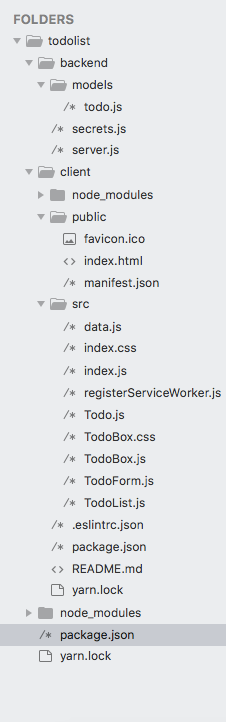
Initialisation of working directory

*$ mkdir mern-todo && cd mern-todo*

*$ npm i -g create-react-app*

*$ create-react-app client*

*$ mkdir backend && mkdir backend/models && touch backend/models/comment.js && touch backend/server.js*

Getting components for backend:

*$ yarn add express body-parser nodemon morgan mongoose concurrently babel-cli babel-preset-es2015 babel-preset-stage-0*

Getting components for frontend:

*$ cd client*

//gotta clean some trash…

*$ rm src/logo.svg src/App.css src/App.test.js src/App.js*

//init main source folder

*$ cd src && touch TodoList.js TodoBox.js TodoForm.js Todo.js data.js TodoBox.css*

Start:

//index.js

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import TodoBox from './TodoBox';

import registerServiceWorker from './registerServiceWorker';

ReactDOM.render(<TodoBox />, document.getElementById(‘root'));

registerServiceWorker();

Service Worker - web API that helps you cache your assets and other files so that when the user is offline or on slow network, he/she can still see results on the screen, as such, it helps you build a better user experience, that's what you should know about service worker's for now. It's all about adding offline capabilities to your site.

*$ yarn add react-markdown whatwg-fetch prop-types*

react-markdown — to convert markdown to text

watwg-fetch — to fetch data from browser

prop-types — to ensure we are getting the expected type in to our component

*$ yarn add --dev eslint babel-eslint*

//for Unix:

$ (

export PKG=eslint-config-airbnb;

npm info "$PKG@latest" peerDependencies --json | command sed 's/[\{\},]//g ; s/: /@/g' | xargs yarn add --dev "$PKG@latest"

)

//for Windows: :::::))))0

//../client/src/ .eslintrc.json

{

"extends": "airbnb",

"plugins": [

"react",

"jsx-a11y",

"import"

],

"ecmaFeatures": {

"arrowFunctions": true

},

"env": {

"browser": true,

"node": true

},

"parser": "babel-eslint",

"rules": {

"no-use-before-define": [0],

"react/jsx-filename-extension": [0],

"jsx-a11y/anchor-is-valid": [0],

"no-confusing-arrow": [0],

"no-underscore-dangle": [0],

"import/no-extraneous-dependencies": [0]

}

}

Init (currently) static files:

//TodoBox.js

import React, { Component } from 'react';

import TodoList from './TodoList';

import TodoForm from './TodoForm';

import DATA from './data';

import './TodoBox.css';

class TodoBox extends Component {

constructor() {

super();

this.state = { data: [] };

}

render() {

return (

<div className="container">

<div className="todos">

<h2>Todos:</h2>

<TodoList data={DATA} />

</div>

<div className="form">

<TodoForm />

</div>

</div>

);

}

}

export default TodoBox;

//TodoList.js

import React from 'react';

import PropTypes from 'prop-types';

import Todo from './Todo';

const TodoList = (props) => {

const todoNodes = props.data.map(todo => (

<Todo

key={todo.\_id}

id={todo.\_id}

>

{ todo.text}

</Todo>

));

return (

<div>

{ todoNodes }

</div>

);

};

TodoList.propTypes = {

data: PropTypes.arrayOf(PropTypes.shape({

id: PropTypes.string,

text: PropTypes.string,

})),

};

TodoList.defaultProps = {

data: [],

};

export default TodoList;

//TodoForm.js

import React from 'react';

import PropTypes from 'prop-types';

const TodoForm = props => (

<form onSubmit={props.submitTodo}>

<input

type="text"

name="text"

placeholder="You need to..."

value={props.text}

onChange={props.handleChangeText}

/>

<button type="submit">Oh, yeah</button>

</form>

);

