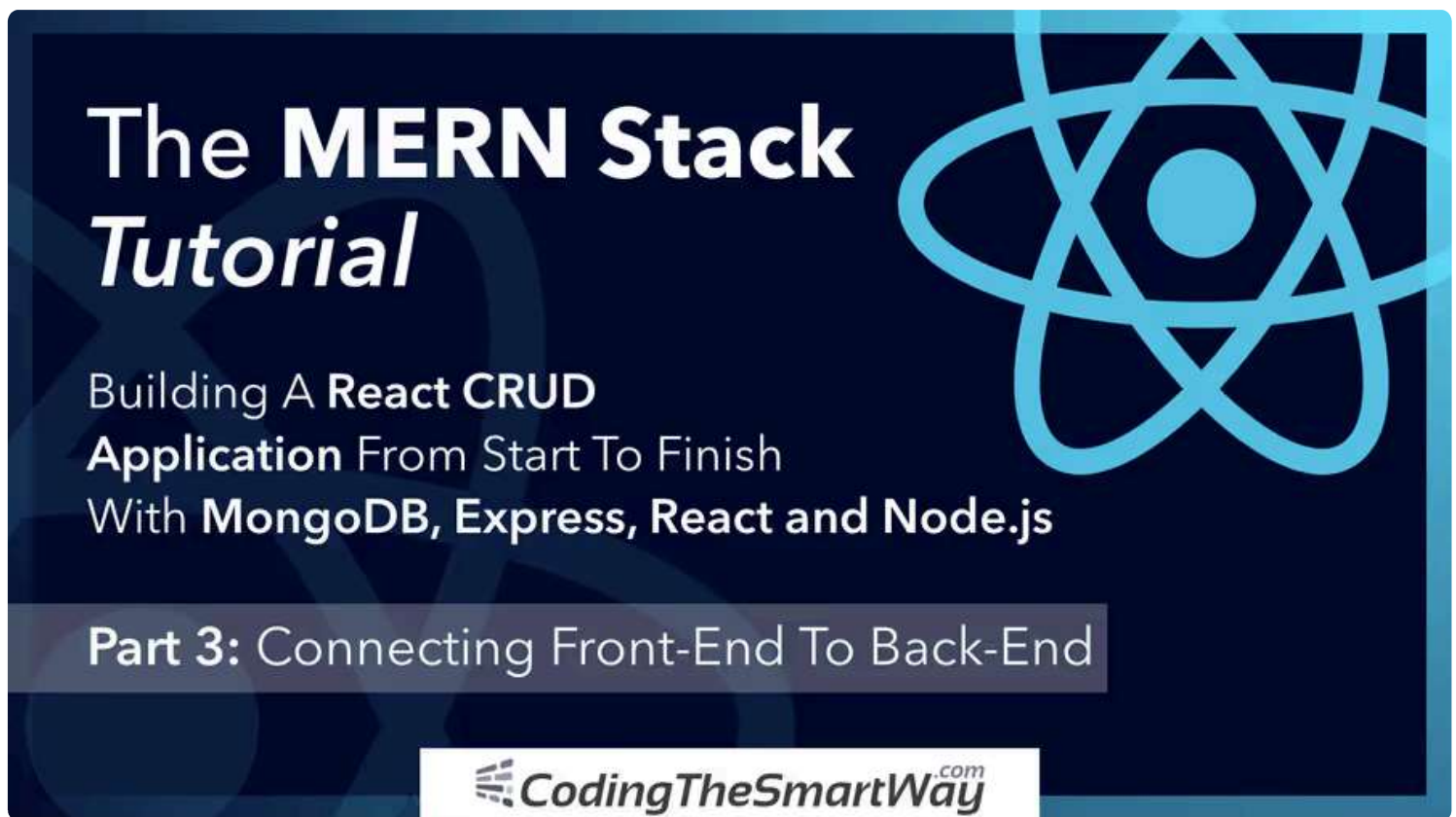


Dec 29, 2018 ~ 5 min read

# The MERN Stack Tutorial – Building A React CRUD Application From Start To Finish – Part 3



[Video Tutorial on YouTube](#)

## Part 3: Connecting Front-End To Back-End

This is the third part of *The MERN Stack Tutorial - Building a React CRUD Application From Start To Finish* series. In the second part we've finished the implementation of the back-end part by using Node.js, Express, and MongoDB. In this part we're now ready to return to the React front-end application (which we've started to implemented in the first part) and add the connection to the back-end, so that the user will be able to

- create new todo items
- see an overview of all todo items

The communication between front-end and back-end will be done by sending HTTP request to the various server endpoints we've created in the last part.

