

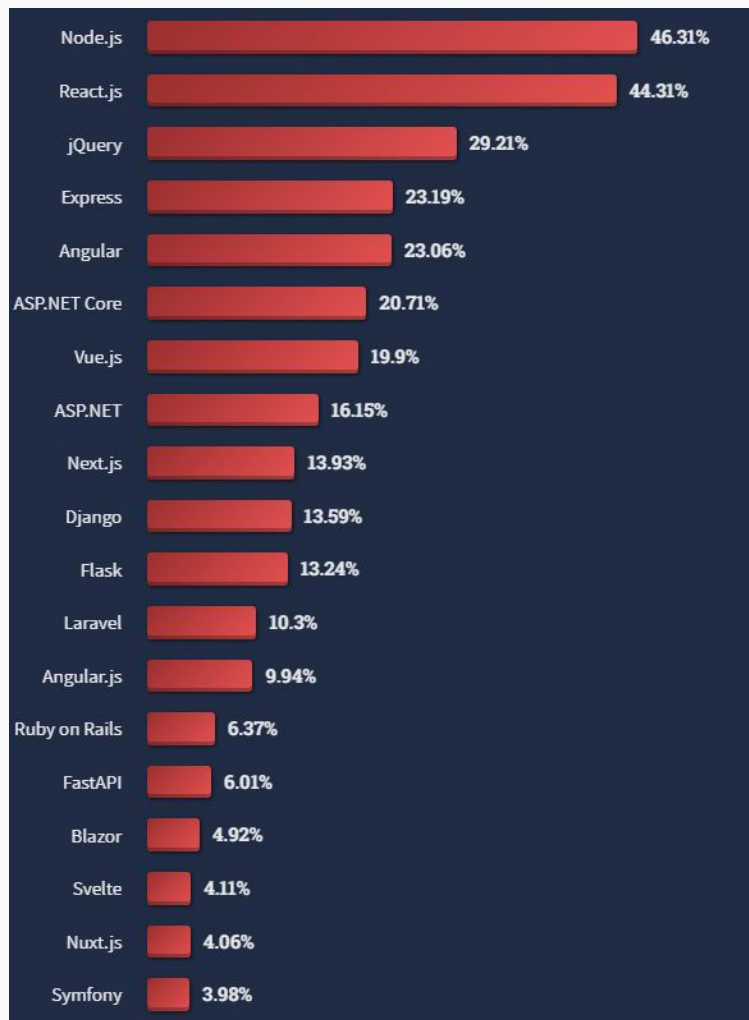
MERN stack

Popularity, CRUD operations and use cases.