TodoForm.propTypes = {

submitTodo: PropTypes.func.isRequired,

handleChangeText: PropTypes.func.isRequired,

text: PropTypes.string,

};

TodoForm.defaultProps = {

text: '',

};

export default TodoForm;

//Todo.js

import React from 'react';

import PropTypes from 'prop-types';

import ReactMarkdown from 'react-markdown';

const Todo = props => (

<div className="singleTodo">

<div className="textContent">

<div className="singleTodoContent">

<ReactMarkdown source={props.children} />

<div className="singleTodoButtons">

</div>

</div>

</div>

</div>

);

Todo.propTypes = {

children: PropTypes.string.isRequired,

id: PropTypes.string.isRequired;

};

export default Todo;

//data.js

const data = [

{

\_id: 0,

text: Развернуть JSON Server,

},

{

\_id: 1,

text: Изучить Vue ,

}

];

export default data;

//TodoBox.css

.container {

background-color: #F7C7D3;

box-sizing: border-box;

padding: 25px;

min-height: 100vh;

width: 100vw;

display: flex;

flex-flow: column nowrap;

align-items: center;

justify-content: space-between;

padding-top: 10%;

}

.todos {

overflow: auto;

width: 75%;

max-width: 700px;

min-width: 300px;

}

.todos h2 {

font-weight: 300;

}

.singleTodo {

padding: 0 20px;

margin-bottom: 5px;

display: flex;

}

.singleTodoContent {

min-width: 100px;

background-color: #fff;

border-radius: 0 25px 25px 0;

padding: 10px;

display: inline-flex;

border: 1px solid #c1c1c1;

}

.singleTodoContent p {

font-size: 14px;

margin: 0;

}

.form {

width: 70%;

min-width: 300px;

max-width: 700px;

}

.form form {

display: flex;

padding: 15px 0;

}

.form input {

height: 30px;

padding-left: 10px;

border: 1px solid #c1c1c1;

margin-right: 10px;

border-radius: 20px;

flex: 3;

}

.form button {

color: #999;

border-radius: 5px;

border: 1px solid #999;

}

.form button:hover {

cursor: pointer;

color: #385997;

border-color: #385997;

}

*$ yarn start:client*

And client must work!

Some set up for server:

//mern-todo/package.json

script:

…

"start:server": "cd backend && nodemon server.js --exec babel-node --presets es2015,stage-0",

"start:client": "cd client && yarn start",

"start:dev": "concurrently \"yarn start:server\" \"yarn start:client\»"

…

//server.js

import express from 'express';

import bodyParser from 'body-parser';

import logger from 'morgan';

import mongoose from 'mongoose';

// and create our instances

const app = express();

const router = express.Router();

// set our port to either a predetermined port number if you have set it up, or 3001

const API\_PORT = process.env.API\_PORT || 3001;

// now we should configure the API to use bodyParser and look for JSON data in the request body

app.use(bodyParser.urlencoded({ extended: true }));

app.use(bodyParser.json());

app.use(logger('dev'));

// now we can set the route path & initialize the API

router.get('/', (req, res) => {

res.json({ message: 'Hello, World!' });

});

// Use our router configuration when we call /api

app.use('/api', router);

app.listen(API\_PORT, () => console.log(`Listening on port ${API\_PORT}`));

*$ yarn start:server*

*<http://localhost:3001/api/>*

*//message: «Hello, world!»*

// server.js

import { getSecret } from './secrets';

// ... removed for brevity

const API\_PORT = process.env.API\_PORT || 3001;

// db config -- set your URI from mLab in secrets.js

mongoose.connect(getSecret(‘dbUri’)); //mongo url

var db = mongoose.connection;

db.on('error', console.error.bind(console, 'MongoDB connection error:'));

//model//todo.js

import mongoose from 'mongoose';

const Schema = mongoose.Schema;

const TodosSchema = new Schema({

text: String,

},

);

// export our module to use in server.js

export default mongoose.model('Todo', TodosSchema);

// server.js

// ...

import Todo from ‘./models/todo';

// …

//server.js

router.get('/todos', (req, res) => {

Todo.find((err, todos) => {

if (err) return res.json({ success: false, error: err });

return res.json({ success: true, data: todos });

});

});

router.post('/todos', (req, res) => {

const todo = new Todo();

// body parser lets us use the req.body

const { text } = req.body;

todo.text = text;

todo.save(err => {

if (err) return res.json({ success: false, error: err });

return res.json({ success: true });

});

});

object destructuring: const { author, text } = req.body

This is basically just pulling those fields off the req.body object and creating two variables (author and text) from those values.

