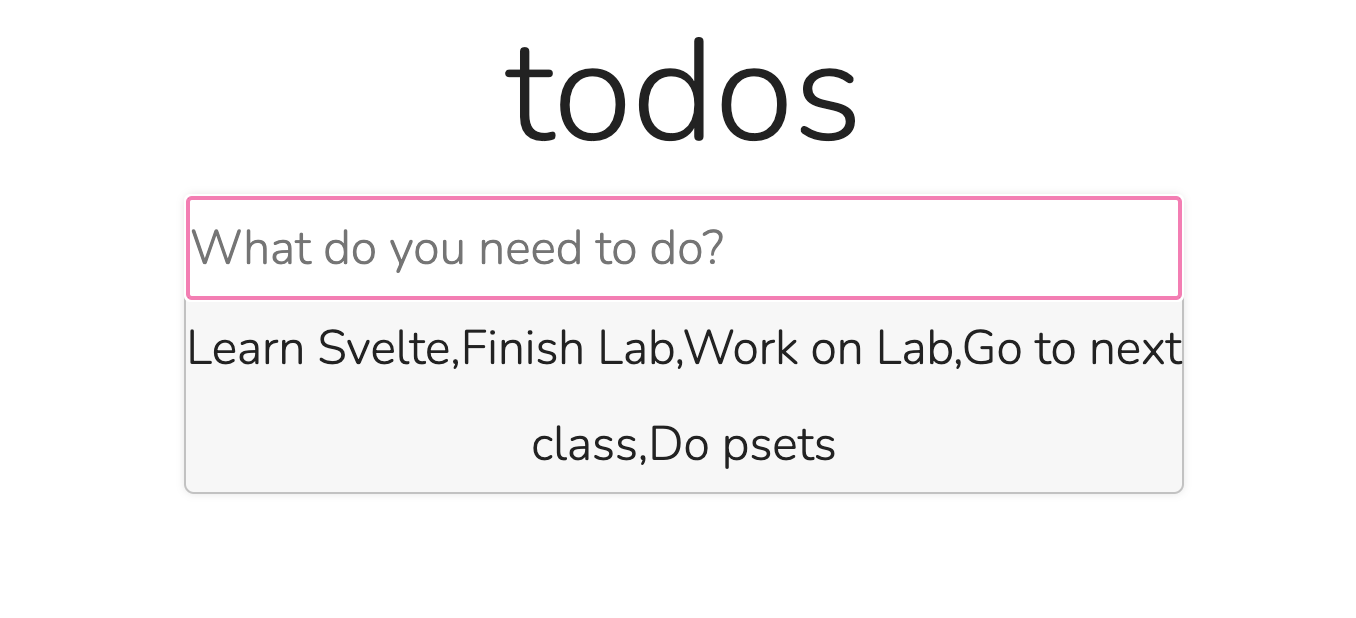


# **3. Unlocking the power of Svelte**

So far, we have worked on elements that we could still write in HTML/CSS/Javascript. What makes Svelte so much more powerful? In this section, we’ll build out the app to fully capitalize on some of Svelte’s highly useful features.