POPULARITY

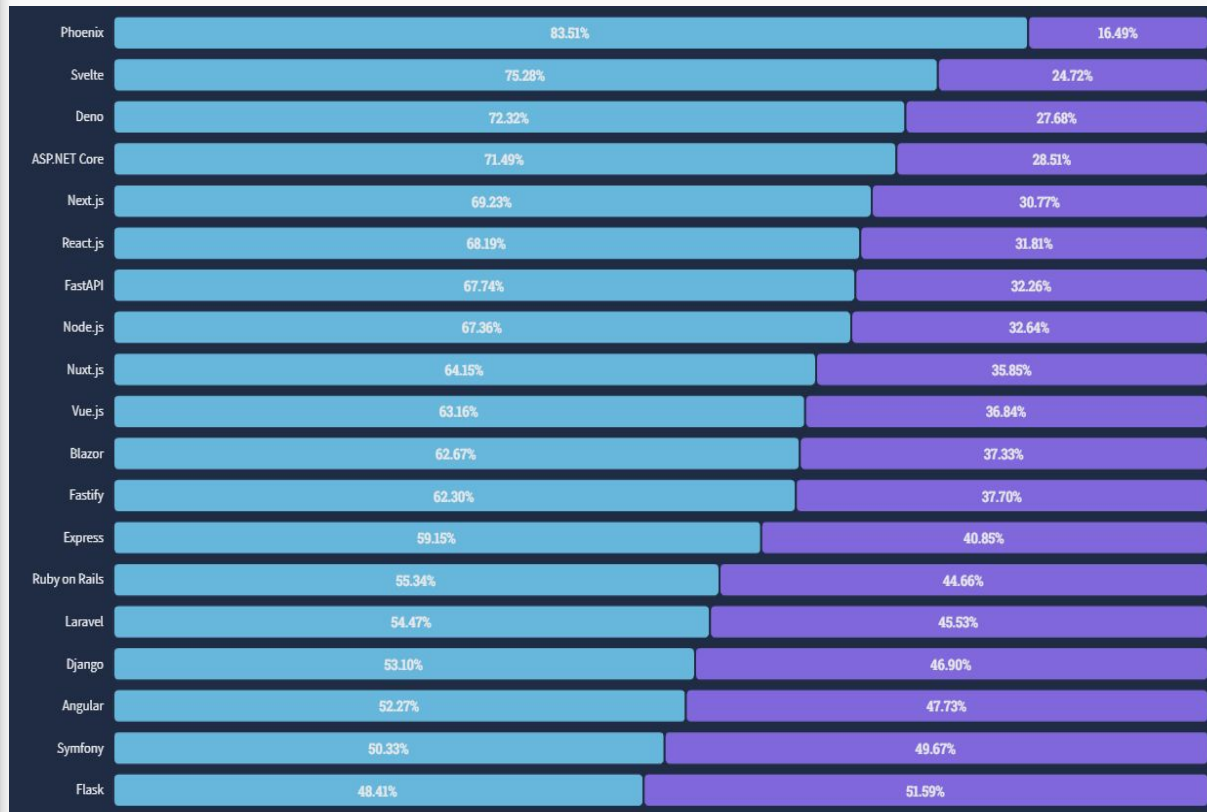
The Stack Overflow dev survey for 2022.

The graph on the right displays the web technologies mostly used in the industry by professional ONLY developers for the current year.



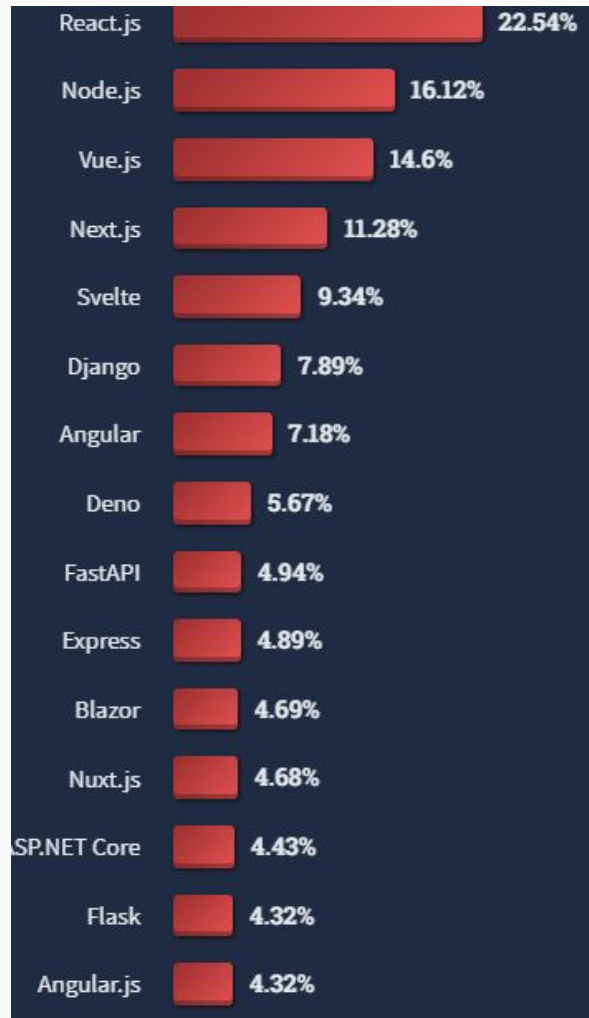
The Stack Overflow dev survey for 2022.

The Loved (Blue) vs Dreaded (Purple) graph.



The Stack Overflow dev survey for 2022.

Technologies that devs show interest in.