## Installing Axios

In order to be able to send HTTP request to our back-end we're making use of the Axios library. Axios is being installed via the following command:

```
`$ npm install axios`
```

Once Axios is added to the project we're ready to further complete the implementation of CreateTodo component and send data to the back-end.

## Completing The Implementation Of createTodo Component

First let's add the following import statement to *create-todo.component.ts* so that we're ready to use the Axios library in that file:

```
import axios from 'axios';
```

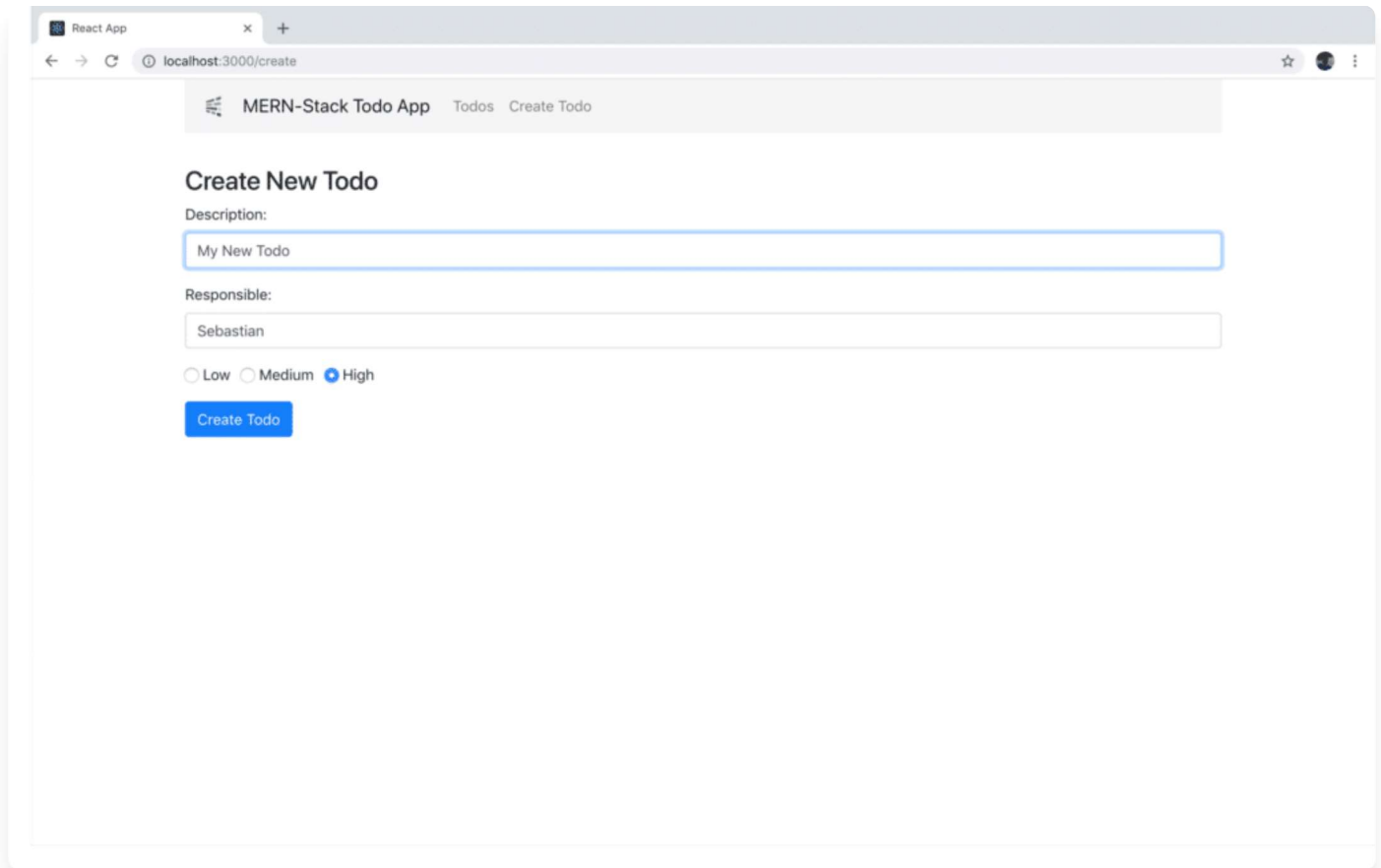
The right place where the code needs to be added which is responsible for sending the data of the new todo element to the back-end is the *onSubmit* method. The existing implementation of *onSubmit* needs to be extended in the following way:

```
onSubmit(e) {  
  e.preventDefault();  
  
  console.log(`Form submitted:`);  
  console.log(`Todo Description: ${this.state.todo_description}`);  
  console.log(`Todo Responsible: ${this.state.todo_responsible}`);  
  console.log(`Todo Priority: ${this.state.todo_priority}`);  
  
  const newTodo = {  
    todo_description: this.state.todo_description,  
    todo_responsible: this.state.todo_responsible,  
    todo_priority: this.state.todo_priority,  
    todo_completed: this.state.todo_completed  
  };  
  
  axios.post('http://localhost:4000/todos/add', newTodo)  
    .then(res => console.log(res.data));  
  
  this.setState({  
    todo_description: '',  
    todo_responsible: '',  
    todo_priority: '',  
    todo_completed: false  
  })  
}
```