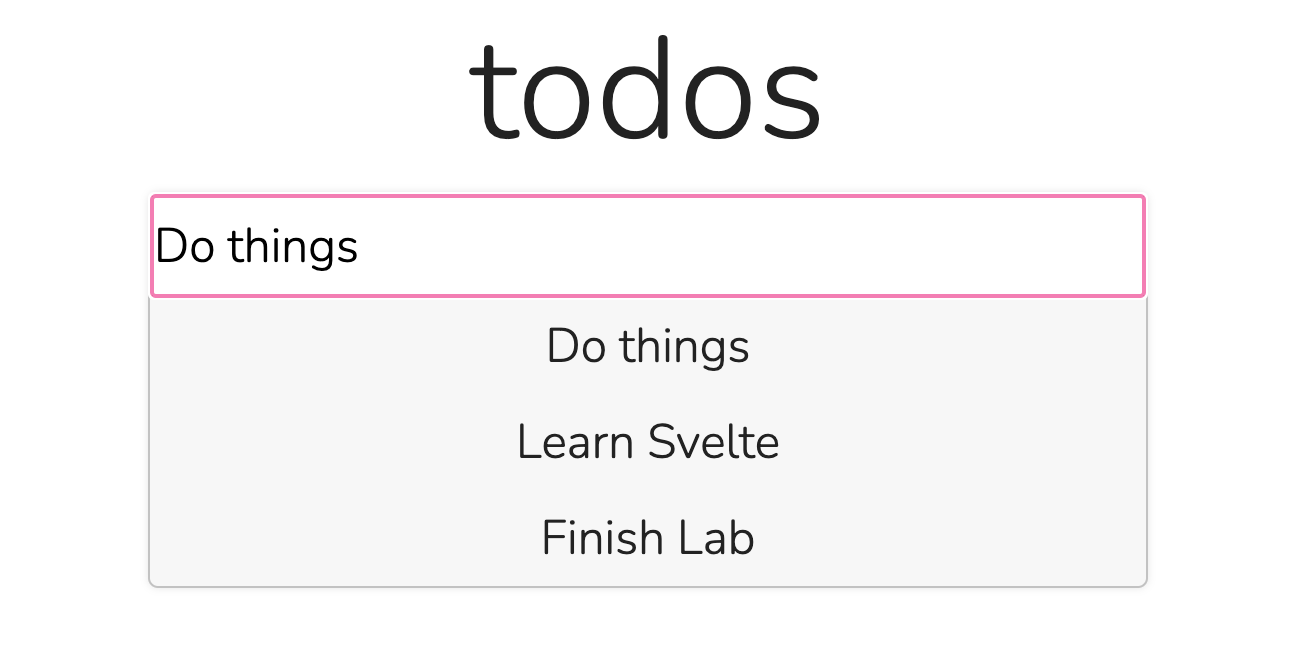
## **3.1 Loops… in HTML**

You may notice that as you add more and more to-dos, our list of to-dos stays on one line and wraps the text around:



We could replace our current to-do paragraph element with multiple paragraph elements:

| **./todomvc/src/components/App.svelte** | | |
| --- | --- | --- |
| ~~<p>{todos}</p>~~  <!-- <p>{todos}</p> -->  <p>{todos[0]}</p>  <p>{todos[1]}</p> | | |



But now, even if we try to submit new to-dos, they disappear since we haven’t explicitly created paragraph elements for them.

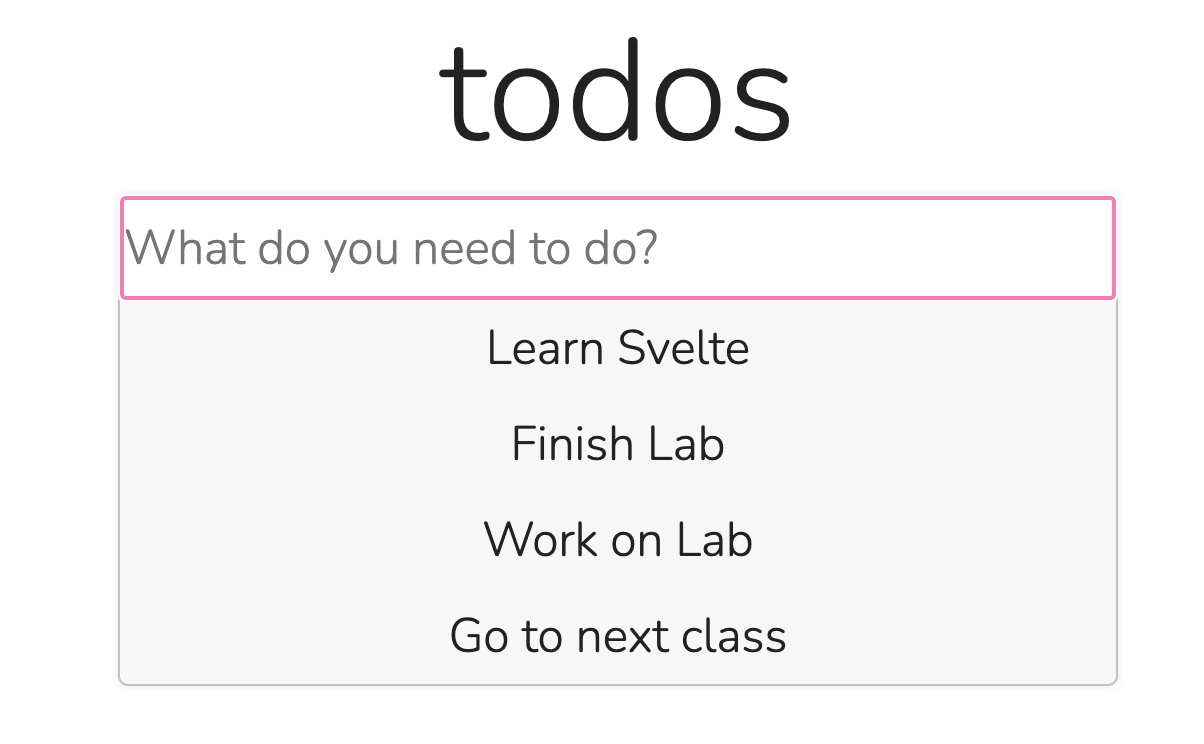
Svelte has a very powerful feature allowing us to create each loops directly in HTML. This way, we can extend our list of to-dos as long as necessary, but be able to loop through it and keep building new paragraph elements for each. This also allows us to remove elements via the list much more easily than with regular Javascript.

