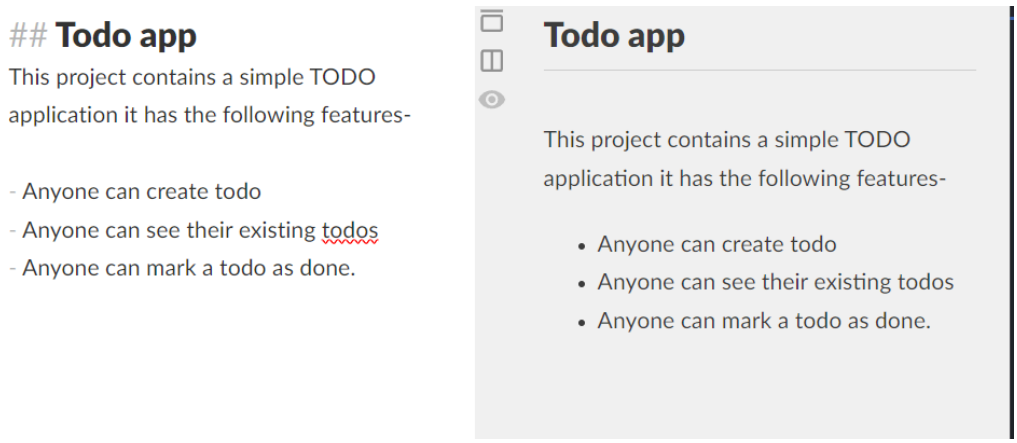
TODO App

Github link:https://github.com/nthapa000/todo-app.git

- **Creating a README.md file:**

md stands for markdown
We will search for a Markdown editor to see how we see things when we write inside the README.md file
(Most project uses this)




```
## Todo app
This project contains a simple TODO application it has the following
features-

- Anyone can create todo
- Anyone can see their existing todos
- Anyone can mark a todo as done.
```

Backend

Building the backend is first priority , if we are creating any project.
Lets create a **backend** folder to make our express application, which will support all of these routes.

We will now first initialize a node project which means put package.json.

Whenever we are creating a backend node.js project this file must be present as this contain lot of information about packages we are using and what all script we have.

npm init -y

OR

npm init

Then it will ask a bunch of questions which we have to answer.

Version will have importance when we deploy npm packages

```
package name: (backend) todo-backend
version: (1.0.0)
description: This is a simple TODO App backend
entry point: (index.js)
test command:
git repository:
keywords:
author: Thapu
license: (ISC)
```

It creates an empty package.json file

```
{
  "name": "todo-backend",
  "version": "1.0.0",
  "description": "This is a simple TODO App backend",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Thapu",
  "license": "ISC"
}
```

npm install express

node_modules folder created

package.json gets updated

```
{
  "name": "todo-backend",
  "version": "1.0.0",
  "description": "This is a simple TODO App backend",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Thapu",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.3"
  }
}
```

Dependencies added , a place where all the external dependencies are listed

npm install jsonwebtoken

There are bunch of other libraries also added not only express, when we open express folder we can see that it also has its own package.json which also has listed lot of dependencies.

More and more application, we have these folders, and we only need these modules if we run the application.

Lets delete these node_modules in our project and lets see what happens

How to bring thse node_modules back either remember it or ,

package.json contains an exhaustive list of everything(dependencies) we added in past

npm install

Hence whenever we share project in github or deploy the project that time we don't share node_modules.

