

PROJECT ON TODO APP

*A project report submitted in the partial fulfillment of
Requirements for the award of the Degree of*

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING

Name of the Student: **DAMERLA NAVYA**

Registration Number: **20501A0535**

Name of the College: **Prasad V.Potluri Siddhartha Institute of Technology**

Assignment-1: TODO APP

**Under the Esteemed Guidance of
Mr.Y.Ayyappa M.Tech,(Ph.D),
Assistant Professor,
Department of CSE**



Department of Computer Science and Engineering

PRASAD V POTLURI SIDDHARTHA INSTITUTE OF TECHNOLOGY

(Permanently affiliated to JNTU: Kakinada, Approved by AICTE)

(An NBA & NAAC A+ accredited and ISO 9001:2015 certified Institution)

Kanuru, Vijayawada -520007

2022-2023

**PRASAD V POTLURI SIDDHARTHA INSTITUTE OF
TECHNOLOGY**

Autonomous & Permanent Affiliation to JNTUK-Kakinada, AICTE
approved An NBA & NAAC accredited and ISO 9001:2015 Certified

Institution **KANURU, VIJAYAWADA – 520007**

**DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING**



CERTIFICATE

This is to certify that the project entitled “ **TODO APP**”is submitted by **DAMERLA NAVYA(20501A0535)**,III B.Tech II Semester in partial fulfillment of the requirement for the award of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING** in the academic year 2022-2023.

Signature of the Guide

Mr.Y.Ayyappa M.Tech,(Ph.D),
Assistant Professor,
Dept. of CSE, PVPSIT.

Signature of the HOD

Dr. A. Jayalakshmi,
Professor & HOD,
Dept. of CSE, PVPSIT.

Student Declaration

I, **Damerla Navya**, a student of B.Tech Program, Reg. No. 20501A0535 of the Department of Computer Science and Engineering, Prasad V. Potluri Siddhartha Institute of Technology do hereby declare that I have completed the mandatory mini-project under the guidance of Mr.Y.Ayyappa, Assistant Professor Department of Computer Science and Engineering, PVPSIT.

(Signature and Date)

Acknowledgement

I would like to take this opportunity to thank our beloved Principal, **Dr.K.Sivaji Babu**, for providing a great support for us in completing my project and for giving us the opportunity of doing the project.

At the same time, we feel elated to thank our Professor and Head of Department, **Dr.A.Jayalakshmi**, and for inspiring us all the way and arranging all the facilities and resources needed for this project.

We are also thankful for my project In charge **Y. Ayyappa**, Assistant Professor, Computer Science & Engineering, for their constant encouragement and valuable support throughout the course of the project.

I am very much grateful to all the staff and faculty of Department of CSE for their cooperation during the course of this project work. Finally, I would like to express our sincere thanks to each and every one of our college, who have contributed their help and guidance for the successful completion of this project.

Project Associate
Damerla Navya,
20501A0535

CONTENTS

CHAPTER 1 : OVERVIEW OF THE PROJECT	01
CHAPTER 2 : DISCRIPTION OF THE CODE	02
CHAPTER 3 : CODE OF THE PROJECT	05
CHAPTER 4 : OUTPUT OF THE PROJECT	11

CHAPTER 1 : OVERVIEW OF THE PROJECT

The Project is to build a TODO-LIST app that allows users to create to-do lists and add tasks to those lists. The app allows users to store their lists and task in a database, which enable them to view and manage their lists.

Each list in the app can contain multiple cards, which represent individual tasks or items. Users can add due dates, checklists, and labels to each card to help them organize and prioritize their tasks.

