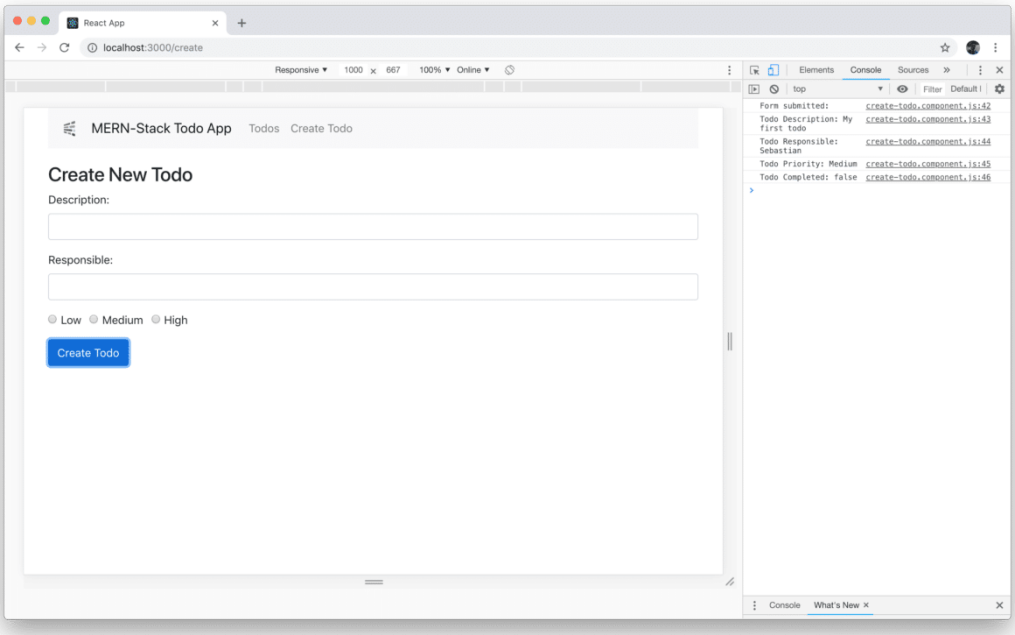
Creating a CRUD API and connecting it to React App

Part 1 - Setting Up The Project



starter react project link - https://drive.google.com/drive/folders/1B_o8EqZV0r562FoiLEkZf05QtbLqu4PB?usp=sharing

Part 2: Setting Up The Back-end

The back-end will comprise HTTP endpoints to cover the following use cases:

- Retrieve the complete list of available todo items by sending an HTTP GET request
- Retrieve a specific todo item by sending HTTP GET request and provide the specific todo ID in addtion
- Create a new todo item in the database by sending an HTTP POST request
- Update an existing todo item in the database by sending an HTTP POST request

Initiating The Back-end Project

To initiate the back-end project let's create a new empty project folder:

```
$ mkdir backend
```

Change into that newly created folder by using:

```
$ cd backend
```

Let's create a *package.json* file inside that folder by using the following command:

```
$ npm init -y
```

With the *package.json* file available in the project folder we're ready to add some dependencies to the project:

```
$ npm install express cors mongodb
```

Let's take a quick look at the four packages:

- **express**: Express is a fast and lightweight web framework for Node.js. Express is an essential part of the MERN stack.
- **cors**: CORS is a node.js package for providing an Express middleware that can be used to enable CORS with various options. Cross-origin resource sharing (CORS) is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served.
- **mongodb -** database operations

Finally we need to make sure to install a global package by executing the following command:

```
$ npm install -g nodemon
```

Nodemon is a utility that will monitor for any changes in your source and automatically restart your server. We'll use *nodemon* when running our Node. server in the next steps.

Inside of the backend project folder create a new file named *server.js* and insert the following basic Node.js / Express server implementation:

```
const express = require("express");
const app = express();
const { MongoClient } = require("mongodb");
const PORT = 4000;
let db;

let connectionString = `mongodb://localhost:27017/crud`;

app.use(express.json());

MongoClient.connect(
    connectionString,
    { useNewUrlParser: true },
    (error, client) => {
        if (error) {
            return console.log("Connection failed for some reason");
        }
        console.log("Connection established - All well");
        db = client.db("crud");
        app.listen(PORT, function () {
            console.log("Server is running on Port: " + PORT);
        });
    },
);
```

Here above the MongoDB.connect() taking the connection string as the first argument, after that a second argument is an object for not gettir the **deprecation warnings** from MongoDB.And finally, the last argument is the callback function which can be used after MongoDB.connect() functic trying to connect to MongoDB.

In our case, we are storing the database in a variable for further use and also started the app for listening in port 4000.

Now that we have our database connected, let's create some endpoints to make the app useful.