PROS AND CONS OF MERN

Pros of the MERN stack

- Quick and cost efficient compared to traditional servers.
- Progressive Web Apps (PWAs) with all in one codebase (like Flutter).
- Fullstack applications (better for smaller codebases).
- Fast and interactive applications with modern state management (eg. Redux).
- Variety of external packages to use.
- Easy to learn and get used to.

Cons of the MERN stack

- React is a library - not a framework, thus third party libraries might be needed in the development.
- Big teams do not seem to operate well on this stack.
- You need external libraries for almost everything (e.g. secure of your HTTP requests), due to lack of a framework (although react frameworks like Next.js seem to be rising in popularity).
- Sometimes configurations of external libraries might lead to low productivity.

THE WEB APP

Task Manager

Add

Total Tasks: 4

Completed Tasks: 2

1. Gym



2. Groceries



3. Car wash



4. Study

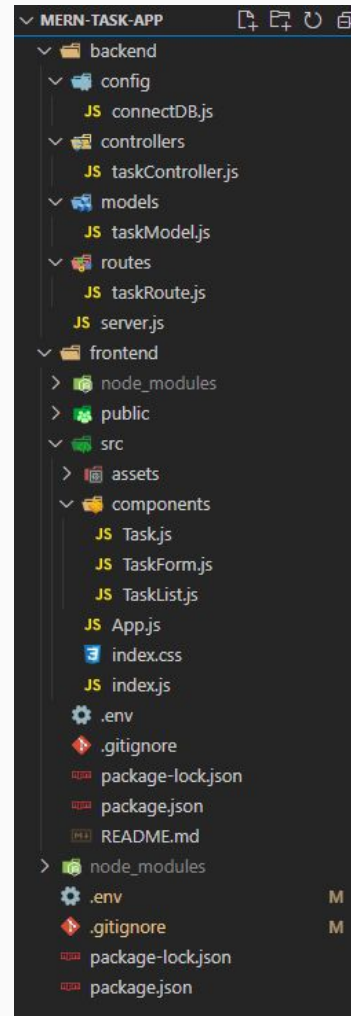


MONGO DB

A NoSQL database like Mongo DB.

- Is a non-relational database (like Maria DB).
- Stores documents with key-value pairs instead of tables.
- Offers flexible schemas.
- Fast queries and cost efficiency.
- Good with Big Data and IoT device/sensor data.
- Easy to setup/use and understand.

The project structure



MONGO DB

backend/config/connectDB.js

backend/models/taskModel.js

JS connectDB.js X

backend > config > JS connectDB.js > [?] <unknown>

```
1  const mongoose = require("mongoose")
2
3  const connectDB = async () => {
4    try {
5      const connect = await mongoose.connect(process.env.MONGO_URI)
6
7      console.log(`MongoDB connected: ${connect.connection.host}`)
8    } catch(error) {
9      console.log(error)
10     process.exit(1)
11   }
12 }
13
14 module.exports = connectDB
```

JS taskModel.js X

backend > models > JS taskModel.js > [?] <unknown>

```
1  const mongoose = require("mongoose")
2
3  const taskSchema = mongoose.Schema(
4    {
5      name: {
6        type: String,
7        required: [true, "Please add a task"]
8      },
9      completed: {
10        type: Boolean,
11        required: true,
12        default: false
13      },
14    },
15    {
16      timestamps: true
17    }
18  )
19
20  const Task = mongoose.model("Task", taskSchema)
21
22  module.exports = Task
```

THE BACKEND

Express/Node server setup

backend/server.js

```
JS server.js X
backend > JS server.js > ...
1  const dotenv = require("dotenv").config()
2  const express = require("express")
3  const connectDB = require("../config/connectDB")
4  const taskRoutes = require("../routes/taskRoute")
5  const cors = require("cors")
6
7  const app = express()
8
9  // Middleware
10 app.use(express.json())
11
12 app.use(cors())
13
14 //Use the routes we created
15 app.use(taskRoutes)
16
17 const PORT = process.env.PORT || 5000
18
19 // Routes
20 app.get("/", (req, res) => {
21   res.send("Home Page")
22 })
23
24 const startServer = async () => {
25   try {
26     await connectDB()
27     app.listen(PORT, () => {
28       console.log(`Server running on Port ${PORT}`)
29     })
30   } catch (error) {
31     console.log(error)
32   }
33 }
34
35 startServer()
36
```


