




# Project Todos

In this week's project, it's time to flex your redux muscles and build an application which pulls together all the parts of redux by building a todo app.

Technigo/project-todos

In this week's project, it's time to flex your redux muscles and build an application which pulls together all the parts of redux by building a todo app. In your todo app, you should be able to add tasks, list tasks and

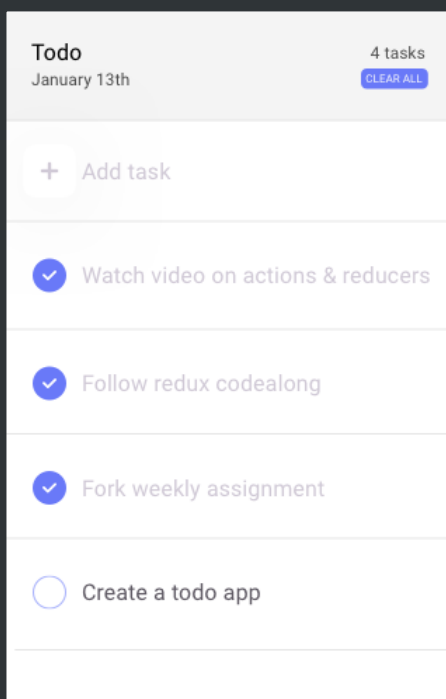
 <https://github.com/Technigo/project-todos>



In your todo app, you should be able to add tasks, list tasks and toggle whether a task is done or not.

You're free to style your todo list however you'd like, but try to keep it simple and clean - remember prospective employers will probably be interested to see this project!

Here's an example of what you could aim for (mobile first design - this project should be a web app, not react native):



## What you will learn 🧠

- How to structure a redux store
- How to write reducers and actions
- How to interact with your store using selectors or by dispatching actions

## How to get started 🦵

1. Fork this repo
2. Clone this repo into your projects folder on your computer
3. Open up VS Code
4. In the terminal, run `cd code` to change into the code folder
5. Install the dependencies needed for react by running `npm install`
6. Run the react development server by running `npm start`

## Hints and tips to complete the project 🧐

As always, start by sketching out what your app will look like and how you'd like to use components. Now we're using redux, you can very easily break things into many small, manageable components.

Once you have your sketch, think about how the data in your store should look. What data does a todo task contain? Sketch it out and get it clear in your head. Knowing how your data will look will help a lot when writing the redux actions and then using that data in your components.

Then you're ready to start working on the project. The `code` folder has a copy of the react starter, but none of the redux packages you need are installed yet, so you'll need to do that yourself:

```
npm install @reduxjs/toolkit react-redux
```

When starting to write your code, try to work on the project in small chunks rather than taking on too much at once. For example, you could start by making a new slice to store your todo tasks and use a hard-coded list of tasks to get up-and-running with.

### ▼ Stuck? Click here to see an example

Here is an example slice you could create to store your tasks:

```
// src/reducers/tasks.js import { createSlice } from '@reduxjs/toolkit' export const tasks =
createSlice({ name: 'tasks', initialState: [ { id: 1, text: 'Watch video on actions & reducers',
complete: true }, { id: 2, text: 'Follow redux codealong', complete: true }, { id: 3, text: 'Fork weekly
assignment', complete: true }, { id: 4, text: 'Create a todo app', complete: false }, ] })
```

Once you've made your first slice, use `combineReducers` and `configureStore` to create a store in App.js which can be passed to a `Provider` as shown in the codealong.

▼ **Stuck? Click here to see an example**

Here is how you can set up your store to be passed to the provider:

```
// src/App.js import React from 'react' import { Provider } from 'react-redux' import { combineReducers, configureStore } from '@reduxjs/toolkit' import { tasks } from './reducers/tasks' const reducer = combineReducers({ tasks: tasks.reducer }) const store = configureStore({ reducer }) export const App = () => { return ( <Provider store={store}> Your components can be mounted here, inside the Provider. </Provider> ) }
```

With all that set up, you can now start creating components. For example, you could make a `TaskList` component which uses `useSelector` from the 'react-redux' package to fetch the list of todos and map over them.

Once you have your todos being listed, it's much easier to see actions you start creating working or not! A good next step is to implement some form of `addTask` action in the `tasks` reducer which will `push` a new task into the array of hard coded tasks. You can then set up a form (keep using `useState` to control form elements and then pass the values into your actions on submit).

## How to hand in the code 🎯

- When you're finished with the project, push your code to GitHub with these commands:




```
git add . git commit -m "your commit message" git push origin master
```

- Navigate to your repo and create a Pull Request into the Technigo repo (Add a link to your deployed project.)
- Wait for the code review from your teachers

## How to get help 🆘


Ask for help and share your knowledge about this project with the 'project-todos' tag on [Stack Overflow](#). Talk to your team on Slack and help each other out. Do some research about your problem, you are surely not the first one with this problem, Google is your friend 😊. And you can of course also book a tech call.

## Requirements 📝

Your project should fulfill the  **Blue Level** and all of the **General Requirements**. Use the  **Red Level** and  **Black Level** to push your knowledge to the next level!

### General Requirements

- Contribute by helping others with this project on Stack Overflow.
- If selected; demo your solution for your team.
- Code follows Technigo's code guidelines:

 [Copy of Guidelines for how to write good code](#)

- Publish your site on Netlify.

### **Blue Level (Minimum Requirements)**

- Your app should list all todo tasks - completed or uncompleted

- You should be able to mark an uncompleted task as complete (and change it back to uncomplete)
- You should be able to add and remove tasks
- Your app should show a count of either all todos, or all uncomplete todos (or both)

💡 Make sure you've committed and pushed a version of your project before starting with the intermediary and advanced goals.

### 🔴 Red Level (Intermediary Goals)

- Add a timestamp for each task indicating when it was created. Timestamp should be displayed as formatted date, but stored as raw date. You can either use built in JS functionalities or [moment.js](#)
- Add a **clear all** button to set all tasks to \*done\* status. You could also use this opportunity to make your app look nice when there's no data. See [empty states UX design](#) for some ideas.
- Use styled-components instead of vanilla CSS to do your styling

### ⬛ Black Level (Advanced Goals)

- Add a date input to your new task form to set a due date on a task. It could be required, or optional - it's up to you. You could then display this in the list and style it differently when a task is overdue.
- Add filters to display completed/uncompleted tasks, tasks created after a given date or anything else you consider important.

#### ▼ Hint

Often, when approaching things like this in redux, it's common to use the redux store to save the current filter and dispatch actions to change it. You can then use that state in your selector to decide what todos to return from the selector.

- Create categories/tags for tasks so they can be grouped - for example, 'Housework', 'Shopping', etc.
- Create projects for tasks → A project could be a group of tasks which all need to be completed and when they are completed, the project is marked as complete.

💡 🚀 Don't forget to add, commit and push the changes to GitHub when you're done. 🍷