An example structure of each loop is as follows. *You don’t have to use this code just yet!* Keep it in mind for the following task.

| **each loop example, basic** | | |
| --- | --- | --- |
| {#each LIST\_NAME as VAR\_NAME}  <ELEMENT\_TYPE>  {VAR\_NAME}  </ELEMENT\_TYPE>  {/each} | | |

Try to **convert the previous list of paragraph to-do elements into an each loop**. Now, when you add new to-dos, the each loop should automatically create new paragraph elements for each to-do.

| **./todomvc/src/components/App.svelte** | | |
| --- | --- | --- |
| ~~<!-- <p>{todos}</p> -->~~  ~~<p>{todos[0]}</p>~~  ~~<p>{todos[1]}</p>~~  {#each todos as todo}  <p>  {todo}  </p>  {/each} | | |



The loop has some additional features, and you could optionally get the current index as a second argument or destructure the variable. More information for this can be found [here](https://svelte.dev/tutorial/each-blocks).

## **3.2 Objects**

We would also like to give each loop a unique identifier keyed to each variable, so that Svelte can better figure out which DOM node to change when the component updates. More information for why this is important can be found [here](https://svelte.dev/tutorial/keyed-each-blocks). Additionally, we would like to keep track of more information for each to-do, such as whether it has been completed or not. To do this, we can **convert each to-do into an object**, with its own ID, text, and status.

| **./todomvc/src/components/App.svelte** | | |
| --- | --- | --- |
| let todos = [  { id: '0', text: 'Learn Svelte', completed: false },  { id: '1', text: 'Finish Lab', completed: false }  ]  let next\_id = 2; | | |

Now, each to-do item in our todo array has the attributes of ID, the text, and the completion status. We can also **set up a next\_id variable** to keep track of what is the next ID available, and increase this by 1 every time we create a new to-do. **Add todo.id as the index** for each loop, and to make sure it’s working, add it to the paragraph element as well.

| **./todomvc/src/components/App.svelte** | | |
| --- | --- | --- |
| {#each todos as todo (todo.id)}  <p>  {todo.id}: {todo.text}  </p>  {/each} | | |