Index.js

Write a basic express boilerplate code

With express.json() middleware

```
// write basic express boilerplate code
// write express.json() middleware
const express = require("express")
const app = express();

app.use(express.json());

app.post("/todo", function (req,res){

})
app.get("/todos",function(req,res){

})
app.post("/completed",function(req,res){

})
```

Now lets Validate with ZOD

```
// data expecting from the user is
// body{
// title:string;
// description:string;
// }
```

Create a **type.js** file , and write all the zod input which we expect from the user.

npm install zod

Write zod schema for this

```
/*
{
    title:string,
    description:string,
}
```

```
{
    id:string
}
*/
```

```
const zod = require("express");

// Write Zod schema for this
/*
{
    title:string,
    description:string,
}
{
    id:string
}
*/

const createTodo = zod.object({
    title:zod.string(),
    description:zod.string()
})

const updateTodo = zod.object({
    id:zod.string()
})

module.exports = {
    createTodo: createTodo,
    updateTodo: updateTodo
}
```

Now lets work on Validation

importing in **index.js** from types.js

```
const {createTodo, updateTodo} = require("./types")
// destructuring of object
```

```
// we can also do it like this
// const types = require("./types")
// then using it like this
// const parsePayload = types.createTodo;
// importing something which has been exported
```

/todo endpoint (Inserting a todo , checking whether the input are valid according to the schema defined)

```
app.post("/todo", function (req,res){
    // validation
    const createPayload = req.body;
    const parsedPayload = createTodo.safeParse(createPayload);
    if(!parsedPayload.success){
        res.status(411).json({
            msg:"You sent the wrong inputs",
        })
        return;
    }
    // put it in mongodb
})
```

/completed endpoint (Updating a todo)

```
app.post("/completed",function(req,res){
    // marking todo as complicated
    const updatePayload = req.body;
    const parsedPayload = updateTodo.safeParse(updatePayload);
    if(!parsedPayload.success){
        res.status(411).json({
            msg:"You sent the wrong todos id",
        })
        return;
    }
})
```

Now we will creating a mongoDB Schema and updating and inserting in MongoDB

npm install mongoose

Mongoose library being used to connect to mongoDB database.

Create db.js

```
// mongoDb schema
/*
    Todo {
        title: string;
        description : string;
        completed : boolean
    }
*/
const mongoose = require("mongoose")
// mongodb+srv://admin:KmCihXn0l1podXRj@cluster0.9gr3ic2.mongodb.net/todos
mongoose.connect("mongodb+srv://admin:KmCihXn0l1podXRj@cluster0.9gr3ic2.mo
ngodb.net/todos")
// we should put it in env file

const todoSchema = mongoose.Schema({
    title: String,
    description: String,
    completed: boolean
})

const todo = mongoose.model('todos',todoSchema);
module.exports ={
    todo
}
```

Now we have to import this schema , insert in mongoDB

index.js after the backend is completed

```
// write basic express boilerplate code
// write express.json() middleware
const express = require("express")
const {createTodo, updateTodo} = require("./types")
const {todo} = require("./db")
// destructuring of object
```