Start the server by using *nodemon*:

```
$ nodemon server
```

You should now see an output similar to the following:

Implementing The Server Endpoints

To setup the endpoints we need to create an instance of the Express Router by adding the following line of code in `server.js` :

```
const todoRoutes = express.Router();
```

The router will be added as a middleware in `server.js` and will take control of request starting with path */todos*:

```
app.use('/todos', todoRoutes);
```

First of all we need to add an endpoint which is delivering all available todos items:

```
todoRoutes.route('/').get(function(req, res) {
    db.collection('todos')
    .find()
    .toArray(function (err, items) {
      res.send(items)
```

```
    })
});
```

The next endpoint which needs to be implemented is */:id*. This path extension is used to retrieve a todo item by providing an ID. The implementation log is straight forward:

```
todoRoutes.route("/:id").get(function (req, res) {
    let id = req.params.id;
    db.collection("todos")
        .find({ _id: id })
        .toArray(function (err, items) {
            res.send(items);
        });
});
```

Here we're accepting the URL parameter *id* which can be accessed via *req.params.id*.

Next, let's add the route which is needed to be able to add new todo items by sending a HTTP post request (*/add*):

```
todoRoutes.route("/add").post(function (req, res) {
    db.collection("todos").insertOne(req.body, function (err, info) {
        res.json(info.ops[0]);
    });
});
```

The new todo item is part the the HTTP POST request body, so that we're able to access it view *req.body.*

HTTP PUT route */update/:id* is added:

```
todoRoutes.route("/update/:id").put(function (req, res) {
    db.collection("data").findOneAndUpdate(
        { _id: req.params.id },
        {
            $set: {
                todo_description: req.body.todo_description,
                todo_responsible: req.body.todo_responsible,
                todo_priority: req.body.todo_priority,
                todo_complete: req.body.todo_complete,
            },
        },
        function () {
            res.send("Success updated!");
        },
    );
});
```

This route is used to update an existing todo item (e.g. setting the *todo_completed* property to *true*). Again this path is containing a parameter: id.

Finally HTTP DELETE route to /delete/:id is added:

```
todoRoutes.route("/delete/:id").delete(function (req, res) {
    db.collection('todos').deleteOne(
    { _id: new ObjectId(req.params.id) },
    function () {
      res.send('Successfully deleted!')
    }
  )
});
```

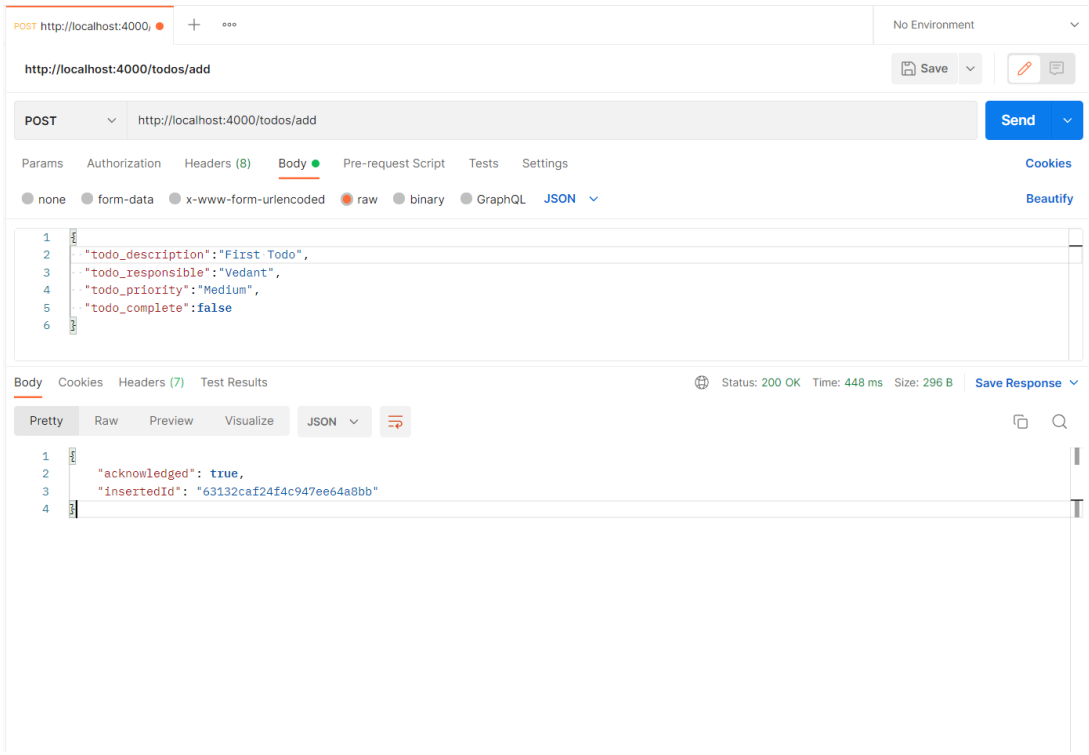
Finally, in the following you can see the complete and final code of *server.js* again:

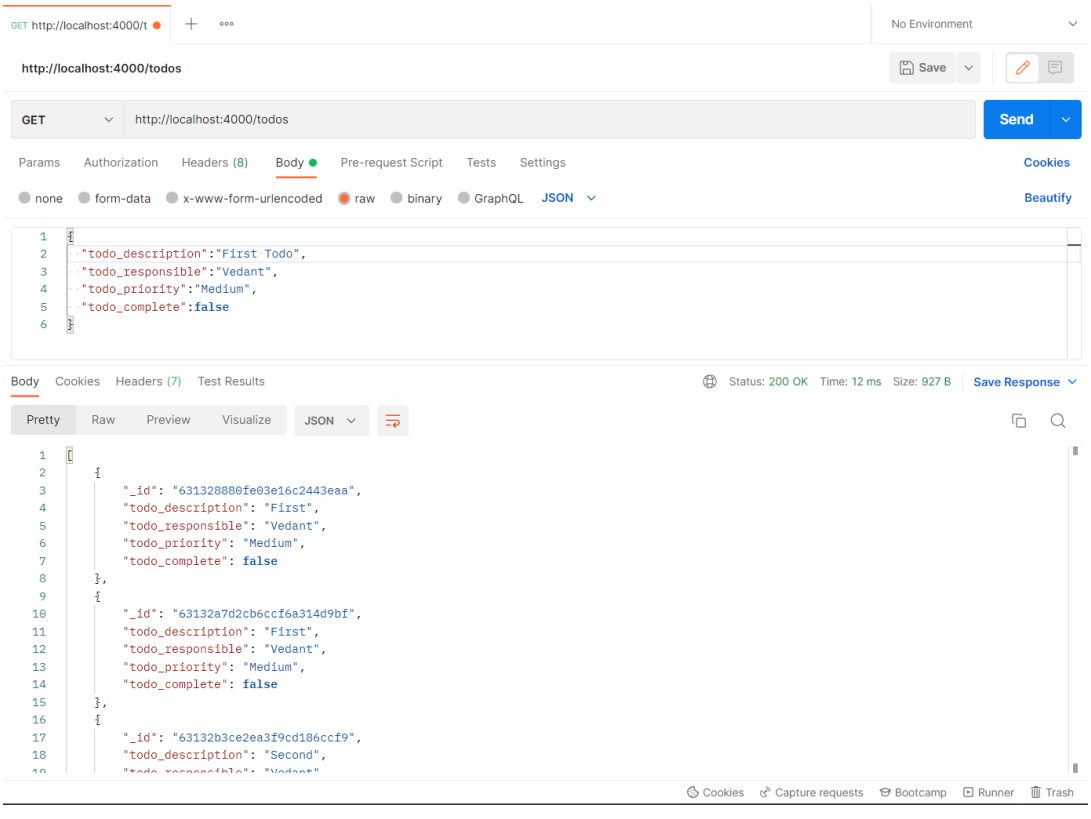
```
const express = require("express");
const app = express();
const { MongoClient, ObjectId } = require("mongodb");
const PORT = 4000;
```

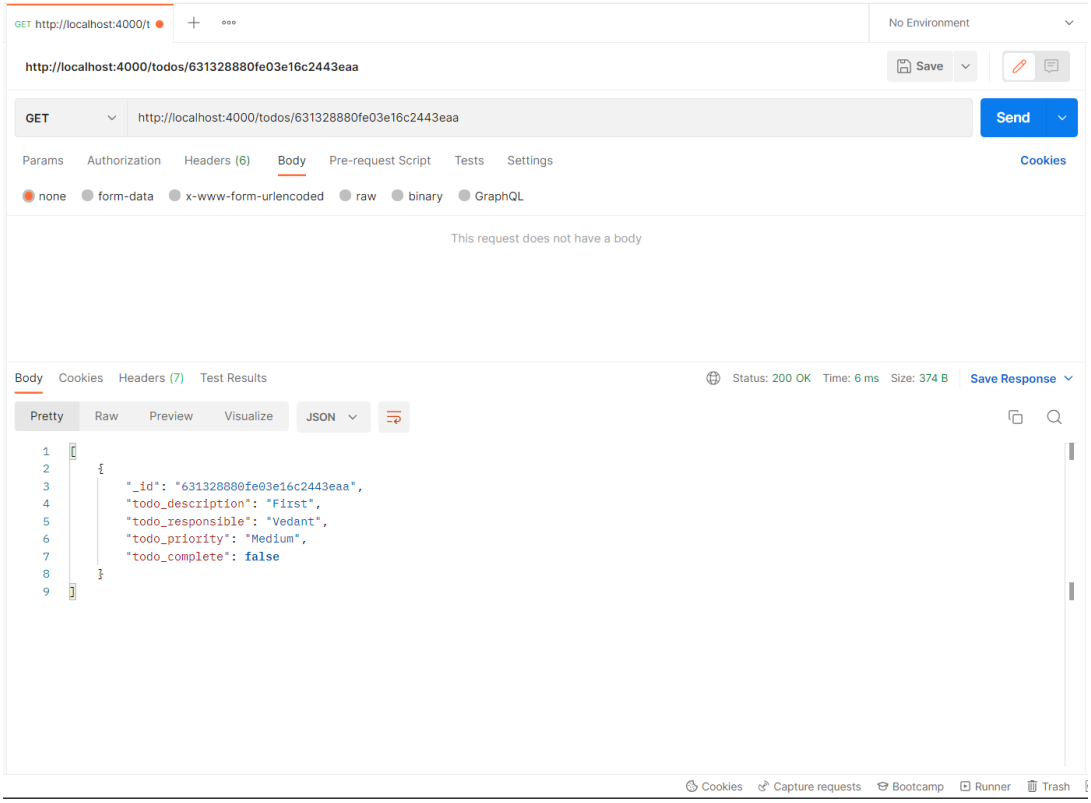
```
let db;

let connectionString = `mongodb://localhost:27017/crud`;

app.use(express.json());

const todoRoutes = express.Router();

todoRoutes.route("/").get(function (req, res) {
    db.collection("todos")
        .find()
        .toArray(function (err, items) {
            res.send(items);
        });
});

todoRoutes.route("/:id").get(function (req, res) {
    let id = req.params.id;
    db.collection("todos")
        .find({ _id: id })
        .toArray(function (err, items) {
            res.send(items);
        });
});

todoRoutes.route("/add").post(function (req, res) {
    db.collection("todos").insertOne(req.body, function (err, info) {
        res.json(info);
    });
});

todoRoutes.route("/update/:id").put(function (req, res) {
    db.collection("todos").findOneAndUpdate(
        { _id: new ObjectId(req.params.id) },
        {
            $set: {
                todo_description: req.body.todo_description,
                todo_responsible: req.body.todo_responsible,
                todo_priority: req.body.todo_priority,
                todo_complete: req.body.todo_complete,
            },
        },
        function () {
            res.send("Success updated!");
        },
    );
});

todoRoutes.route("/delete/:id").delete(function (req, res) {
    db.collection("todos").deleteOne(
        { _id: new ObjectId(req.params.id) },
        function () {
            res.send("Successfully deleted!");
        },
    );
});

app.use("/todos", todoRoutes);

MongoClient.connect(
    connectionString,
    { useNewUrlParser: true },
    (error, client) => {
        if (error) {
            return console.log("Connection failed for some reason");
```