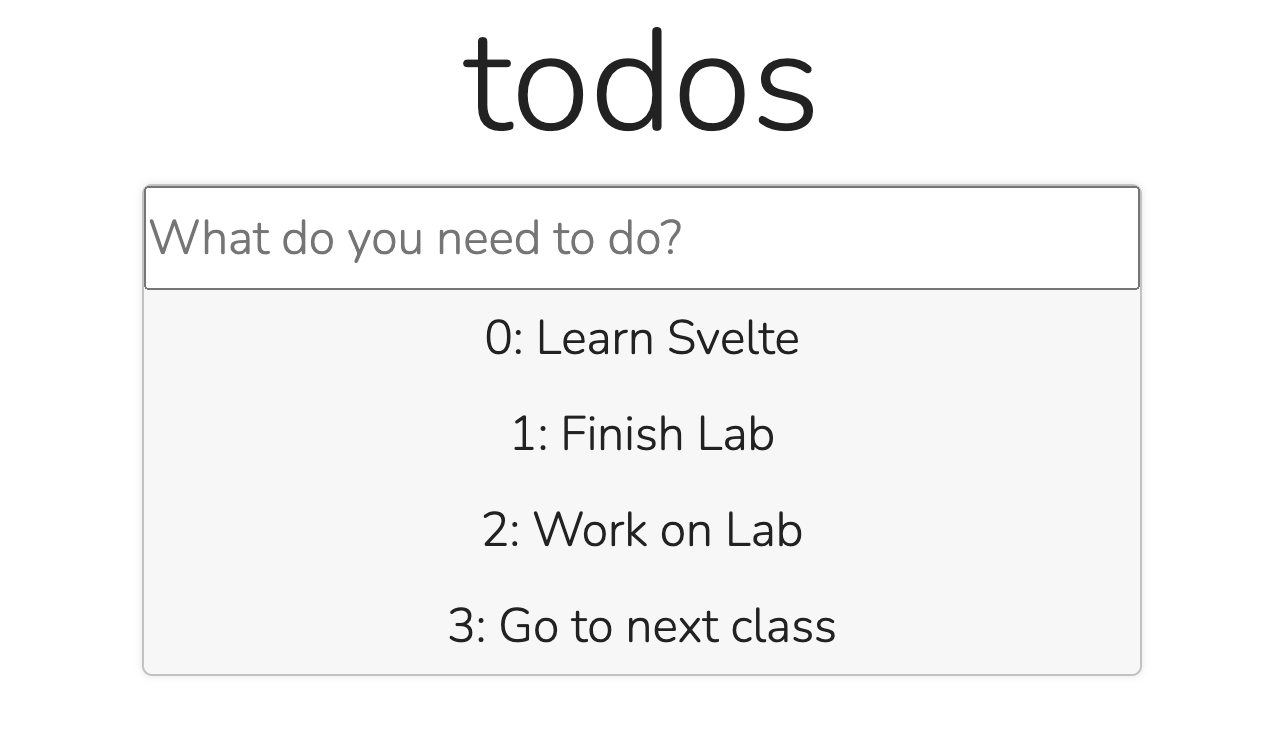
Next, **change the add() function** to reflect the updated to-do object form. The object can be concatenated using the same function as we were using before, but now it should have the following form:

{id: ENTER\_ID, text: ENTER\_TEXT, completed: ENTER\_STATUS}

Make sure to **replace the fields with the correct variables or values**, and **increment the next\_id** **variable** by 1 to move it to the next available ID. Now, when we add new to-dos in the app, we should see the corresponding IDs. Your new script block/add() function should look like:

| **./todomvc/src/components/App.svelte** | | |
| --- | --- | --- |
| <script>  let placeholder = "What do you need to do?";  let todo\_text = "";  let todos = [  { id: "0", text: "Learn Svelte", completed: false },  { id: "1", text: "Finish Lab", completed: false },  ];  let next\_id = 2;  function add() {  todos = todos.concat({  id: next\_id,  text: todo\_text,  completed: false,  });  next\_id = next\_id + 1;  todo\_text = "";  }  </script> | | |

And your web page should look like:



## **3.3 Creating new components**

Wouldn’t it be great if we could edit and check off our to-dos? With Svelte, we could change our current paragraph elements into input elements, so each to-do could be a text box that the user can directly edit to edit to-dos. As an example, that would look like the following code. *You do not have to do anything with this code snippet yet!* Keep it in mind for the following sections.

| **./todomvc/src/components/App.svelte** | | |
| --- | --- | --- |
| {#each todos as todo (todo.id)}  <input placeholder="Deleted todo" bind:value={todo.text} />  {/each} | | |



Although this works, our styling for the main “add to-do” input box carries over to our new to-do input elements. If we wanted to add a checkbox of some kind, we would have to revamp our current component’s CSS completely.

With Svelte, components each restrain styling to just that component, and styles do not leak out. This means that we could turn each to-do into a component and style them however we would like without impacting other components. This would also allow us to build modularity.