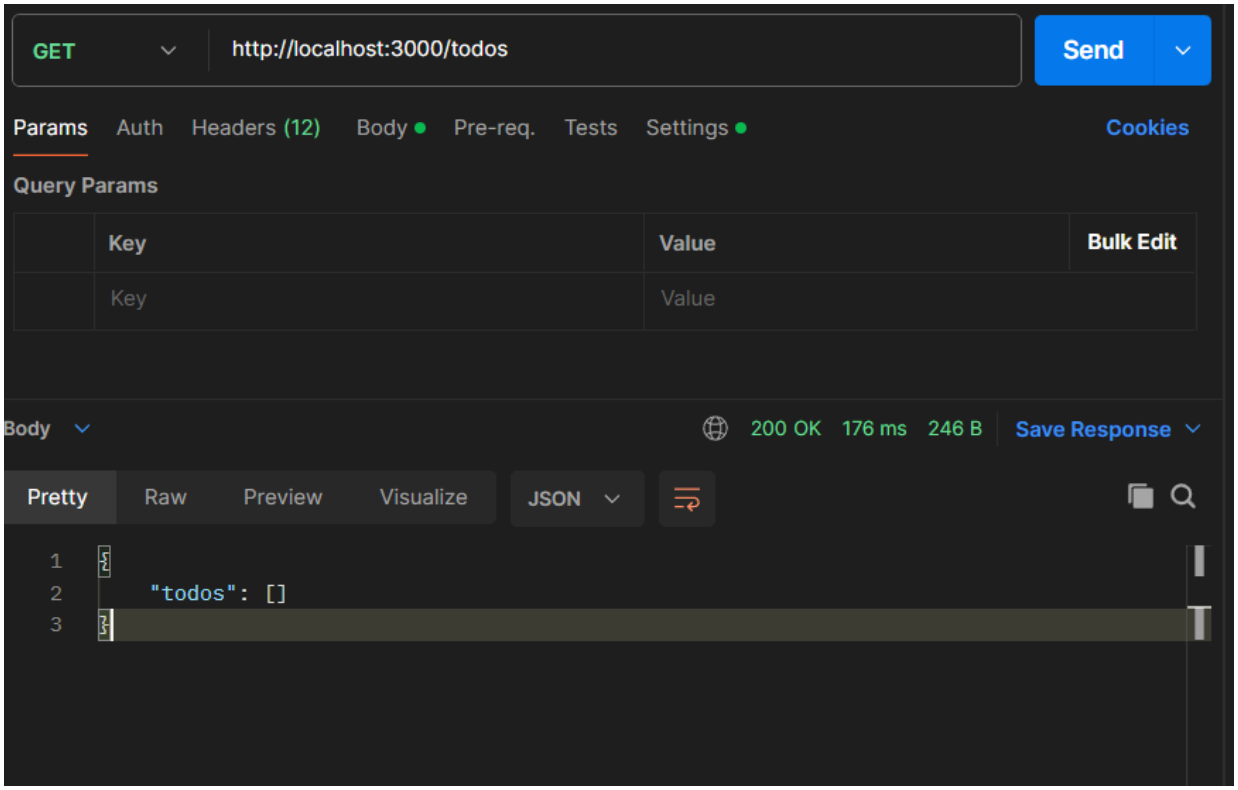
```
// we can also do it like this
// const types = require("./types")
// then using it like this
// const parsePayload = types.createTodo;
// importing something which has been exported
const app = express();

app.use(express.json());

app.post("/todo", async function (req,res){
    // validation
    const createPayload = req.body;
    const parsedPayload = createTodo.safeParse(createPayload);
    if(!parsedPayload.success){
        res.status(411).json({
            msg:"You sent the wrong inputs",
        })
        return;
    }
    // put it in mongodb
    // await syntax only inside async function
    await todo.create({
        title: createPayload.title,
        description: createPayload.description,
        completed:false
    })
    // if we don't await , then it like saying todo is inserted without
todo waiting for database.
    // await for todo (to actually reach database) , then only send the
message
    // if it throws exception in case something happens to database.
    res.json({
        msg:"Todo created"
    })
})

app.get("/todos",async function(req,res){
// getting the todo
const todos =await todo.find({});
// const todos = todo.find({
```

```
// title:"go to gym Boi!!"
// });
// If we want to do specific search
// console.log(todos). this will log as promise
// because todo.find() need to actually hit the database
// it will take long time we have to await for it. If we log right now
what we get is promise which eventually resolve with the data.
res.json({
    todos
})
})
app.post("/completed",async function(req,res){
    // marking todo as complicated
    const updatePayload = req.body;
    const parsedPayload = updateTodo.safeParse(updatePayload);
    if(!parsedPayload.success){
        res.status(411).json({
            msg:"You sent the wrong todos id",
        })
        return;
    }
    await todo.update({
        // automatically generated
        _id: req.body.id
    },{
        // marked the _id todo as completed
        completed:true
    })
    res.json({
        msg:"Todo marked as completed"
    })
})
app.listen(3000);
```