Controller

backend/controller/taskController.js

JS taskController.js X

backend > controllers > JS taskController.js > [?] getTasks

```
1  const Task = require("../models/taskModel");
2
3  const createTask = async (req, res) => {
4    try {
5      const task = await Task.create(req.body)
6      res.status(200).json(task)
7    } catch (error) {
8      res.status(500).json({msg: error.message})
9    }
10 }
11
12 const getTasks = async (req, res) => [
13   try {
14     const task = await Task.find()
15     res.status(200).json(task)
16   } catch (error) {
17     res.status(500).json({msg: error.message})
18   }
19 ]
20
21 const getTask = async (req, res) => {
22   try {
23     const {id} = req.params
24     const task = await Task.findById(id)
25     if (!task) {
26       return res.status(404).json(`No task with id: ${id}`)
27     }
28     res.status(200).json(task)
29   } catch (error) {
30     res.status(500).json({msg: error.message})
31   }
32 }
33
34 const deleteTask = async (req, res) => {
35   try {
36     const {id} = req.params
37     const task = await Task.findByIdAndDelete(id)
```

Routes

backend/routes/taskRoute.js

JS taskRoute.js X

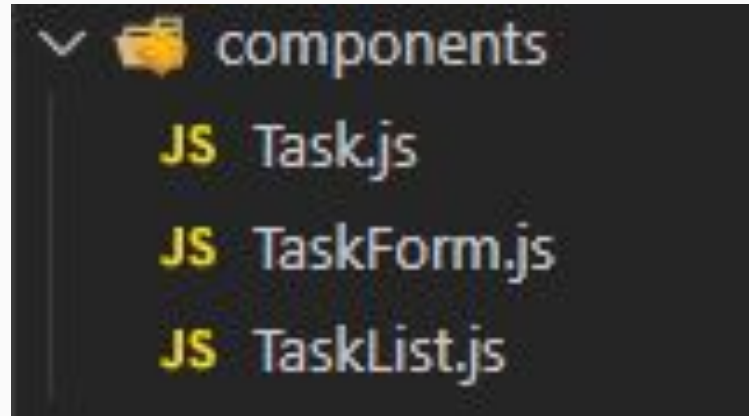
backend > routes > JS taskRoute.js > [?] <unknown>

```
1  const express = require("express")
2  const { createTask, getTasks, getTask, deleteTask, updateTask } = require("../controllers/taskController")
3  const router = express.Router()
4
5  //Create a Task
6  router.post("/api/tasks", createTask)
7
8  // Get Tasks
9  router.get("/api/tasks", getTasks)
10
11 // Get single Task
12 router.get("/api/tasks/:id", getTask)
13
14 // Get single Task
15 router.delete("/api/tasks/:id", deleteTask)
16
17 // Update Task
18 router.put("/api/tasks/:id", updateTask) //update with put --> specify every field --> "name" and "completed"
19
20 module.exports = router
```

THE FRONTEND

React js frontend

- Simple npm create-react-app structure.
- Added the components folder only.



Hooks and state management

frontend/component/TaskList.js

```
JS TaskList.js M • JS TaskForm.js
frontend > src > components > JS TaskList.js > [⌘] TaskList > [⌘] getTasks
1  import { useEffect, useState } from "react"
2  import { toast } from "react-toastify"
3  import Task from "../Task"
4  import TaskForm from "../TaskForm"
5  import axios from "axios"
6  import { URL } from "../App"
7  import loadingImage from "../assets/loader.gif"
8
9  const TaskList = () => {
10     const [tasks, setTasks] = useState([])
11     const [completedTasks, setCompletedTasks] = useState([])
12     const [isLoading, setIsLoading] = useState(false)
13     const [isEditing, setIsEditing] = useState(false)
14     const [taskId, setTaskID] = useState('')
15
16     const [formData, setFormData] = useState({
17       name: "",
18       completed: false
19     })
20     const {name} = formData
21
22     useEffect(() => {
23       getTasks()
24     }, [])
25
26     const handleInputChange = (e) => {
27       const {name, value} = e.target
28       setFormData({...formData, [name]: value})
29     }
30
```