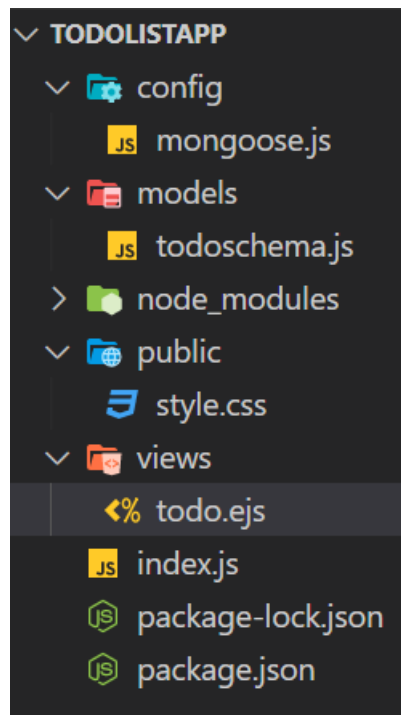
To build this app, I used a combination of technologies, including a database management system like MongoDB, a backend framework like Express and EJS template engine. The app allows users to add To-Do Tasks by filling in a form that requires the user to enter the Task, Category and Due Date. Once the user has entered the details and clicked the “ADD-TASK” button, it saves the data to the MongoDB database. We can observe the tasks that needed to be finished with deadline in mongo campus. Once if the user has finished a specific task, user can select the tasks and press “DELETE-TASK”, then immediately the task will be deleted from the database.

MongoDB is a popular NoSQL document-oriented database that provides a flexible and scalable way to store and retrieve data. MongoDB stores data in JSON-like documents, which allows for greater flexibility in data modeling compared to traditional relational databases. It also provides features such as automatic sharding and high availability, making it a popular choice for building scalable web applications.

Express is a minimalist and flexible web application framework for Node.js that provides a set of tools and utilities for building web applications. It allows developers to create server-side web applications using a variety of HTTP methods and middleware functions. Express is widely used for building APIs, web applications, and mobile applications.

EJS (Embedded JavaScript) is a templating engine for Node.js that allows developers to generate dynamic HTML content based on data from the server. It provides an easy-to-use syntax for embedding JavaScript code into HTML templates, which allows for dynamic content to be generated on the server before it is sent to the client. EJS is widely used in web applications built with Express.

CHAPTER 2: DESCRIPTION OF THE CODE



FILE NAME : index.js

This is a Node.js application using the Express.js framework to create a REST API that communicates with a MongoDB database. It uses the Mongoose library to interact with the database.

The code starts by importing the necessary modules, including Express, Mongoose, and body-parser. The port number and the application instance are also defined.

The schema for the MongoDB collection is imported, and the `express.json()` middleware is used to parse incoming JSON data. The body-parser middleware is also used to parse incoming requests with url-encoded payloads.

The views folder is set as the location for the application's views and the EJS template engine is set as the view engine.

An endpoint is defined to handle GET requests to the root path. This endpoint retrieves all documents in the MongoDB collection using the Mongoose schema and renders them using the 'todo' EJS view.

The application connects to the MongoDB database using the Mongoose `connect()` method and then defines a POST endpoint to handle requests to add new documents to the MongoDB collection. The endpoint uses the Mongoose schema to create a new document and saves it to the database. If the document is saved successfully, the server redirects the client to the root

path.

Another endpoint is defined to handle DELETE requests to delete documents from the MongoDB collection. This endpoint uses the Mongoose deleteOne() method to remove a document from the database by its ID.

Finally, the application listens on the specified port for incoming requests.

FILE NAME : todoschema.js

This code exports a Mongoose schema that defines the structure of a document in a MongoDB collection.

The schema includes three fields: todo, category, and date. Each field is defined as a string type, and the todo and category fields are marked as required using the required option.

The schema is defined using the mongoose.Schema() method, and the fields are defined as properties of the object passed to this method.

The module exports a Mongoose model for the 'data' collection using the mongoose.model() method. This model is created from the schema object and is returned as the module's export.

The model allows the application to interact with the 'data' collection in the database by creating, reading, updating, and deleting documents.

FILE NAME : todo.ejs

This is a web page code for a simple ToDo app using HTML, CSS, and JavaScript. The app allows users to add, list, and delete tasks.

The HTML code contains the basic structure of the web page, including the head and body sections. The head section includes the page title, links to external stylesheets for font and icon libraries, and internal styles for customizing the app's appearance.

The body section consists of a header with the app title, a form for adding new tasks, and a container for displaying existing tasks. The form has three input fields: task name, category, and due date. The task name field is a text input, and the category and due date fields are select and date inputs, respectively. The form also includes a button to submit the new task.