**Create a new file under components called ToDo.svelte**

This is the command when you are in the components folder unix:

**touch ToDo.svelte**

Most of this file will be styling for buttons to add potential functionality to our app. **Copy and paste the below code in ToDo.svelte**. Like before, **go ahead and add in your own colors** under the .root{} section of the <style> tag, and see how this changes the colors on the page. Notice how styling on this component does not leak out and impact the styling on the rest of the app.

| **./todomvc/src/components/ToDo.svelte** | | |
| --- | --- | --- |
| <script>  export let todo;  </script>  <main class="todo">  <div class="todo-item">  <input  on:change={() => completeTodo(todo.id)}  bind:checked={todo.completed}  id="todo"  type="checkbox"  class="checkbox-round"  />  <span class:completed={todo.completed}>  <input  placeholder="Deleted todo"  bind:value={todo.text}  type="text"  disabled={todo.completed}  />  </span>  <button  aria-label="Remove todo"  on:click={() => removeTodo(todo.id)}  class="remove"  />  </div>  </main>  <style>  :root {  --x-color: /\* changes the color of the remove button, ex rgb(103, 103, 103); \*/  --x-highlight: /\* changes the color of the remove button when hovered, ex rgb(108, 38, 38); \*/  --checkbox-color: /\* changes the color of the checkbox, ex rgb(108, 38, 38); \*/  --font-size: /\* changes the height of the line and general spacing, ex 24px; \*/  --line-height: calc(var(--font-size) \* 3); /\* dependent on the font-size \*/  }  </style> | | |

Note the flow of information from App.svelte to ToDo.svelte. In ToDo.svelte, we have in the script a placeholder for a to-do object, aptly named todo. We assume within the main area for elements that this variable will have all the expected attributes, specifically id, completed, and text. Additionally, Svelte uses the export keyword in a unique way here, to signify that we want to pass data from one component down to its children.

Finally, we **put in placeholders for potential functions that we may add in** - specifically, removeTodo. We’ll build out these functions in a moment, but for now, we need to put this new ToDo component to use.

Within App.svelte, **import in ToDo.svelte**.

| **./todomvc/src/components/App.svelte** | | |
| --- | --- | --- |
| <script>  import ToDo from '../components/ToDo.svelte';  let placeholder = ...;  let todo\_text = ...;  let todos = [...];  let next\_id = ...;  function add() {  todos = todos.concat({  id: next\_id,  text: todo\_text,  completed: false,  });  next\_id = next\_id + 1;  todo\_text = "";  }  </script>  <main>  <h1>todos</h1>  <section class="todos">  <form on:submit|preventDefault={add}>  <input {placeholder} bind:value={todo\_text} />  </form>  {#each todos as todo (todo.id)}  <ToDo bind:todo={todo} />  {/each}  </section>  </main>  <style>  ...  </style> | | |

We have a very similar loop here, but we build ToDo components for each item in the todo list. Now, if we **refresh our** [**http://localhost:5173**](http://localhost:5173) **page** and take another look, we can see our new and improved version of a ToDo app (this is just an example, yours will be unique based on the selections you chose in your code):



## **3.4 Handling events**

Finally, let’s handle a removeTodo event. We want a button click of the × symbol in a ToDo component to remove the corresponding item from the todo list in the App component. **Make the following edits in the files:**

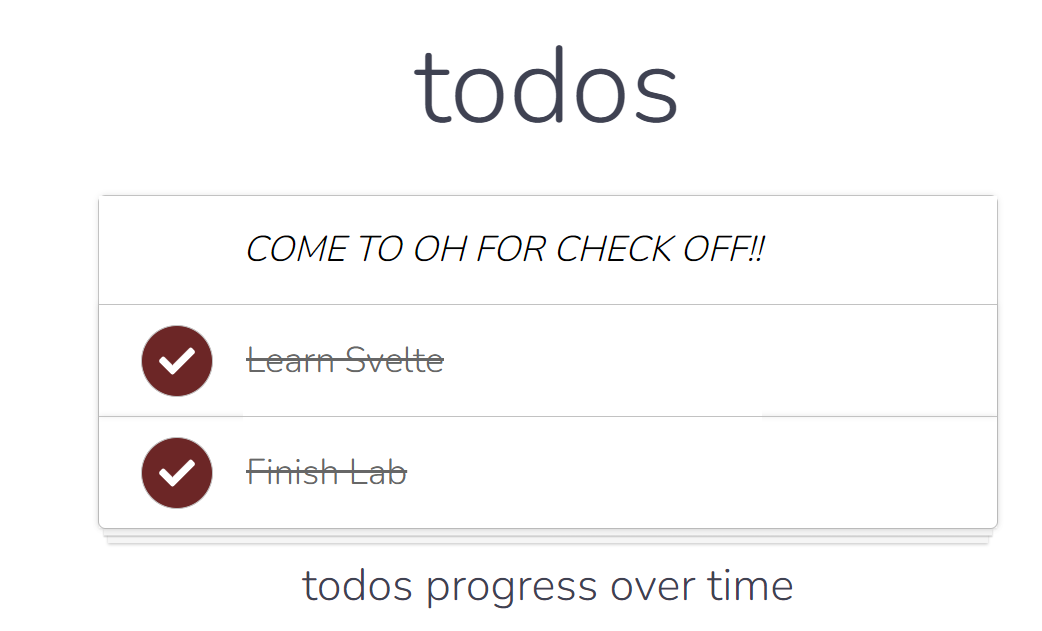
| **./todomvc/src/components/ToDo.svelte** | | |
| --- | --- | --- |
| <script>  export let todo;  export let removeTodo;  </script> | | |