Here we're using the *axios.post* method to send an HTTP POST request to the back-end endpoint *http://localhost:4000/todos/add*. This endpoint is expecting to get the

new todo object in JSON format in the request body. Therefore we need to pass in the *newTodo* object as a second argument.

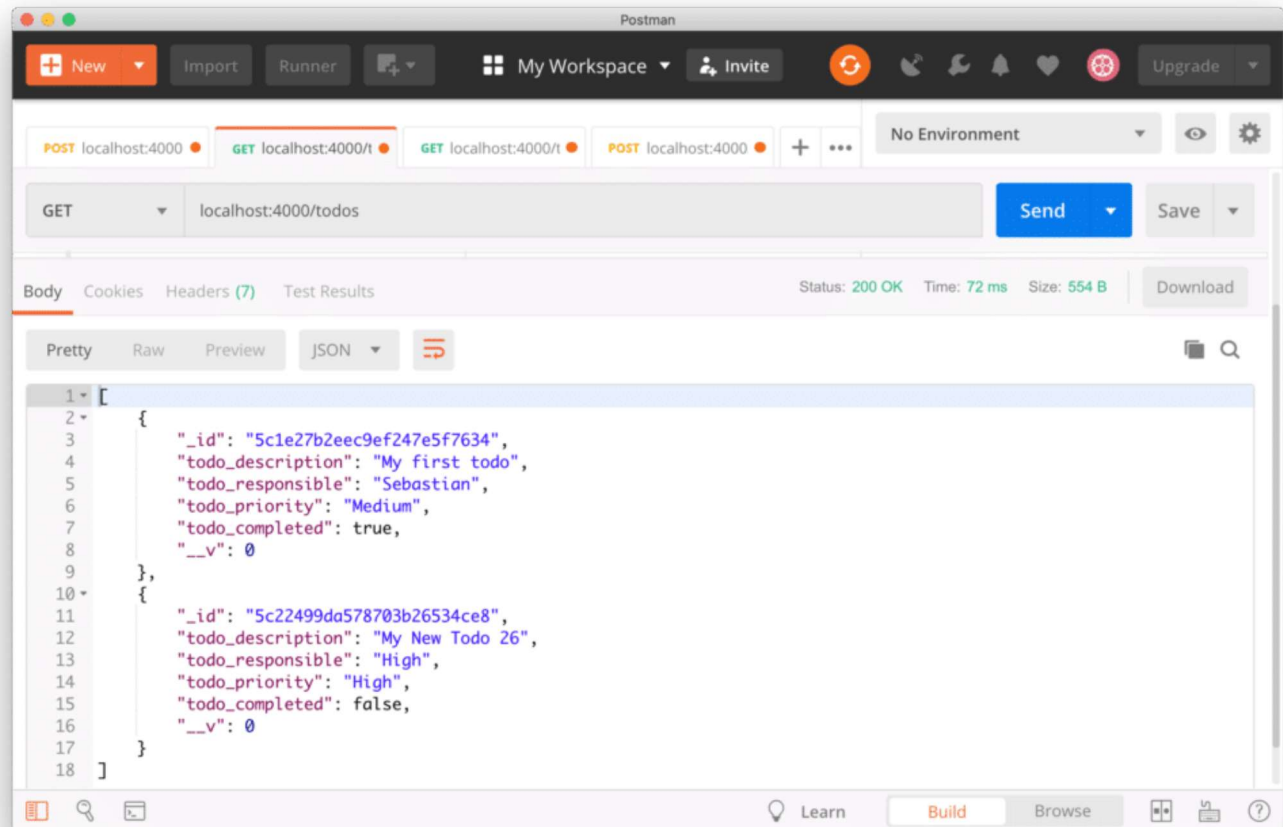
Let's fill out the create todo form:



The screenshot shows a web browser window with the title 'React App' and the address bar displaying 'localhost:3000/create'. The page has a navigation bar with 'MERN-Stack Todo App', 'Todos', and 'Create Todo'. The main content area is titled 'Create New Todo' and contains the following form elements:

- A label 'Description:' followed by a text input field containing 'My New Todo'.
- A label 'Responsible:' followed by a text input field containing 'Sebastian'.
- Three radio buttons for priority: 'Low', 'Medium', and 'High'. The 'High' radio button is selected.
- A blue button labeled 'Create Todo'.

Hit button *Create Todo* to send entered data to the server. By using Postman again, we're able to check if data has been stored in the MongoDB database. Let's again send a GET request to endpoint */todos* and check if the new todo item is being returned:



Now that we've made sure that we're able to create new todo items from the React front-end application we're ready to move on and further complete the implementation of *TodosList* component in the next step.

## Completing The Implementation Of TodosList Component

In *todos-list.component.ts* we start by adding the following import statement on top:

```
import { Link } from 'react-router-dom';  
import axios from 'axios';
```

In the next step, let's use the component's constructor to initialize the state with an empty *todos* array:

```
constructor(props) {  
  super(props);  
  this.state = {todos: []};  
}
```

To retrieve the todos data from the database the *componentDidMount* lifecycle method is added:

```
componentDidMount() {  
  axios.get('http://localhost:4000/todos/')  
    .then(response => {  
      this.setState({ todos: response.data });  
    })  
    .catch(function (error){  
      console.log(error);  
    })  
}
```

Here we're using the *axios.get* method to access the */todos* endpoint. Once the result becomes available we're assigning *response.data* to the *todos* property of the component's state object by using the *this.setState* method.

Finally the JSX code needs to be added to the *return* statement of the *render* function like you can see in the following listing:

```
render() {  
  return (  
    <div>  
      <h3>Todos List</h3>  
      <table className="table table-striped" style={{ marginTop: 20 }} >
```

```

        <thead>
          <tr>
            <th>Description</th>
            <th>Responsible</th>
            <th>Priority</th>
            <th>Action</th>
          </tr>
        </thead>
        <tbody>
          { this.todoList() }
        </tbody>
      </table>
    </div>
  )
}

```

The output is done as a table and inside the *tbody* element we're making use of the *todoList* method to output a table row for each todo item. Because of that we need to add the implementation of *todoList* method to *TodosList* component as well:

```

todoList() {
  return this.state.todos.map(function(currentTodo, i){
    return <Todo todo={currentTodo} key={i} />;
  })
}

```

Inside this method we're iterating through the list of todo items by using the *map* function. Each todo item is output by using the *Todo* component which is not yet implemented. The current todo item is assigned to the *todo* property of this component.