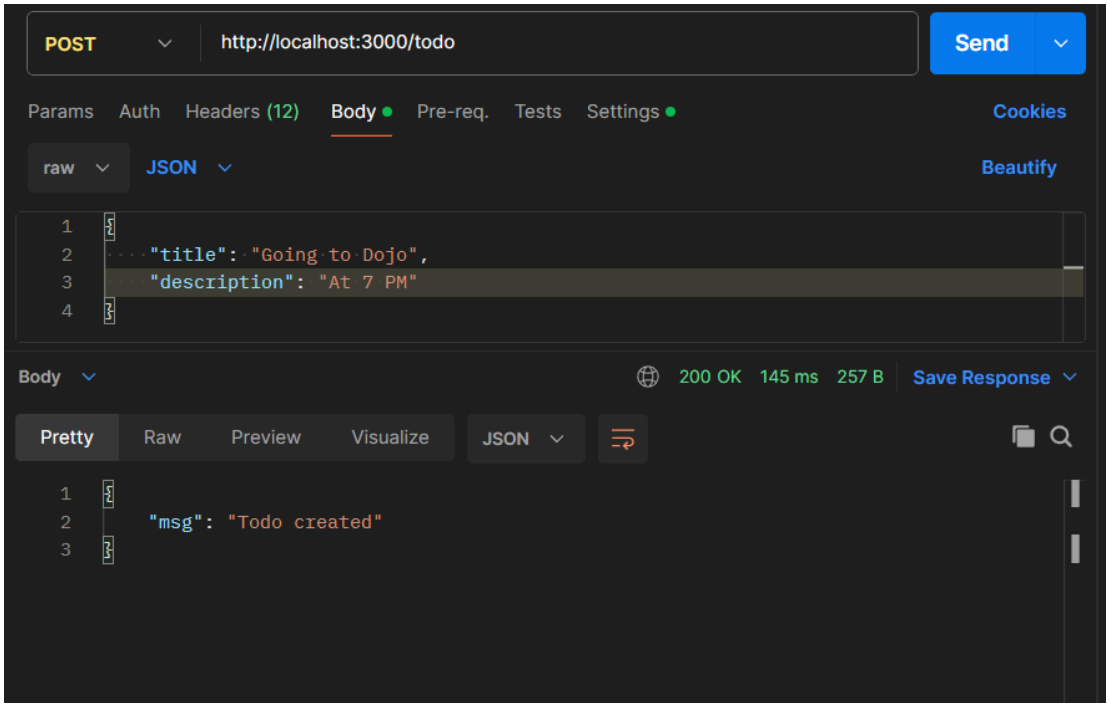
Lets check if all this working properly when we haven't created a frontend is POSTMAN

Intially our todos are empty

Sending the get request :



Lets send a POST request to BE for inserting a todo in database.



Inserting couple of more todos

Q/Na:

- We can make completed as a default false in db schema

```
const todoSchema = mongoose.Schema({
        title: String,
        description: String,
        completed: {
                type: Boolean,
                default: false
        }
})
```

- `await todo updateById(req.body._id, {`
  `completed: true`
  `})`

  It ask for two arguments

  ```
  await todo.update({
          //condition
          _id: req.body.id
  },{
          // make database entry
          completed: true
   })
  ```

- package-lock-json : ensures that all the teammate are using the same version
- “dependencies” dependencies which are required while running the app and “devDependencies” are those dependencies that are required while developing (eq vite (which just like a bundler))

Frontend

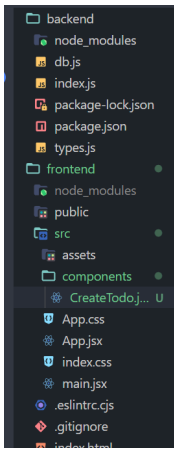
npm create vite@latest

Initialized an empty frontend project

Now we need to create Two section:

1. For rendering the Todos
2. For inserting the Todos

File Structure:



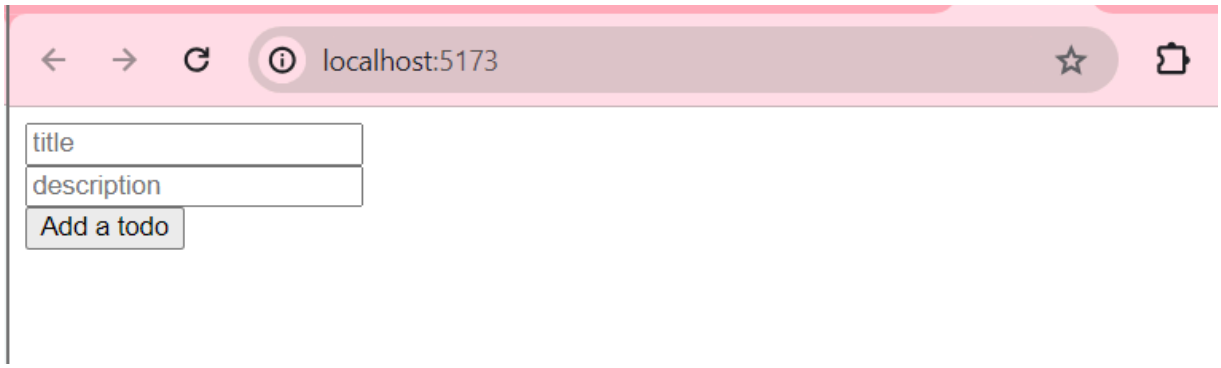
CreateTodo.jsx inside the component folder

```
export function CreateTodo(){
    return <div>
        <input type="text" placeholder="title" />
        <br />
        <input type="text" placeholder="description"/>
        <br />
        <button>
            Add a todo
        </button>
    </div>
}
```

App.jsx

```
import { CreateTodo } from "./components/CreateTodo"

function App() {
  return (
    <div>
```

```
      <CreateTodo></CreateTodo>
    </div>
  )
}

export default App
```



Adding some styles in CreateTodo Component

```
export function CreateTodo(){
    return <div>
        <input style={{
            margin:10,
            padding:10
        }} type="text" placeholder="title" />
        <br /><br />
        <input style={{
            margin:10,
            padding:10
        }} type="text" placeholder="description"/>
        <br /><br />
        <button style={{
            margin:10,
            marginTop:0,
            padding:10
        }}>
            Add a todo
        </button>
        {/* We haven't hit the backend yet */}
```

```
    </div>
}
```

Structure of Todos.jsx

```
export function Todos(){
    return <div>
        <h1>Go to Gym</h1>
        <h2>You need to go to gym</h2>
        <button>Mark as Completed</button>
    </div>
}
```

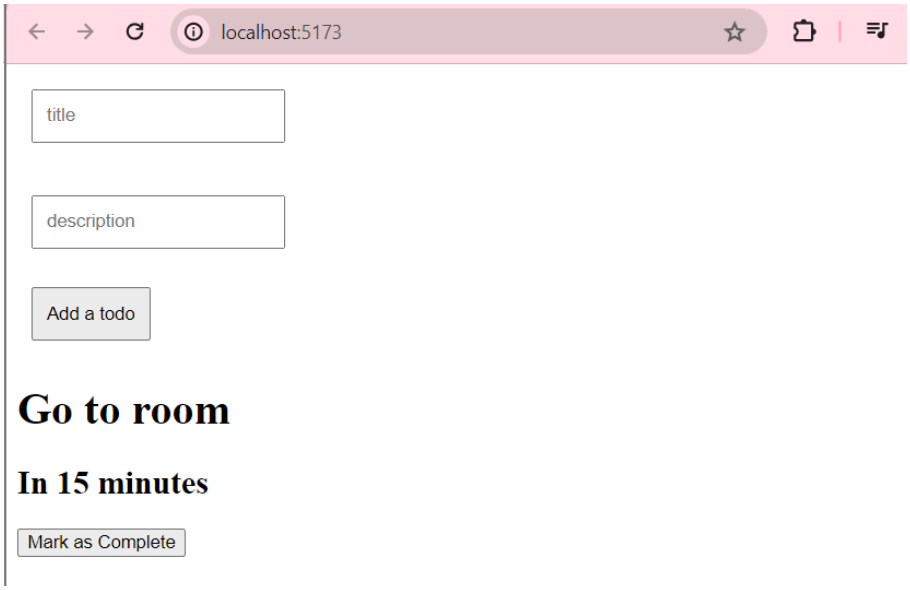
Todos.jsx

```
export function Todos({todos}){
    // object destructuring
    /*
        todos = [
        {
            title: "go to gym",
            description: "We will go to gym"
        }
        ]
    */
    // export function Todos(props){
    // const todos = props.todos;
    return <div>
        {/* given the array how do we render one by one?? */}
        {/* other way
        <div>
            <h1>{todos[0].title}<h1>
            <h2>{todos[0].description}<h2>
            <button>{todos[0].completed == true ? "Completed":"Mark as
Complete"}</button>
        </div>
        <div>
            <h1></h1>
            <h2></h2>
```

```
        </div>
         */}
        {todos.map(function(todo){
            // iterate over the array and map into this div
            return <div>
                {/* we need to have single parent div, sibling not allowed
*/}
                <h1>{todo.title}</h1>
                <h2>{todo.description}</h2>
                <button>{todo.completed == true? "Completed":"Mark as
Complete"}</button>
            </div>
        })}
    </div>
}
```