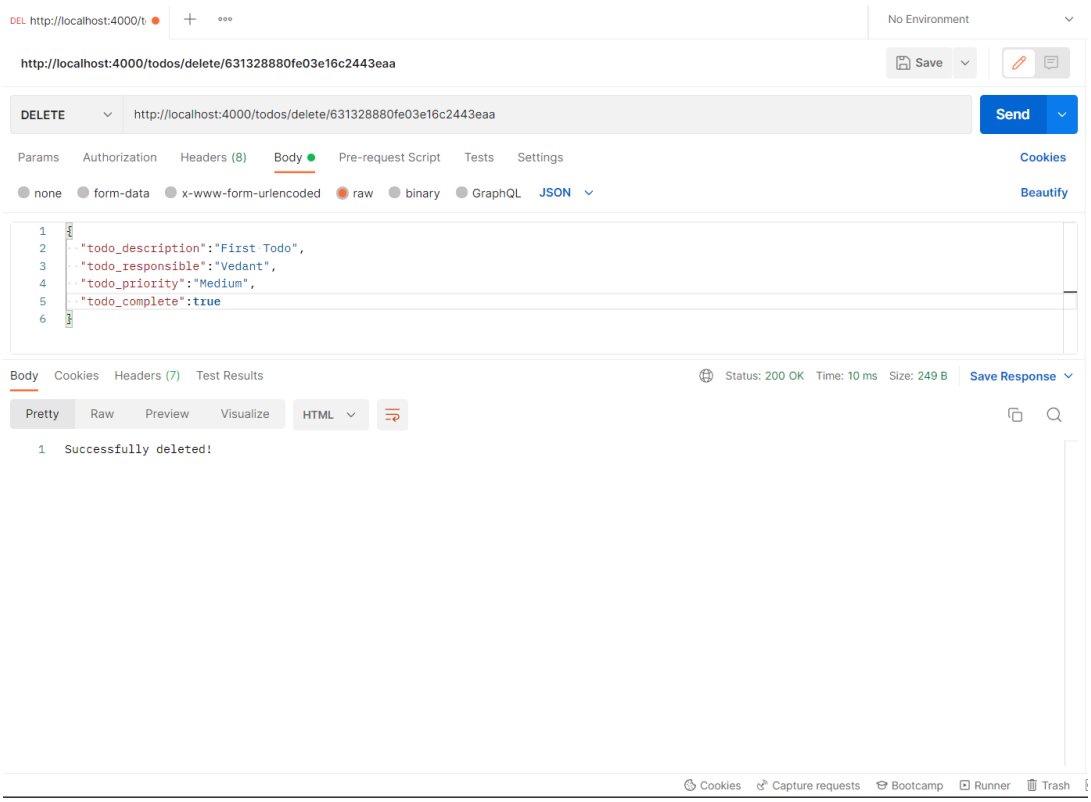
```
        }
        console.log("Connection established - All well");
        db = client.db("crud");
        app.listen(PORT, function () {
            console.log("Server is running on Port: " + PORT);
        });
    },
);
```

Testing The Server API With Postman









Part 3 - Connecting Front-End To Back-End

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 la 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.js* so that we're ready to use the Axios library in that file:

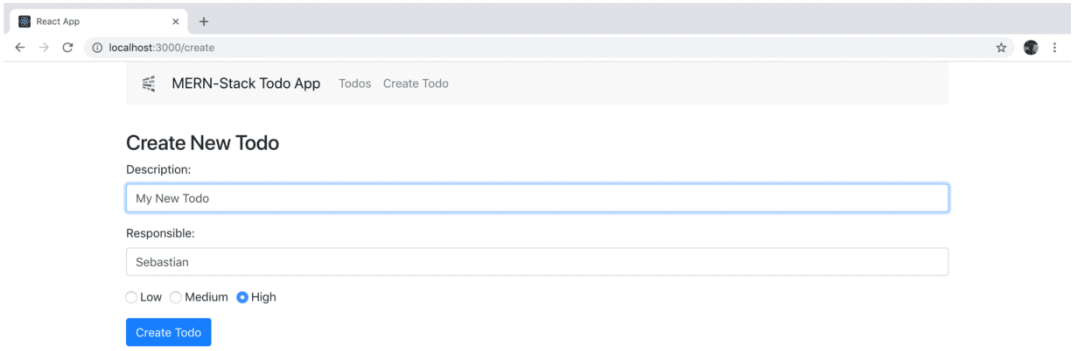
```
import axios from 'axios';
```

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 the *onSubmit* method. The existing implementation of *onSubmit* needs to be extended in the following way:

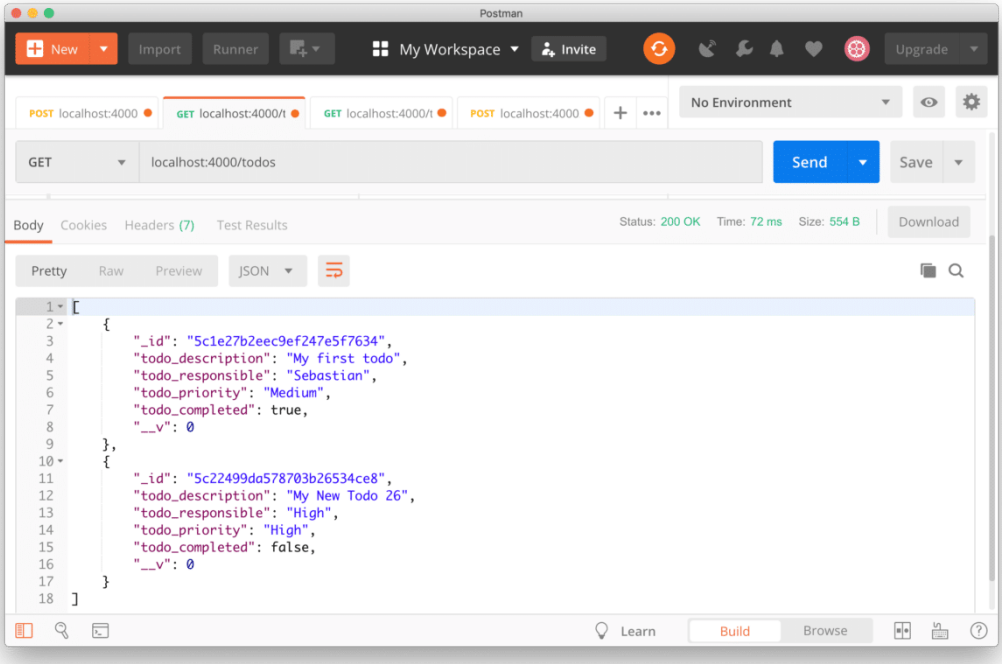
```
onSubmit(e) {
        e.preventDefault();

        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 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:



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 MongoD 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 th implementation of *TodosList* component in the next step.

Completing The Implementation Of TodosList Component:

In *todos-list.component.js* 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 and componentDidUpdate* lifecycle method is added:

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

componentDidUpdate() {
        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* 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>
        )
    }
```

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 not yet implemented. The current todo item is assigned to the todo property of this component.

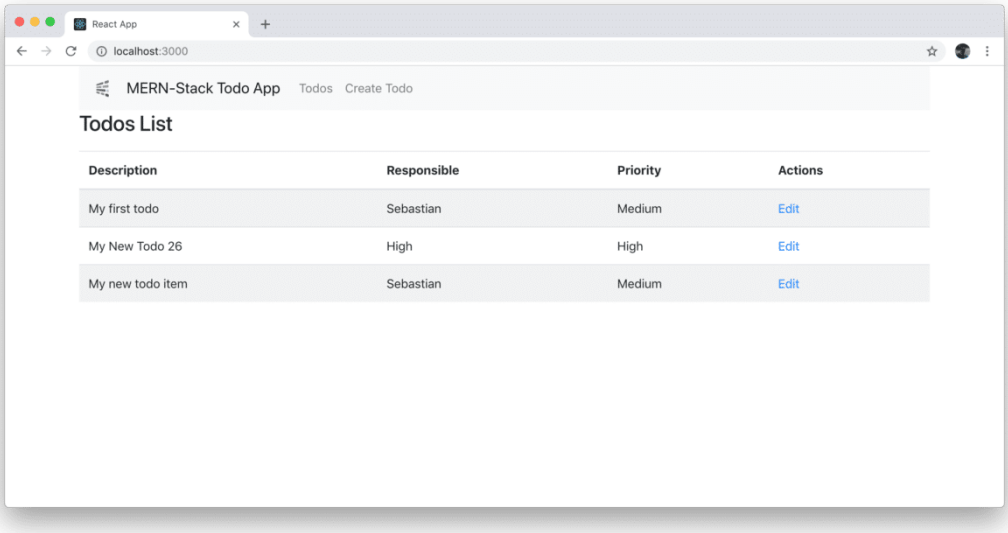
To complete the code in *todos-list.component.js* we need to add the implementation of Todo component as well. In the following listing you can see th complete code:

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

const Todo = props => (
    <tr>
        <td className={props.todo.todo_complete ? "completed" : ""}>
            {props.todo.todo_description}
        </td>
        <td className={props.todo.todo_complete ? "completed" : ""}>
            {props.todo.todo_responsible}
        </td>
        <td className={props.todo.todo_complete ? "completed" : ""}>
            {props.todo.todo_priority}
        </td>
        <td>
            <Link to={"/edit/" + props.todo._id}>Edit</Link>
            <Link
                style={{ marginLeft: "10px" }}
                to='/'
                onClick={() =>
                    axios
                        .delete(`http://localhost:4000/todos/delete/${props.todo._id}`)
                        .then(() => window.location.reload())
                }>
                Delete
            </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);
            });
    }

    componentDidUpdate() {
        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>Actions</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 ite 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:



Part 4 - Finishing The Application

Linking To EditTodo Component:

The link to EditTodo component has already been included in the output which returned by Todo component (implemented in *todos-list.component.js*):

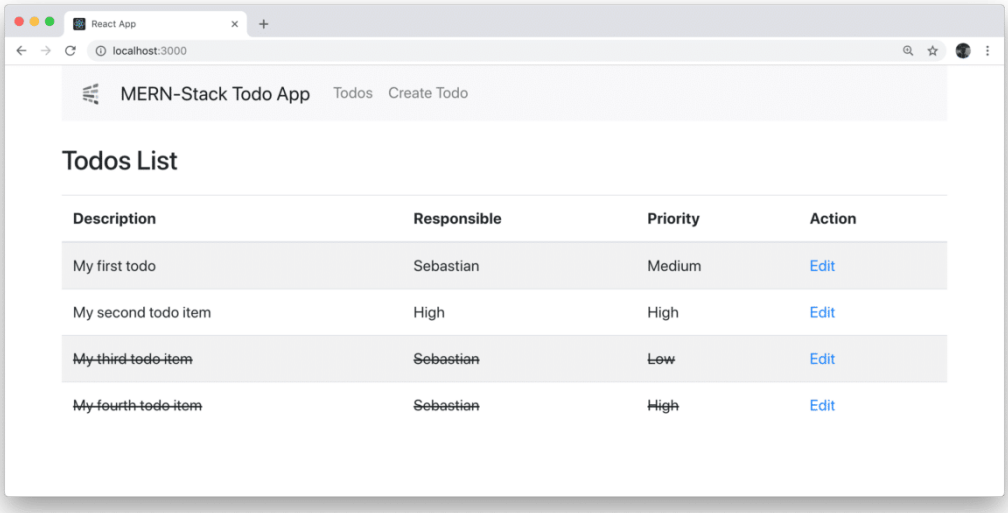
```
const Todo = props => (
    <tr>
        <td className={props.todo.todo_completed ? 'completed' : ''}>{props.todo.todo_description}</td>
        <td className={props.todo.todo_completed ? 'completed' : ''}>{props.todo.todo_responsible}</td>
        <td className={props.todo.todo_completed ? 'completed' : ''}>{props.todo.todo_priority}</td>
        <td>
            <Link to={"/edit/"+props.todo._id}>Edit</Link>
        </td>
    </tr>
)
```

The Edit link which is output for each todo entry is pointing to path */edit/:id*. The ID of the current todo is included in the URL so that we're able to retriev the current ID in the implementation of *EditTodo* component again.

Furthermore you can see that depending on the *todo_completed* value of a todo to completed CSS class is applied (if *todo_completed* is true). The CS class is added in index.css:

```
.completed {
  text-decoration: line-through;
}
```

By applying this CSS class the text information of a todo item is displayed with crossed out text like you can see in the following:



Edit Todos:

Ok, so let's turn to the implementation of *EditTodo* component in *edit-todo.component.js*. First of all the Axios library needs to be made available b adding the following import statement:

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

We would also need to create a function that will wrap our Edit component so that we can access route params in it:

```
const withRouter = WrappedComponent => props => {
    const params = useParams();
    // etc... other react-router-dom v6 hooks

    return (
        <WrappedComponent
            {...props}
            params={params}
            // etc...
        />
    );
};
```

Next we're adding a class constructor to set the initial state:

```
constructor(props) {
        super(props);

        this.state = {
            todo_description: '',
            todo_responsible: '',
            todo_priority: '',
            todo_complete: false
        }
    }
```

The state object is consisting of four properties which are representing one single todo item. To retrieve the current todo item (based on it's ID) from th back-end and update the component's state accordingly the *componentDidMount* lifecycle method is added in the following way:

```
componentDidMount() {
        axios.get('http://localhost:4000/todos/'+this.props.params.id)
            .then(response => {
                this.setState({
                    todo_description: response.data[0].todo_description,
                    todo_responsible: response.data[0].todo_responsible,
                    todo_priority: response.data[0].todo_priority,
```

```
                todo_complete: response.data[0].todo_complete
            })
        })
        .catch(function (error) {
            console.log(error);
        })
    }
```

Here we're making use of Axios once again to send an HTTP GET request to the back-end in order to retrieve todo information. Because we've bee handing over the ID as a URL parameter we're able to access this information via this.props.match.params.id so that we're able to pass on th information to the back-end.