That links our removeTodo function to a variable that we can propagate out using the export keyword to App.svelte.

| **./todomvc/src/components/App.svelte** | | |
| --- | --- | --- |
| <script>  ...  function removeTodo(id) {  todos = todos.filter((todo) => todo.id !== id);  }  </script>  <main>  ...  {#each todos as todo (todo.id)}  <ToDo bind:todo={todo} {removeTodo}/>  {/each}  ...  </main> | | |

Now, when we click the remove button for any to-do item, it gets removed from our list! Congratulations, you can now remove “Finish Lab” 🙂

Below is an example of what your app would look like with various customizations in your code. During check off, show us your fun customizations!



# **Checkoff**

Have your website loaded on your web browser and be ready to show it to your TA. We will check for the following:

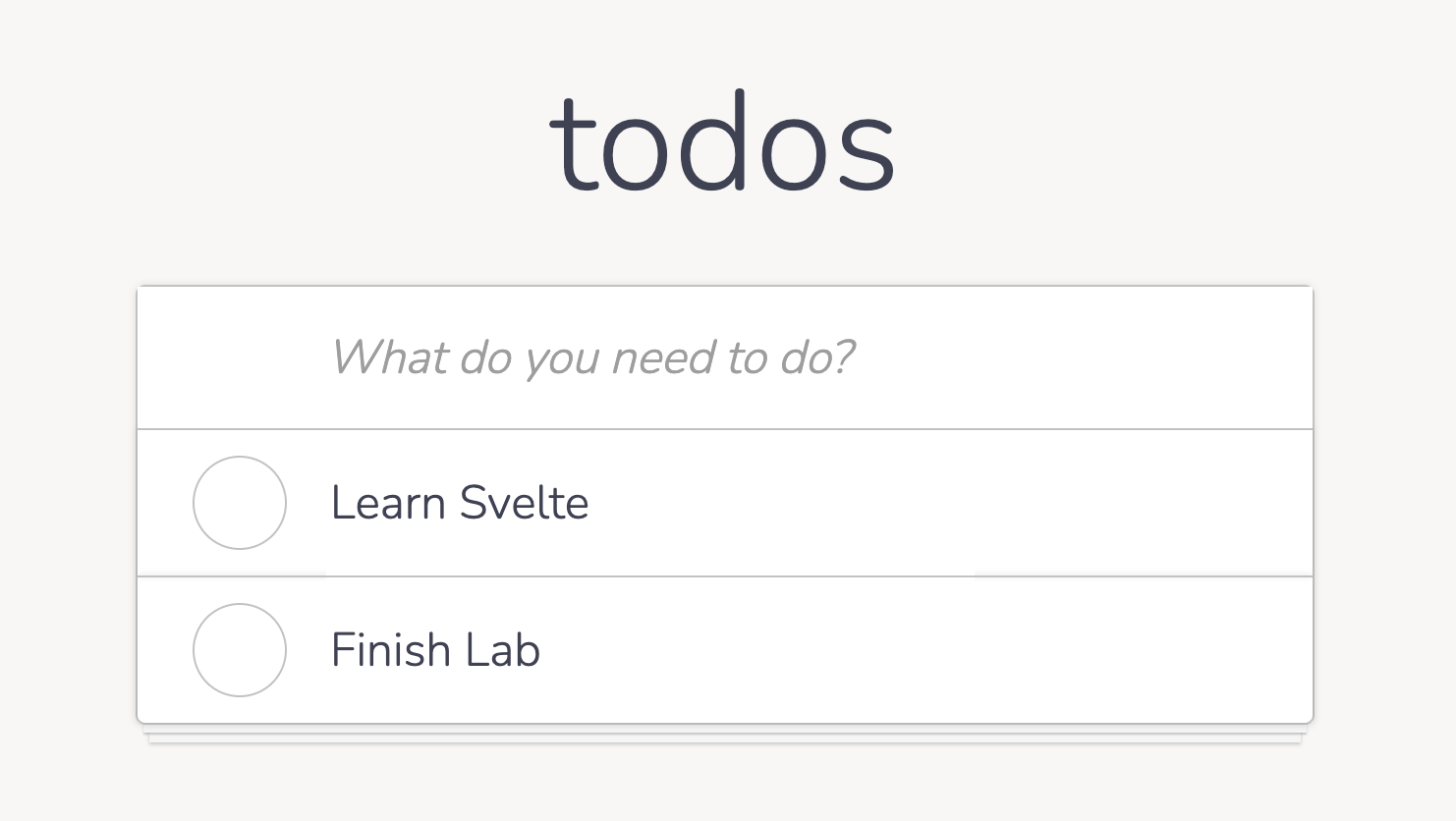
* Add todos
  + Can add new to-dos that immediately (reactively) appear on the list
* Edit todos
  + To-dos are editable, and edits permanently remain
* Remove todos
  + Remove button removes to-dos
* Colors and formatting
  + Elements have student’s unique colors and formatting looks correct