To complete the code in *todos-list.component.js* we need to add the implementation of Todo component as well. In the following listing you can see the complete code:

```
import React, { Component } from 'react';
import { Link } from 'react-router-dom';
import axios from 'axios';

const Todo = props => (
  <tr>
    <td>{props.todo.todo_description}</td>
    <td>{props.todo.todo_responsible}</td>
    <td>{props.todo.todo_priority}</td>
    <td>
      <Link to={"/edit/"+props.todo._id}>Edit</Link>
    </td>
  </tr>
)

export default class TodosList extends Component {

  constructor(props) {
    super(props);
    this.state = { todos: [] };
  }

  componentDidMount() {
    axios.get('http://localhost:4000/todos/')
      .then(response => {
        this.setState({ todos: response.data });
      })
      .catch(function (error){
        console.log(error);
      })
  }
}
```



```

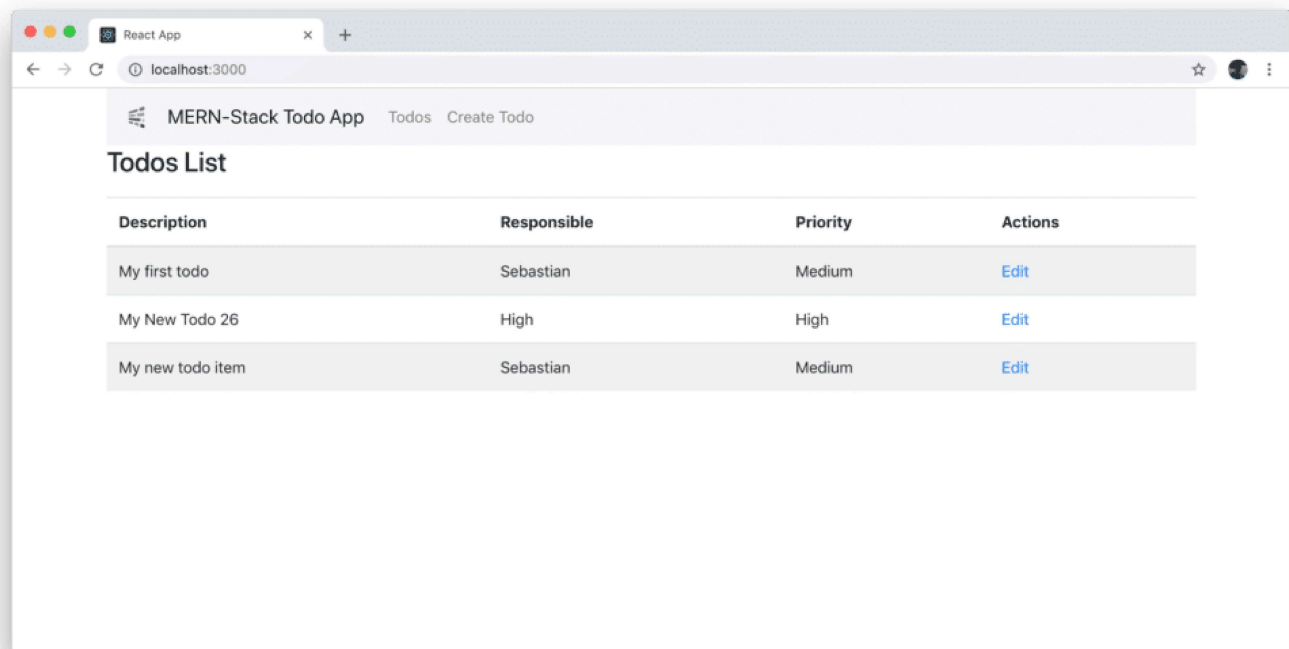
    todoList() {
      return this.state.todos.map(function(currentTodo, i){
        return <Todo todo={currentTodo} key={i} />;
      })
    }

    render() {
      return (
        <div>
          <h3>Todos List</h3>
          <table className="table table-striped" style={{ marginTop: 20 }} >
            <thead>
              <tr>
                <th>Description</th>
                <th>Responsible</th>
                <th>Priority</th>
                <th>Action</th>
              </tr>
            </thead>
            <tbody>
              { this.todoList() }
            </tbody>
          </table>
        </div>
      )
    }
  }
}

```

Todo component is implemented as a functional React component. It outputs the table row which contains the values of the properties of the todo item passed into that component. Inside the Actions column of the table we're also outputting a link to ``/edit/:id`` route by using the Link component.

The resulting output should then look like the following:



## What's Next

Now, we've connected the first parts of the React front-end application to the back-end. However, some part are still missing. In the upcoming and last part of this tutorial series we're going to further complete our front-end application so that the user will also be able to edit todo items and set todo items to completed.

react   web-development   mern   mern-stack

Imprint / Impressum · Data Privacy Policy / Datenschutzerklärung ·