Hard coded sending the todo data

```
import { CreateTodo } from "./components/CreateTodo"
import { Todos } from "./components/Todos"

function App() {
  return (
    <div>
      <CreateTodo></CreateTodo>
      <Todos todos={[
        {
          title:"Go to room",
          description:"In 15 minutes",
          completed:false
        }
      ]}></Todos>
    </div>
  )
}

export default App
```



Now we have to hit the backend get the current set of todos and pass it to setTodo and update it.
(Now updating the state)

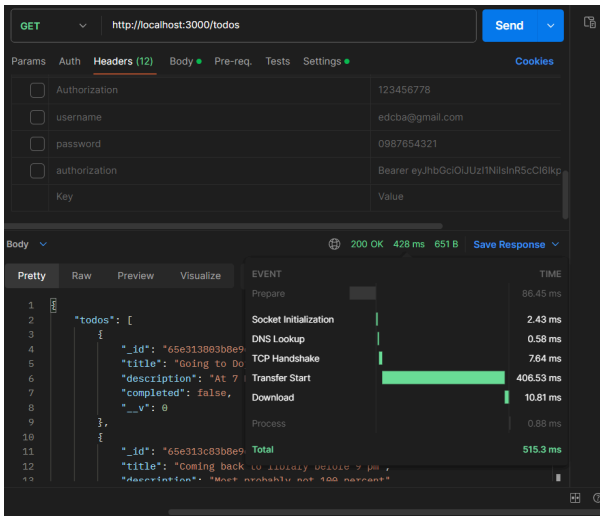
Lets first see an incorrect way of doing it.

App.jsx

```
import { useState } from "react"
import { CreateTodo } from "./components/CreateTodo"
import { Todos } from "./components/Todos"


function App() {
  const [todos,setTodos] = useState([])

  fetch("http://localhost:3000/todos")
  .then(async function(res){
    const json = await res.json();
    setTodos(json.todos);
  })

  return (
```

```
    <div>
      <CreateTodo></CreateTodo>
      <Todos todos={todos}></Todos>
    </div>
  )
}

export default App
```