# **4. Bonus for fun 😄**

Here are some extra things you could work on to build out your new todo app!

## **4.1 Extra formatting ideas**

* Add a background color to the whole app
* Add a fun border to the app
* Play around with centering



# **5. Final Code**

Here are the TA’s final App.svelte and ToDo.svelte files! Feel free to use these as reference if you get stuck at any point, but **please do not copy and paste the whole file in for your checkoff** - TAs will check for specific comments and elements that were not included here if there are too many similarities.

## **5.1 App.svelte**

| **./todomvc/src/components/App.svelte** | | |
| --- | --- | --- |
| <script>  import ToDo from '../components/ToDo.svelte';  let placeholder = "What do you need to do?";  let todo\_text = "";  let todos = [  { id: "0", text: "Learn Svelte", completed: false },  { id: "1", text: "Finish Lab", completed: false },  ];  let next\_id = 2;  function add() {  todos = todos.concat({  id: next\_id,  text: todo\_text,  completed: false,  });  next\_id = next\_id + 1;  todo\_text = "";  }  function removeTodo(id) {  todos = todos.filter((todo) => todo.id !== id);  }  </script>  <main>  <h1>todos</h1>  <section class="todos">  <form on:submit|preventDefault={add}>  <input {placeholder} bind:value={todo\_text} />  </form>  {#each todos as todo (todo.id)}  <ToDo bind:todo={todo} {removeTodo}/>  {/each}  <div class="actions" />  </section>  <h2 style="margin-top: 15px">todos progress over time</h2>  </main>  <style>  @import url("https://fonts.googleapis.com/css2?family=Nunito:wght@300;400;700&display=swap");  :root {  --color-bg: #ffffff;  --color-outline: #c2c2c2;  --color-shadow: hsl(0, 0%, 0%, 0.1);  --color-text: #3f4252;  --color-bg-1: hsla(0, 0%, 0%, 0.2);  --color-shadow-1: hsl(0, 0%, 96%);  }  \*,  \*::before,  \*::after {  margin: 0;  padding: 0;  box-sizing: border-box;  }  main {  height: 100%;  display: grid;  place-content: center;  text-align: center;  font-family: "Nunito", sans-serif;  font-weight: 300;  line-height: 2;  font-size: 24px;  color: var(--color-text);  }  label,  input,  button {  font-family: inherit;  font-weight: inherit;  line-height: inherit;  font-size: 24px;  width: 100%;  }  input {  padding-left: 96px;  line-height: 72px;  font-style: italic;  border: none;  }  ::placeholder {  color: #9e9e9e;  }  h1 {  font-size: 72px;  font-weight: 300;  line-height: 2;  }  h2 {  font-size: 30px;  font-weight: 300;  line-height: 1.5;  }  .todos {  width: 600px;  background-color: var(--color-bg);  border-radius: 5px;  border: 1px solid var(--color-outline);  box-shadow: 0 0 4px var(--color-shadow);  }  .actions {  position: relative;  display: flex;  align-items: center;  justify-content: space-between;  }  .actions:before {  content: "";  height: 40px;  position: absolute;  right: 0;  bottom: 0;  left: 0;  box-shadow: 0 1px 1px var(--color-shadow-1), 0 8px 0 -3px var(--color-shadow-1),  0 9px 1px -3px var(--color-bg-1), 0 16px 0 -6px var(--color-shadow-1),  0 17px 2px -6px var(--color-bg-1);  z-index: -1;  }  </style> | | |

