

**T A S K   M A N A G E R**

nicola szwaja

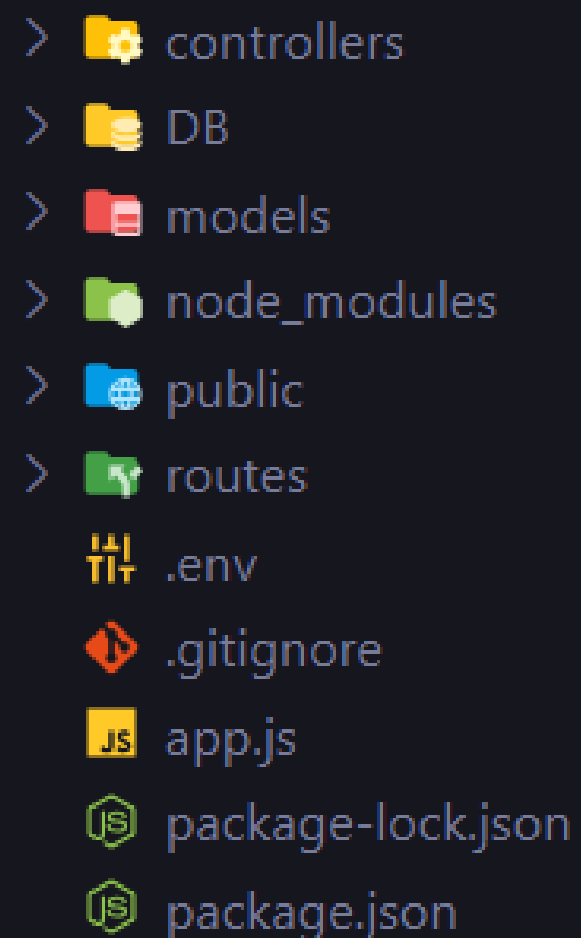


Express



**WYKORZYSTANIE  
NODE.JS I  
EXPRESS.JS**

# WYKORZYSTANIE NODE.JS I EXPRESS.JS



```
> controllers
> DB
> models
> node_modules
> public
> routes
.env
.gitignore
app.js
package-lock.json
package.json
```

Aplikacja została zaprojektowana zgodnie z najlepszymi praktykami Node.js, co obejmuje klarownie zdefiniowaną strukturę katalogów i plików

- **/src/controllers:** W tym katalogu znajdują się pliki kontrolerów
- **/src/middlewares:** Zawiera middleware wykorzystywane przez aplikację, umożliwiające np. autentykację, walidację itp.
- **/src/routes:** Tutaj umieszczone są pliki definiujące ścieżki routingu aplikacji.
- **/public:** Zasoby statyczne, takie jak style CSS, obrazy czy pliki JavaScript.

# WYKORZYSTANIE NODE.JS I EXPRESS.JS

Aplikacja wykorzystuje system routingu dostarczony przez Express.js. Ścieżki są jasno zdefiniowane w plikach znajdujących się w katalogu **/src/routes**. Middleware są odpowiednio używane, aby przetwarzać żądania na różnych etapach przepływu danych w aplikacji.

Przykład definicji ścieżki w pliku `/src/routes/tasks.js`:

```
const express = require("express")
const router = express.Router()

const { getAllTasks,
  createTask,
  getTask,
  updateTask,
  deleteTask } = require("../controllers/tasks")

router.route("/").get(getAllTasks).post(createTask)
router.route("/:id").get(getTask).patch(updateTask).delete(deleteTask)

module.exports = router
```

# WYKORZYSTANIE NODE.JS I EXPRESS.JS

```
const getTask = async (req, res) => {
  try {
    const { id: taskID } = req.params
    const task = await Task.findOne({_id: taskID})
    if(!task){
      return res.status(404).json({msg: `No task with id : ${taskID}`})
    }
    res.status(200).json({task})
  } catch (error) {
    res.status(500).json({msg: error})
  }
}
```

e.g. wash dishes

Error, Please Try Again

Aplikacja zawiera mechanizm obsługi błędów, który ułatwia śledzenie i diagnozowanie problemów. Błędy są odpowiednio logowane, a użytkownikom prezentowane są czytelne komunikaty.

# BAZA DANYCH

## NARZĘDZIA

Aplikacja została zintegrowana z bazą danych MongoDB, a do komunikacji z bazą danych używamy biblioteki Mongoose jako odwzorowania obiektowego (ODM - Object Data Modeling).





# DLACZEGO MONGO DB?

1. MongoDB umożliwia elastyczne zarządzanie danymi. Bez potrzeby ściśle zdefiniowanych schematów, można łatwo dostosowywać strukturę danych do bieżących potrzeb projektu.
2. MongoDB cieszy się dużą popularnością, co przekłada się na aktywną społeczność i dostępność bogatej dokumentacji.
3. MongoDB oferuje intuicyjne API i narzędzia do zarządzania bazą danych. Dzięki temu mogą szybko tworzyć, edytować i odczytywać dane bez zbędnego skomplikowania.