The response which is returned from the back-end is the todo item the user has requested to edit. Once the result is available we're setting th component's state again with the values from the todo item received.

With the state containing the information of the todo item which has been selected to be edited we're now ready to output the form, so that the user able to see what's available and is also able to use the form to alter data. As always the corresponding JSX code needs to be added the *return* statement of the component's *render* method.

```
render() {
        return (
            <div>
                <h3 align="center">Update Todo</h3>
                <form onSubmit={this.onSubmit}>
                    <div className="form-group">
                        <label>Description: </label>
                        <input  type="text"
                                className="form-control"
                                value={this.state.todo_description}
                                onChange={this.onChangeTodoDescription}
                                />
                    </div>
                    <div className="form-group">
                        <label>Responsible: </label>
                        <input
                                type="text"
                                className="form-control"
                                value={this.state.todo_responsible}
                                onChange={this.onChangeTodoResponsible}
                                />
                    </div>
                    <div className="form-group">
                        <div className="form-check form-check-inline">
                            <input  className="form-check-input"
                                    type="radio"
                                    name="priorityOptions"
                                    id="priorityLow"
                                    value="Low"
                                    checked={this.state.todo_priority==='Low'}
                                    onChange={this.onChangeTodoPriority}
                                    />
                            <label className="form-check-label">Low</label>
                        </div>
                        <div className="form-check form-check-inline">
                            <input  className="form-check-input"
                                    type="radio"
                                    name="priorityOptions"
                                    id="priorityMedium"
                                    value="Medium"
                                    checked={this.state.todo_priority==='Medium'}
                                    onChange={this.onChangeTodoPriority}
                                    />
                            <label className="form-check-label">Medium</label>
                        </div>
                        <div className="form-check form-check-inline">
                            <input  className="form-check-input"
```

```
                                type="radio"
                                name="priorityOptions"
                                id="priorityHigh"
                                value="High"
                                checked={this.state.todo_priority==='High'}
                                onChange={this.onChangeTodoPriority}
                                />
                            <label className="form-check-label">High</label>
                        </div>
                    </div>
                    <div className="form-check">
                        <input  className="form-check-input"
                                id="completedCheckbox"
                                type="checkbox"
                                name="completedCheckbox"
                                onChange={this.onChangeTodoCompleted}
                                checked={this.state.todo_complete}
                                value={this.state.todo_complete}
                                />
                        <label className="form-check-label" htmlFor="completedCheckbox">
                            Completed
                        </label>
                    </div>

                    <br />

                    <div className="form-group">
                        <input type="submit" value="Update Todo" className="btn btn-primary" />
                    </div>
                </form>
            </div>
        )
    }
```

Herewith the following form is generated:

The form is using several event handler methods which are connected to the onChange event types of the input controls:

- *onChangeTodoDescription*
- *onChangeTodoResponsible*
- *onChangeTodoPriority*
- *onChangeTodoCompleted*

Furthermore the *submit* event of the form is bound to the *onSubmit* event handler method of the component.

The four *onChange* event handler methods are making sure that the state of the component is update everytime the user changes the input values of th form controls:

```
onChangeTodoDescription(e) {
        this.setState({
            todo_description: e.target.value,
        });
    }

    onChangeTodoResponsible(e) {
        this.setState({
            todo_responsible: e.target.value,
        });
    }

    onChangeTodoPriority(e) {
        this.setState({
            todo_priority: e.target.value,
        });
    }
```

```
    onChangeTodoCompleted(e) {
        this.setState({
            todo_complete: !this.state.todo_complete,
        });
    }
```

The onSubmit event handler method is creating a new todo object based on the values available in the component's state and then initiating a po request to the back-end endpoint http://localhost:4000/todos/update/:id to create a new todo item in the MongoDB database:

```
onSubmit(e) {
        e.preventDefault();
        const obj = {
            todo_description: this.state.todo_description,
            todo_responsible: this.state.todo_responsible,
            todo_priority: this.state.todo_priority,
            todo_complete: this.state.todo_complete
        };
        console.log(obj);
        axios.post('http://localhost:4000/todos/update/'+this.props.params.id, obj)
            .then(res => console.log(res.data));

        this.props.navigate('/');
    }
```

By calling *this.props.history.push('/')* it is also made sure that the user is redirected back to the default route of the application, so that the list of todos shown again.