The container for displaying tasks is an unordered list with each task represented as an element. The element displays the task name, category, and due date. Each task also has a delete button represented as an <a> element with a span for the button label.

The code also includes embedded JavaScript code that uses a templating engine (possibly EJS) to dynamically display tasks in the unordered list. The tasks are retrieved from a database and

passed to the template engine as an array of objects. The JavaScript code uses a `forEach` loop to iterate over the array and dynamically create an `` element for each task with the appropriate task data.

FILE NAME : mongoose.js

The code that included in this file uses the Mongoose library to connect to a MongoDB database. Here's what each line of the code does:

- `const mongoose = require('mongoose');` This line imports the Mongoose library and assigns it to the variable `mongoose`. Mongoose is a popular Node.js library that provides a way to interact with MongoDB databases using a schema-based approach.
- `mongoose.connect('mongodb://127.0.0.1/todoDB');` This line connects to a MongoDB database running on the local machine, with the database name `todoDB`. If the database doesn't exist, MongoDB will create it automatically.
- `const db = mongoose.connection;` This line creates a new variable `db` and assigns it to the connection object returned by `mongoose.connection`. This connection object is an instance of `mongoose.Connection`, which represents the connection to the MongoDB database.
- `db.on('error', console.error.bind(console, 'ERROR IN CONNECTION'));` This line sets up an event listener for the error event on the `db` connection object. If there's an error connecting to the database, this listener will log the error message to the console.
- `db.once('open', function() { console.log("DB Successfully Connected"); });` This line sets up an event listener for the open event on the `db` connection object. When the connection is successfully opened, this listener will log a message to the console indicating that the database has been successfully connected.

Overall, The file sets up a connection to a MongoDB database using Mongoose and logs a message to the console indicating whether the connection was successful or not.

FILE NAME : Package.json

This is a package.json file for a Node.js backend application. It contains metadata about the application, such as the application name, version number, description, and author.

The dependencies section lists the packages required by the application, including:

- **Express** : a popular Node.js web framework for building web applications and APIs.
- **Body-parser** : a middleware that extracts the request body and makes it available on req.body.
- **Mongoose** : a library that provides a way to interact with MongoDB databases in Node.js.
- **Nodemon** : a tool that watches for changes in your source code and automatically restarts the server.
- **Ejs** : a template engine that allows you to generate HTML with plain JavaScript.
- **Path** : a Node.js core module that provides utilities for working with file and directory paths.

The scripts section defines several scripts that can be run using `npm run <script-name>`. In this case, there are two scripts defined:

- **Start** : runs the server using the node command.
- **Server** : runs the server using nodemon, which is useful during development as it automatically restarts the server when changes are made to the source code.

CHAPTER 3 : CODE OF THE PROJECT:

FILE NAME:index.js

```
const express = require("express");
const PORT = 8087;

const mongoose = require("./config/mongoose");
const data = require("./models/todoschema");

const app = express();
app.set("view engine", "ejs");
app.use(express.urlencoded("extend:true"));

app.get("/", function (req, res) {
  res.send("I am in Homepage");
});

app.get("/todo", function (req, res) {
  const todos = data.find({}).exec();
  todos
    .then((todo1) => {
      console.log(todo1);
      res.render("todo", { task: todo1 });
    })
    .catch((err) => {
      console.log("Error while fetching data from db");
    });
});

app.post("/add-item", function (req, res) {
  const tododata = new Promise((resolve, reject) => {
    data
      .create({
        description: req.body.description ? req.body.description : "Add a Task",
        category: req.body.category ? req.body.category : "None",
        date: req.body.date ? req.body.date : "No DeadLine",
      })
      .then((newData) => {
        console.log("*** new Data ***");
        resolve(newData);
      })
      .catch((err) => {
        console.log("Error in adding the task" + err);
        reject(err);
      });
  });
  tododata
    .then((newData) => {
      res.redirect("back");
    })
  });
```

```

    })
    .catch((err) => {
        console.log("error", err);
    });
});

app.get("/delete-item", function (req, res) {
    var id = req.query;
    var count = Object.keys(id).length;
    var deletePromises = [];
    for (let i = 0; i < count; i++) {
        deletePromises.push(data.findByIdAndDelete(Object.keys(id)[i]));
    }
    Promise.all(deletePromises)
        .then(() => {
            console.log("task(s) deleted successfully");
            return res.redirect("back");
        })
        .catch((err) => {
            console.log("Error in deleting data", err);
            return res.redirect("back");
        });
});

app.listen(PORT, function (err) {
    if (err) {
        console.log("Error");
        return;
    }
    console.log("Server is running on " + PORT);
});