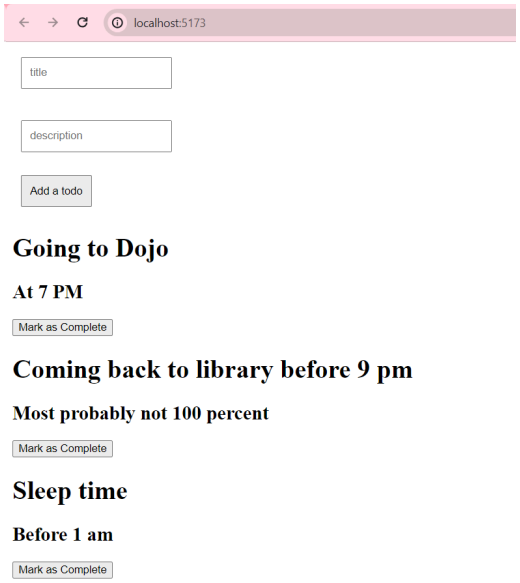
node backend/index.js



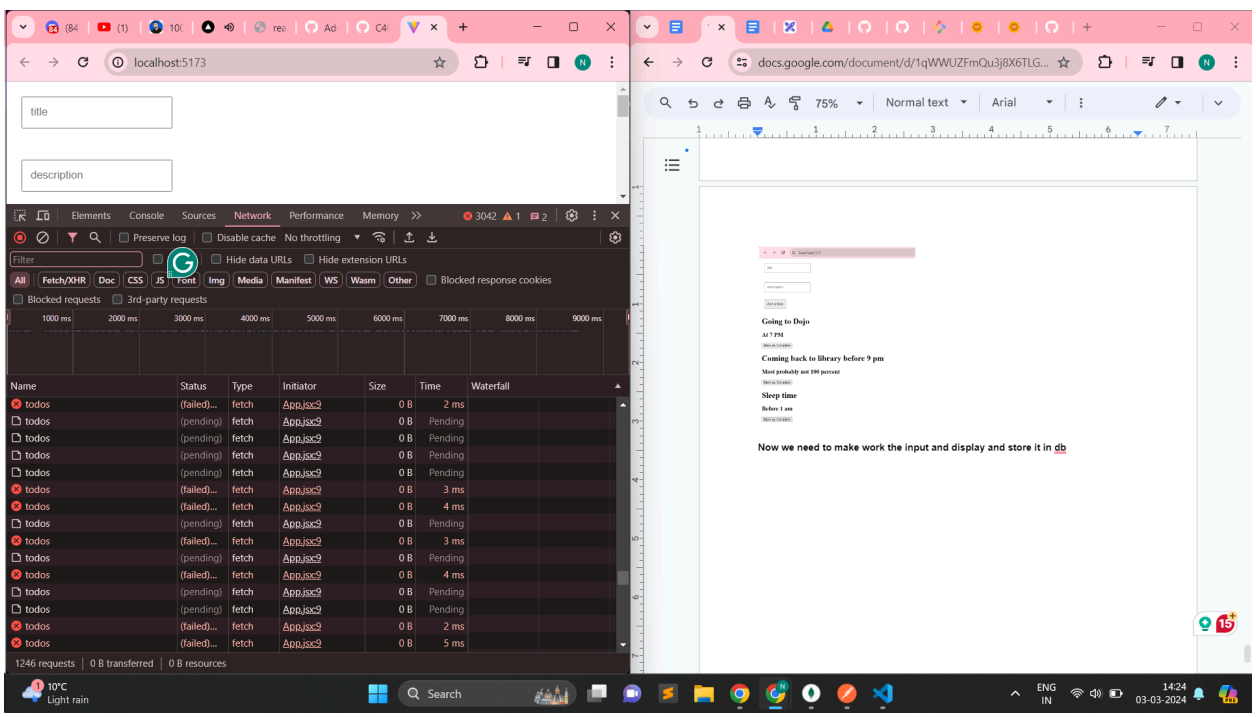
We cannot hit silently localhost:3000 one backend url to frontend url , unless backend allows it ,CORS error

npm install cors (in backend directory)

```
const cors = require("cors")
// anyfrontend can hit it now not only localhost:3000
// we can restrict
// app.use(cors({
//     origin:"http://localhost:5173"
// }))
```



The problem with the above approach is that it is sending request again and again



Why this is bug?