Requests to our API

frontend/component/TaskList.js

```
const getTasks = async (e) => {
  setIsLoading(true)
  try {
    const {data} = await axios.get(`${URL}/api/tasks`)
    setTasks(data)
    setIsLoading(false)
  } catch (error) {
    toast.error(error.message)
    setIsLoading(false)
  }
}

const createTask = async (e) => {
  e.preventDefault()
  if (name === "") {
    return toast.error("Input field cannot be empty")
  }
  try {
    await axios.post(`${URL}/api/tasks`, formData)
    toast.success("Task added successfully")
    setFormData({...formData, name: ""})
    getTasks()
  } catch (error) {
    toast.error(error.message)
  }
}

const deleteTask = async (id) => {
  try {
    await axios.delete(`${URL}/api/tasks/${id}`)
    getTasks()
  } catch (error) {
    toast.error(error.message)
  }
}

const updateTask = async (e) => {
  e.preventDefault()
  if (name === "") {
    return toast.error("Input field cannot be empty.")
  }
  try {
    await axios.put(`${URL}/api/tasks/${taskId}`, formData)
    setFormData({...formData, name: ''})
    setIsEditing(false)
    getTasks()
  } catch (error) {
    toast.error(error.message)
  }
}
```

The taskList content

```
frontend/component/TaskList.js
```

```
return (
  <div>
    <h2>Task Manager</h2>
    <TaskForm name={name} handleInputChange={handleInputChange} createTask={createTask} isEditing={isEditing} updateTask={updateTask}/>
    {tasks.length > 0 && (
      <div className="--flex-between --pb">
        <p>
          <b>Total Tasks:</b> {tasks.length}
        </p>
        <p>
          <b>Completed Tasks:</b> {completedTasks.length}
        </p>
      </div>
    )}
    <hr />
    {
      isLoading && (
        <div className="--flex-center">
          <img src={loadingImage} alt="Loading" />
        </div>
      )
    }
    {
      !isLoading && tasks.length === 0 ? (
        <p className="--py">No task added. Please add a task</p>
      ) : (
        <>
          {tasks.map((task, index) => {
            return (
              <Task key={task._id} task={task} index={index} deleteTask={deleteTask} getSingleTask={getSingleTask} setToComplete={setToComplete}/>
            )
          })}
        </>
      )
    }
  </div>
)
```

The Task content

frontend/component/Task.js

```
JS Task.js  X  JS TaskForm.js
frontend > src > components > JS Task.js > ...
1  import {FaEdit, FaCheckDouble, FaRegTrashAlt} from "react-icons/fa"
2
3  const Task = ({task, index, deleteTask, getSingleTask, setToComplete}) => {
4    return (
5      <div className={task.completed ? "task completed" : "task"}>
6        <p>
7          <b>{index + 1}. </b>
8          {task.name}
9        </p>
10       <div className="task-icons">
11         <FaCheckDouble color="green" onClick={() => setToComplete(task)} />
12         <FaEdit color="purple" onClick={() => getSingleTask(task)} />
13         <FaRegTrashAlt color="red" onClick={() => deleteTask(task._id)} />
14       </div>
15     </div>
16   )
17 }
18
19 export default Task
20
```


The TaskForm content

JS TaskForm.js M ●

frontend > src > components > JS TaskForm.js > [🔍] default

```
1  const TaskForm = ({createTask, name, handleInputChange, isEditing, updateTask}) => {
2    return (
3      <form className="task-form" onSubmit={isEditing ? updateTask : createTask}>
4        <input type="text" placeholder="Add a Task" name="name" value={name} onChange={handleInputChange} />
5        <button type="submit">{ isEditing ? 'Edit' : 'Add'}</button>
6      </form>
7    )
8  }
9
10 export default TaskForm
11
```

USE CASES

Use cases of MERN stack

- Data intensive apps (eg. iot apps, streaming platforms).
- Smaller codebase apps.
- Interactive and state management intensive apps.
- Single page applications.
- Scalable applications that will be maintained in the future (thanks to npm's popularity).

and many more...



**THANK YOU
FOR THE
YOUR TIME!**