Because we're accessing the component's state (this.state) in the event handler method we need to create a lexcial binding to this for all five methods the constructor:

```
constructor(props) {
        super(props);

        this.onChangeTodoDescription = this.onChangeTodoDescription.bind(this);
        this.onChangeTodoResponsible = this.onChangeTodoResponsible.bind(this);
        this.onChangeTodoPriority = this.onChangeTodoPriority.bind(this);
        this.onChangeTodoCompleted = this.onChangeTodoCompleted.bind(this);
        this.onSubmit = this.onSubmit.bind(this);

        this.state = {
            todo_description: '',
            todo_responsible: '',
            todo_priority: '',
            todo_complete: false
        }
    }
```

With these code changes in place we now should have a fully working MERN-stack application which allows us to:

- View a list of todo items
- Create new todo items
- Update existing todo items
- Set todo items to status completed

Finally, take a look at the following listing. Here you can see the complete and final code of *edit-todo.component.js* again:

```
import axios from "axios";
import React, { Component } from "react";
import { useNavigate, useParams } from "react-router";

const withRouter = WrappedComponent => props => {
    const params = useParams();
    const navigate = useNavigate();
```

```
    return <WrappedComponent {...props} params={params} navigate={navigate} />;
};

class EditTodo extends Component {
    constructor(props) {
        super(props);

        this.onChangeTodoDescription = this.onChangeTodoDescription.bind(this);
        this.onChangeTodoResponsible = this.onChangeTodoResponsible.bind(this);
        this.onChangeTodoPriority = this.onChangeTodoPriority.bind(this);
        this.onChangeTodoCompleted = this.onChangeTodoCompleted.bind(this);
        this.onSubmit = this.onSubmit.bind(this);

        this.state = {
            todo_description: "",
            todo_responsible: "",
            todo_priority: "",
            todo_complete: false,
        };
    }

    componentDidMount() {
        console.log(this.props);
        axios
            .get("http://localhost:4000/todos/" + this.props.params.id)
            .then(response => {
                this.setState({
                    todo_description: response.data[0].todo_description,
                    todo_responsible: response.data[0].todo_responsible,
                    todo_priority: response.data[0].todo_priority,
                    todo_complete: response.data[0].todo_complete,
                });
            })
            .catch(function (error) {
                console.log(error);
            });
    }

    onChangeTodoDescription(e) {
        this.setState({
            todo_description: e.target.value,
        });
    }

    onChangeTodoResponsible(e) {
        this.setState({
            todo_responsible: e.target.value,
        });
    }

    onChangeTodoPriority(e) {
        this.setState({
            todo_priority: e.target.value,
        });
    }

    onChangeTodoCompleted(e) {
        this.setState({
            todo_complete: !this.state.todo_complete,
        });
    }

    onSubmit(e) {
        e.preventDefault();
```

```
    const obj = {
        todo_description: this.state.todo_description,
        todo_responsible: this.state.todo_responsible,
        todo_priority: this.state.todo_priority,
        todo_complete: this.state.todo_complete,
    };
    console.log(obj);
    axios
        .put("http://localhost:4000/todos/update/" + this.props.params.id, obj)
        .then(res => console.log(res.data));

    this.props.navigate("/");
}

render() {
    return (
        <div>
            <h3 align='center'>Update Todo</h3>
            <form onSubmit={this.onSubmit}>
                <div className='form-group'>
                    <label>Description: </label>
                    <input
                        type='text'
                        className='form-control'
                        value={this.state.todo_description}
                        onChange={this.onChangeTodoDescription}
                    />
                </div>
                <div className='form-group'>
                    <label>Responsible: </label>
                    <input
                        type='text'
                        className='form-control'
                        value={this.state.todo_responsible}
                        onChange={this.onChangeTodoResponsible}
                    />
                </div>
                <div className='form-group'>
                    <div className='form-check form-check-inline'>
                        <input
                            className='form-check-input'
                            type='radio'
                            name='priorityOptions'
                            id='priorityLow'
                            value='Low'
                            checked={this.state.todo_priority === "Low"}
                            onChange={this.onChangeTodoPriority}
                        />
                        <label className='form-check-label'>Low</label>
                    </div>
                    <div className='form-check form-check-inline'>
                        <input
                            className='form-check-input'
                            type='radio'
                            name='priorityOptions'
                            id='priorityMedium'
                            value='Medium'
                            checked={this.state.todo_priority === "Medium"}
                            onChange={this.onChangeTodoPriority}
                        />
                        <label className='form-check-label'>Medium</label>
                    </div>
                    <div className='form-check form-check-inline'>
                        <input
                            className='form-check-input'
```

```
                                type='radio'
                                name='priorityOptions'
                                id='priorityHigh'
                                value='High'
                                checked={this.state.todo_priority === "High"}
                                onChange={this.onChangeTodoPriority}
                            />
                            <label className='form-check-label'>High</label>
                        </div>
                    </div>
                    <div className='form-check'>
                        <input
                            className='form-check-input'
                            id='completedCheckbox'
                            type='checkbox'
                            name='completedCheckbox'
                            onChange={this.onChangeTodoCompleted}
                            checked={this.state.todo_complete}
                            value={this.state.todo_complete}
                        />
                        <label className='form-check-label' htmlFor='completedCheckbox'>
                            Completed
                        </label>
                    </div>

                    <br />

                    <div className='form-group'>
                        <input
                            type='submit'
                            value='Update Todo'
                            className='btn btn-primary'
                        />
                    </div>
                </form>
            </div>
        );
    }
}

export default withRouter(EditTodo);
```

Conclusion

The MERN stack combines MongoDB, Express, React and Node.js for back- and front-end web development. This four-part module provided you with practical introduction to building a MERN stack application from start to finish. By building a simple todo manager application you've learnt how th various building blocks of the MERN stack are fitting together and are applied in a practical real-world application.

Final Project Link - Project-Link

Thank You !