```

FILE NAME:todo.ejs

```

<html>
<head>

    <script src="https://kit.fontawesome.com/155b8cdebe.js"
crossorigin="anonymous"></script>
    <link rel="stylesheet" href="/public/style.css">
    <style>
    body{
        justify-content: center;
        margin-left: 30%;
        margin-right: 30%;
    }
    input[type=text] {
        width: 100%;
        padding: 12px 10px;
        margin: 8px 0;
        border: none;
        border-bottom: 2px solid black;
    }

```

```

        display: block;
    }
    input[type=date] {
        width: 100%;
        padding: 12px 20px;
        border: none;
        border-radius: 4px;
        background-color: #f1f1f1;
        display: inline-block;

    }
    input[type=checkbox]:checked + label.box{
        text-decoration:line-through;
    }
    .box1{
        text-decoration:line-through;
    }
    .parent{
        display: flex;
    }
    .child{
        margin-left: 20px;
        margin-right: 20px;
    }
    .button1 {
        background-color:#121010;
        border: none;
        color: white;
        padding: 15px 32px;
        text-align: center;
        text-decoration: none;
        font-size: 16px;
        margin-left: 50px;
        margin-right: 50px;
    }
    .button2 {
        background-color:#e80b3b;
        border: none;
        color: white;
        padding: 15px 32px;
        text-align: center;
        text-decoration: none;
        font-size: 16px;
        margin-left: 50px;
        margin-right: 50px;
    }
    select {
        width: 100%;
        padding: 12px 20px;
        border: none;
        border-radius: 4px;

```

```

    background-color: #f1f1f1;
  }
  td{
    font-size: 18;
  }
  h1{
    text-align: center;
    color: red;
  }
  .front{
    text-align: center;
  }
  .status{
    background-color: yellowgreen;
    border-radius: 0.8rem;
    padding-left: 0.5rem;
  }
  table{
    border-collapse: collapse;
  }
  tr{
    border-bottom: 1pt solid #121010;
  }
  i{
    margin-right: 10px;
  }
</style>
</head>
<body>
<script>
function struck(checkbox){
  var row = checkbox.parentNode.parentNode;
  var t3 = row.getElementsByTagName("td")[0];
  var t1 = row.getElementsByTagName("td")[1];
  if (checkbox.checked) {
    t3.style.textDecoration = "line-through";
    t1.style.textDecoration = "line-through";
    console.log("TASK FINISHED SUCCESSFULLY");
  } else {
    t3.style.textDecoration = "none";
    t1.style.textDecoration = "none";
  }
}
</script>

<h1 align:center>TODO App</h1>
<div class="form-wrapper">
  <form action="add-item" method="post" id="form1">
    <label for="description" style="font-size: 20;">DESCRIPTION</label>
    <br />
    <input

```

```

    type="text"
    name="description"
    id="description"
    placeholder="What do you need to do?"
  >
</input>
  <br>
  <div class="parent">
    <div class="child">
      <label for="category" style="font-size: 20;">CATEGORY</label>
      <select name="category" id="category" placeholder="Choose a category">
        <option selected disabled>Choose a category</option>
        <option value="Personal">Personal</option>
        <option value="Work">Work</option>
        <option value="School">School</option>
        <option value="Cleaning">Cleaning</option>
        <option value="Other">Other</option>
      </select>
    </div>
    <br>
    <div class="child">
      <label for="date" style="font-size: 20;">DUE DATE</label>
      <input type="date" name="date" id="date" />
    </div>
  </div>
  <br>
  <button type="submit" class="button1" style="margin-left: 35%;border-radius:
0.5rem;">
    <i class="fa-solid fa-plus"></i>ADD TASK</button>

</form>
<form action="/delete-item" method="get" id="form2" >
  <button type="submit" class="button2" style="margin-left: 35%; border-radius:
0.5rem;">
    <i class="fa-solid fa-trash"></i>DELETE TASKS</button>
  <br />
  <br>
  <br>

<table class="parenttable" >
  <thead style="font-size: 20px;padding: auto;">
    <tr>
      <th style="padding-right: 26rem;">Task Name</th>
      <th>Category</th>
    </tr>
  </thead>
  <% for(let i of task) { %>
    <tr >
      <td ><input type="checkbox" name="<%= i._id %>" value="box" class="box"
onchange="struck(this)">
        <%=i.description%>