## **5.2 ToDo.svelte**

| **./todomvc/src/components/ToDo.svelte** | | |
| --- | --- | --- |
| <script>  export let todo;  export let removeTodo;  </script>  <main class="todo">  <div class="todo-item">  <input  on:change={() => completeTodo(todo.id)}  bind:checked={todo.completed}  id="todo"  type="checkbox"  class="done"  />  <span class:completed={todo.completed}>  <input  placeholder="Deleted todo"  bind:value={todo.text}  type="text"  disabled={todo.completed}  />  </span>  <button  aria-label="Remove todo"  on:click={() => removeTodo(todo.id)}  class="remove"  />  </div>  </main>  <style>  /\*\*  Define basic variables for use within component  \*\*/  :root {  --x-color: rgb(103, 103, 103);  --x-highlight: rgb(108, 38, 38);  --checkbox-color: rgb(108, 38, 38);  --font-size: 24px;  --line-height: calc(var(--font-size) \* 3);  }  /\*\*  Styles for entire todo component, to make sure everything stays inline  \*\*/  .todo {  border-top: 1px solid var(--color-outline);  box-shadow: 0 0 4px var(--color-shadow);  }  .todo-item {  position: relative;  display: flex;  align-items: center;  }  /\*\*  Checkbox  \*\*/  .done {  width: calc(var(--font-size) \* 2);  height: calc(var(--font-size) \* 2);  background-color: white;  border-radius: 50%;  border: 1px solid var(--color-outline);  appearance: none;  -webkit-appearance: none;  outline: none;  cursor: pointer;  position: absolute;  left: var(--font-size);  }  .done:checked {  background-image: url("data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHZpZXdCb3g9IjAgMCA1MTIgNTEyIj48cGF0aCBmaWxsPSIjZmZmZmZmIiBkPSJNMTczLjg5OCA0MzkuNDA0bC0xNjYuNC0xNjYuNGMtOS45OTctOS45OTctOS45OTctMjYuMjA2IDAtMzYuMjA0bDM2LjIwMy0zNi4yMDRjOS45OTctOS45OTggMjYuMjA3LTkuOTk4IDM2LjIwNCAwTDE5MiAzMTIuNjkgNDMyLjA5NSA3Mi41OTZjOS45OTctOS45OTcgMjYuMjA3LTkuOTk3IDM2LjIwNCAwbDM2LjIwMyAzNi4yMDRjOS45OTcgOS45OTcgOS45OTcgMjYuMjA2IDAgMzYuMjA0bC0yOTQuNCAyOTQuNDAxYy05Ljk5OCA5Ljk5Ny0yNi4yMDcgOS45OTctMzYuMjA0LS4wMDF6Ii8+PC9zdmc+");  background-repeat: no-repeat;  background-size: calc(var(--font-size) \* 1) calc(var(--font-size) \* 1);  background-position: center center;  justify-content: center;  align-items: center;  background-color: var(--checkbox-color);  }  /\*\*  Todo text box  \*\*/  input[type="text"] {  font-family: inherit;  font-weight: 300;  font-size: var(--font-size);  line-height: var(--line-height);  background-color: var(--color-bg);  color: var(--color-text);  border: none;  width: 90%;  margin-left: calc(var(--font-size) \* 4);  }  input[type="text"]:focus {  outline: none;  }  input[type="text"]:disabled {  text-decoration: line-through;  color: var(--x-color);  }  /\*\*  Remove button  \*\*/  .remove {  display: none;  position: absolute;  right: var(--font-size);  background-color: var(--color-bg);  font-size: calc(var(--font-size) \* 2);  color: var(--x-color);  transition: color 0.2s ease-out;  border: none;  }  .remove:hover {  color: var(--x-highlight);  }  .remove:after {  content: "×";  }  .todo:hover .remove {  display: block;  }  </style> | | |