# BAZA DANYCH

QUERY RESULTS: 1-3 OF 3

```
_id: ObjectId('65ab271170c57309c093f620')  
completed: false  
name: "task01"  
__v: 0
```

```
_id: ObjectId('65ab271570c57309c093f623')  
completed: false  
name: "task02"  
__v: 0
```



# BAZA DANYCH

```
const start = async () => {  
  try {  
    await connectDB(process.env.MONGO_URI)  
    app.listen(port, console.log(`Server started on port ${port}...`))  
  } catch (error) {  
    console.log(error)  
  }  
}
```

# BAZA DANYCH

```
const TaskSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'must provide name'],
    trim: true,
    maxlength: [20, 'name cannot be more than 20 characters']
  },
  completed: {
    type: Boolean,
    default: false
  }
})
```

Schematy danych zostały zaprojektowane zgodnie z wymaganiami aplikacji.

# REJESTRACJA I LOGOWANIE



funkcja jest dopracowywana :)

# WYSZUKIWANIE DANYCH

W moim projekcie zaimplementowana została funkcja umożliwiająca filtrowanie zadań w sposób efektywny. Dzięki tej funkcji użytkownik może łatwo wyświetlić jedynie te zadania, które nie zostały jeszcze oznaczone jako ukończone. Ta intuicyjna opcja filtrowania pozwala skupić się na bieżących i niewykonanych zadaniach, ułatwiając zarządzanie projektem i zwiększając efektywność pracy.

## Task Manager

[only show uncompleted tasks](#)

<input type="checkbox"/>	Task01	<input type="checkbox"/> <input type="checkbox"/>
<input checked="" type="checkbox"/>	Name	<input type="checkbox"/> <input type="checkbox"/>
<input type="checkbox"/>	Zada	<input type="checkbox"/> <input type="checkbox"/>

# Task Manager

☐

Task01

☒

Name

☐

Zada

☐

☐

☐

☐

☐

☐

only show uncompleted tasks

# Task Manager

☐

Task01

☐

Zada

☐

☐

☐

☐

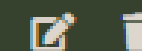
☐

☐

show all tasks

# DODAWANIE EDYTOWANIE I USUWANIE DANYCH

Task01



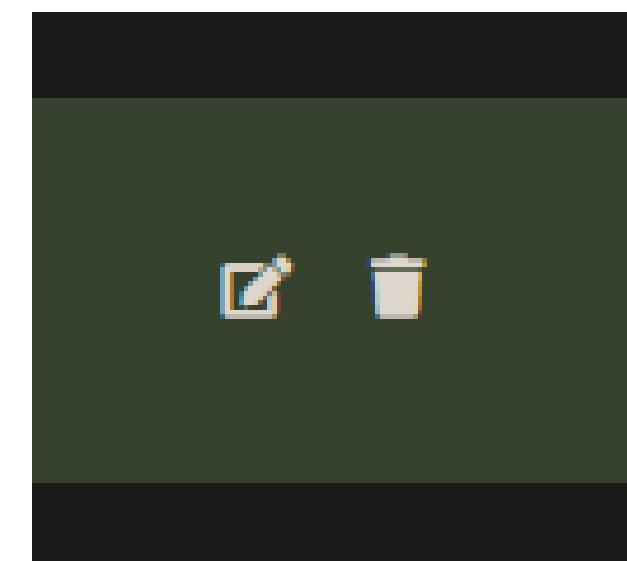
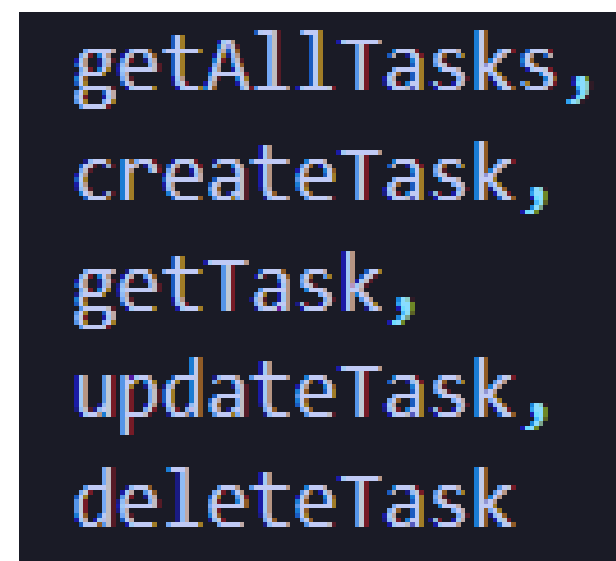
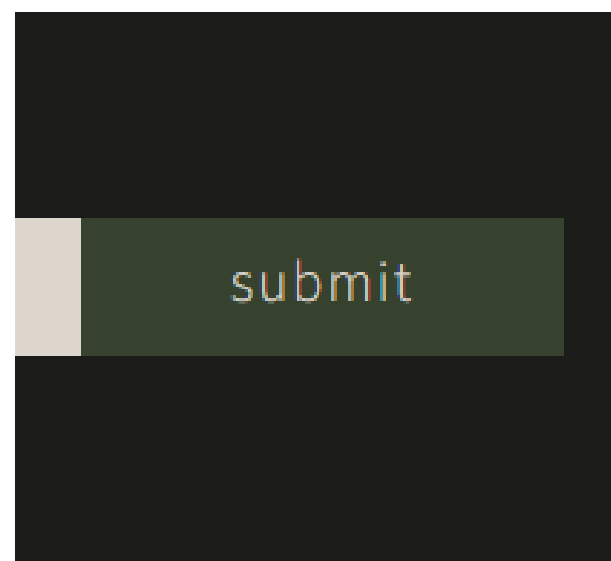
Task02