```



```

        <br>
        <i class="fa-solid fa-calendar-days" ></i><%= i.date %>
    </td>
    <td class="status" style="width: fit-content;"><%= i.category %></td>
</tr>
<% } %>
</table>
</form>

</div>
</body>
</html>

```

FILE NAME:.totoschema.js

```

const mongoose = require('mongoose');
const todoSchemaData = new mongoose.Schema({
  description: {
    type: String,
    required: true
  },
  category :{
    type: String,
    required :true
  },
  date:{
    type: String,
    required :true
  }
});

const data = mongoose.model('data',todoSchemaData);
module.exports = data;

```

FILE NAME:.mongoose.js

```
const mongoose = require('mongoose');

mongoose.connect('mongodb://127.0.0.1/todoDB');

const db= mongoose.connection;

db.on('error',console.error.bind(console,"Error in connection"));

db.once('open',function(){

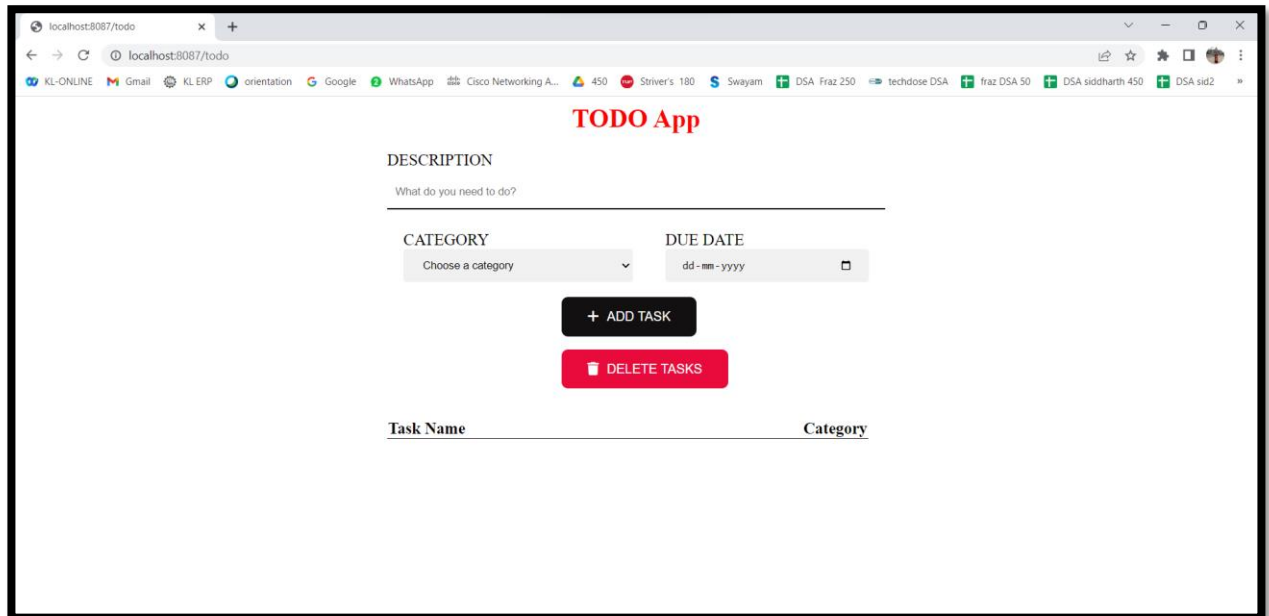
  console.log("DB CONNECTED SUCCESSFULLY");

})
```