This component renders , fetch request resolves and then we call setTodos , and when we update the function the function re-renders and this is going in a loop.

useEffect hook will resolve our problem

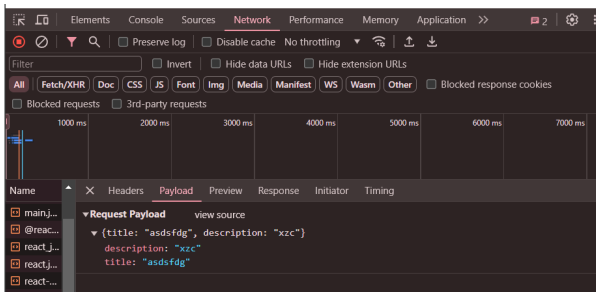
Props should always go from parent to child and not from child to parent (not good practice)

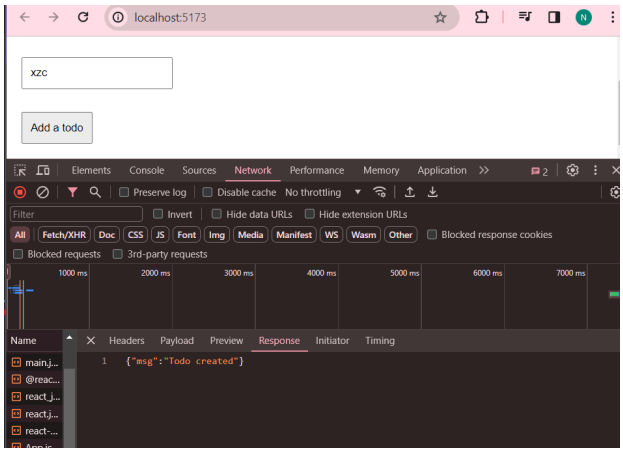
Now we need to make work the input and displaying it.

CreateTodo.jsx

```
import { useState } from "react";

export function CreateTodo(){
    // other-way react-query
    const [title,setTitle]=useState("");
    const [description,setDescription]=useState("");
    // Not optimal as it causes lot of re-renders
    // In react we minimize re-renders

    return <div>
        <input style={{
            margin:10,
            padding:10
        }} type="text" placeholder="title" onChange={function(e){
            const value = e.target.value;
            setTitle(value)
            // e.target is the DOM element
        }} />
        <br /><br />
        <input style={{
            margin:10,
            padding:10
        }} type="text" placeholder="description" onChange={function(e){
            const value = e.target.value;
            setDescription(value)

        }}/>
```

```
        <br /><br />
        <button style={{
            margin:10,
            marginTop:0,
            padding:10
        }} onClick={()=>{
            fetch("http://localhost:3000/todo",{
                method:"POST",
                body: JSON.stringify({
                    // whole point of React was to get away from the
syntax like
                    // document.getElementById("title").innerHTML,
                    // two ways
                    //
                    // react-query(optimal way)
                    title:title,
                    description:description
                }),
                headers:{
                    "Content-type":"application/json"
                }
            })
            .then(async function(res){
                const json = await res.json();
                alert("todo added")
            })
        }}>
            Add a todo
        </button>
        {/* We haven't hit the backend yet */}
    </div>
}
```





We successfully inserted todos from the input and stored it in the database.

Q/Na

1. Dockerize the backend
2. Render network for backend

   Vercel for frontend Deployment
3. Code flow at 1 hour 42 mins
4. lifecycle events in react

   Class-based components
5. Rust we allow them to use multiple cores
6. Adding authorization

```
fetch("http://",{
        method: "POST",
        body: JSON.stringify({
                username: title,
                password: description
```

```
        }),
        headers: {
                "Content-type":"application/json",
                "Authorization":"Bearer "+ localStorage.getItem("token")
        }
})
        .then(async function(res) {
                const json = await res.json();
                localStorage.setItem("token", json.token);
        }
```

Our Todo APP

Backend

Top level readme file
Backend
Express
Zod for Validation
Mongoose for mongo connection
MongoDB as the database
github as the place to push code.

Frontend

2 components
1. create todo
2. render todo

things to fix
1. Infinite request to get todo
2. Not yet implemented update todos