Task03



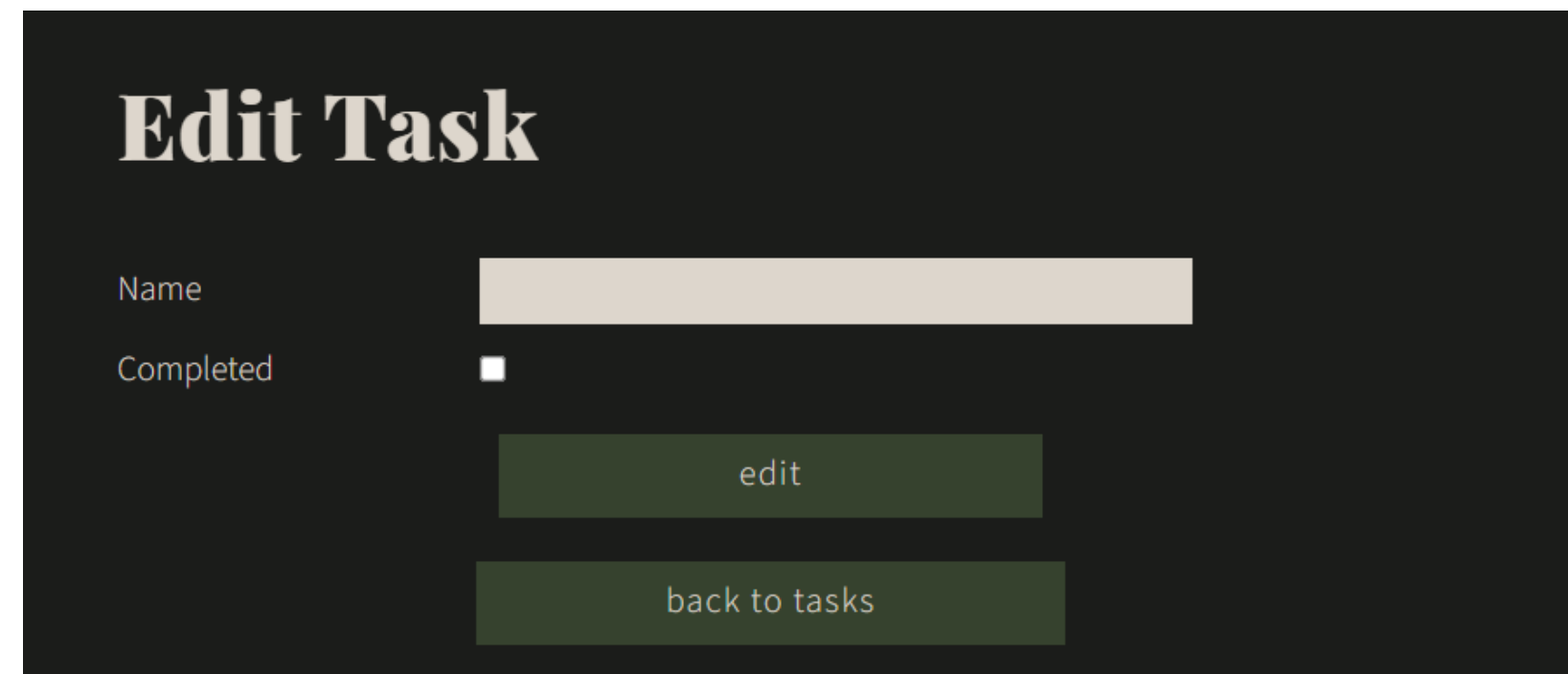
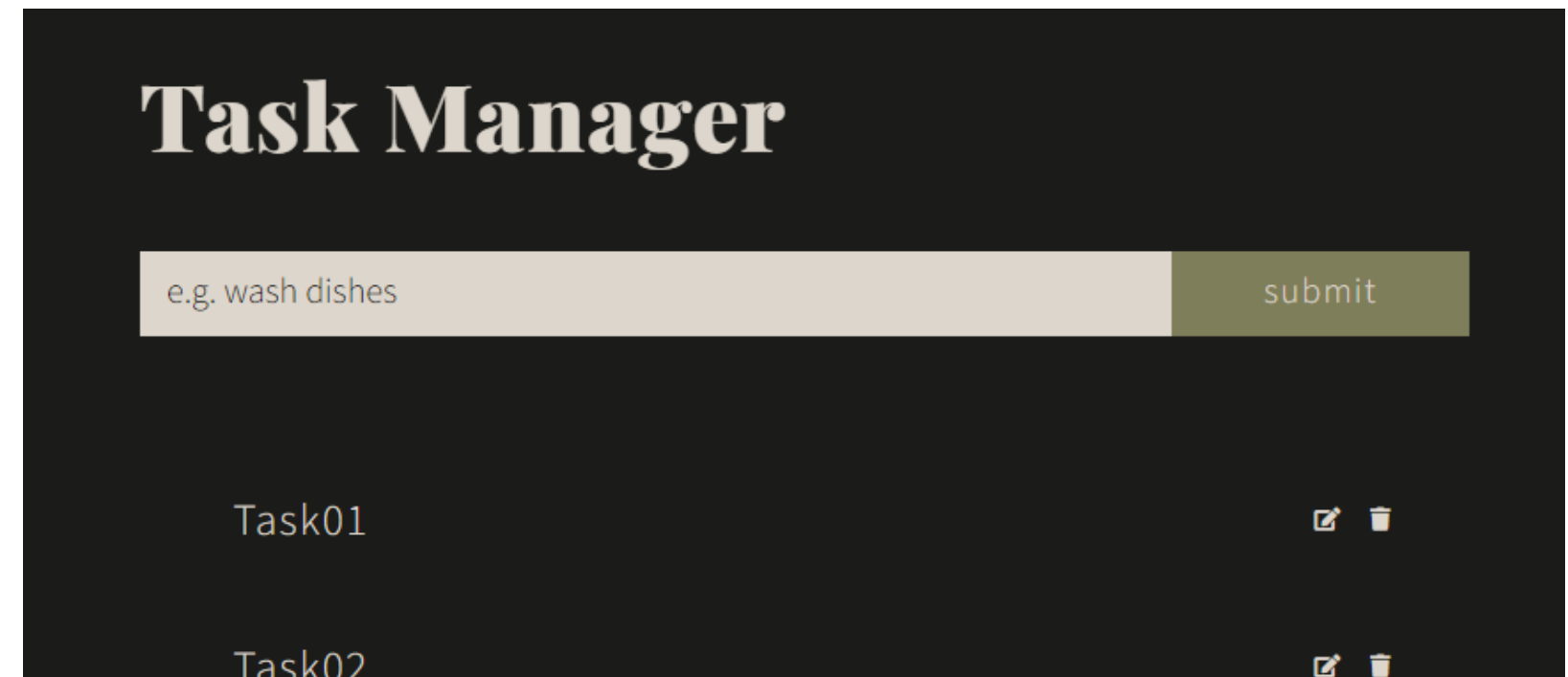
# DODAWANIE EDYTOWANIE I USUWANIE DANYCH



Interfejs użytkownika został zaprojektowany z myślą o przejrzystości i intuicyjności dla wszystkich operacji na danych. Zapewniamy spójność danych poprzez zastosowanie transakcji w odpowiednich scenariuszach. Wszystkie operacje w ramach jednej transakcji są wykonane kompletnie i poprawnie, a w przypadku wystąpienia błędu, żadna z nich nie jest wykonywana. Aplikacja dostarcza jasne komunikaty zwrotne dla użytkownika, informując go o wynikach operacji.

# PRZEJRZYSTOŚĆ I INTUICYJNOŚĆ PROJEKTU

Projekt interfejsu użytkownika został starannie zaplanowany, aby być łatwym w obsłudze i intuicyjnym dla użytkowników. Zastosowano zasady dostępności, zapewniając korzystanie aplikacji osobom o różnych potrzebach. Responsywny design dostosowuje się do różnych urządzeń, natomiast atrakcyjna estetyka projektu tworzy spójne wrażenie wizualne dla użytkowników.





# PRZEJRZYSTOŚĆ I INTUICYJNOŚĆ PROJEKTU



favicon



paleta kolorów