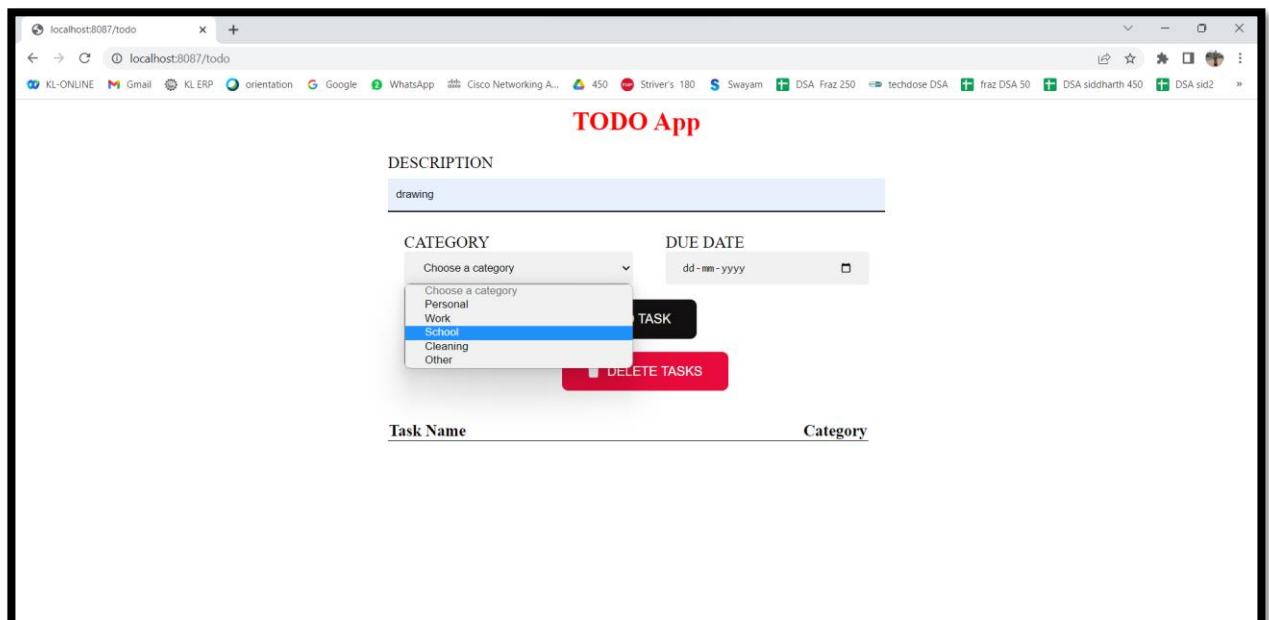
FILE NAME:package.json

```
{
  "name": "todolistapp",
  "version": "1.0.0",
  "description": "Creating a TodoList App using MERN stack technologies",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Navya",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.20.2",
    "ejs": "^3.1.9",
    "express": "^4.18.2",
    "mongodb": "^5.3.0",
    "mongoose": "^7.0.5",
    "nodemon": "^2.0.22",
    "path": "^0.12.7"
  }
}
```

CHAPTER 4 : OUTPUT OF THE PROJECT



The screenshot shows a web browser window with the URL `localhost:8087/todo`. The page title is "TODO App". Below the title, there is a section labeled "DESCRIPTION" with the placeholder text "What do you need to do?". Underneath, there are two input fields: "CATEGORY" with a dropdown menu showing "Choose a category", and "DUE DATE" with a date picker showing "dd-mm-yyyy". Below these fields are two buttons: a black "+ ADD TASK" button and a red "DELETE TASKS" button. At the bottom, there are two labels: "Task Name" and "Category".



The screenshot shows the same web browser window as the previous one, but with the "CATEGORY" dropdown menu open. The menu lists the following options: "Choose a category", "Personal", "Work", "School" (which is highlighted in blue), "Cleaning", and "Other". The "DESCRIPTION" field now contains the text "drawing". The "DUE DATE" field still shows "dd-mm-yyyy". The "+ ADD TASK" and "DELETE TASKS" buttons are still visible. The "Task Name" and "Category" labels are also present at the bottom.