Now after saving nodemon restarts server and Postman will show empty data object.

// client/package.json

},

"proxy": "http://localhost:3001"

}

//TodoBox.js

import React, { Component } from 'react';

import 'whatwg-fetch';

import TodoList from './TodoList';

import TodoForm from './TodoForm';

import './TodoBox.css';

class TodoBox extends Component {

constructor() {

super();

this.state = {

data: [],

error: null,

todo: '',

updateId: null,

};

this.pollInterval = null;

}

componentDidMount() {

this.loadTodosFromServer();

if (!this.pollInterval) {

this.pollInterval = setInterval(this.loadTodosFromServer, 2000);

}

}

componentWillUnmount() {

if (this.pollInterval) clearInterval(this.pollInterval);

this.pollInterval = null;

}

loadTodosFromServer = () => {

fetch('/api/todos/')

.then(data => data.json())

.then((res) => {

if (!res.success) this.setState({ error: res.error });

else this.setState({ data: res.data });

});

}

render() {

return (

<div className="container">

<div className="todos">

<h2>To Do:</h2>

<TodoList

data={this.state.data}

/>

</div>

<div className="form">

<TodoForm

text={this.state.text}

/>

</div>

{this.state.error && <p>{this.state.error}</p>}

</div>

);

}

}

export default TodoBox;

Adding new task:

//TodoBox.js

submitTodo = (e) => {

e.preventDefault();

const { updateId } = this.state;

if (updateId) {

this.submitUpdatedTodo();

} else {

this.submitNewTodo();

}

}

submitNewTodo = () => {

const { text } = this.state;

const data = [...this.state.data, { text, \_id: Date.now().toString() }];

this.setState({ data });

fetch('/api/todos', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ text }),

}).then(res => res.json()).then((res) => {

if (!res.success) this.setState({ error: res.error.message || res.error });

else this.setState({ text: '', error: null });

});

}

render() {

return (

…

<div className="form">

<TodoForm

text={this.state.text}

handleChangeText={this.onChangeText}

submitTodo={this.submitTodo}

/>

….

);

}

}

We’re using a promise to post to our server. If you’re not familiar with promises, this is pretty much just an async function. It will start and finish at some later time. The .then fires and converts the response to json, the following .then takes that json and does what we want with it, either showing an error or clearing out our inputs.

Console Error: ‘Each child in an array or iterator should have a unique “key” prop…’

//TodoList.js

…

<Todo

key={todo.\_id}

id={todo.\_id}

>

…

Edit and delete using :params route

//server.js

…

router.put('/todos/:todoId', (req, res) => {

console.log(req.params);

const { todoId } = req.params;

if (!todoId) {

return res.json({ success: false, error: 'No todo id provided' });

}

Todo.findById(todoId, (error, todo) => {

if (error) return res.json({ success: false, error });

const { text } = req.body;

if (text) todo.text = text;

todo.save(error => {

if (error) return res.json({ success: false, error });

return res.json({ success: true });

});

});

});

router.delete('/todos/:todoId', (req, res) => {

const { todoId } = req.params;

if (!todoId) {

return res.json({ success: false, error: 'No todo id provided' });

}

Todo.remove({ \_id: todoId }, (error, todo) => {

if (error) return res.json({ success: false, error });

return res.json({ success: true });

});

});

…

//TodoBox.js

onUpdateTodo = (id) => {

const oldTodo = this.state.data.find(c => c.\_id === id);

if (!oldTodo) return;

this.setState({ text: oldTodo.text, updateId: id });

}

onDeleteTodo = (id) => {

const i = this.state.data.findIndex(c => c.\_id === id);

const data = [

...this.state.data.slice(0, i),

...this.state.data.slice(i + 1),

];

this.setState({ data });

fetch(`api/todos/${id}`, { method: 'DELETE' })

.then(res => res.json()).then((res) => {

if (!res.success) this.setState({ error: res.error });

});

}

submitNewTodo = () => {

const { text } = this.state;

const data = [...this.state.data, { text, \_id: Date.now().toString() }];

this.setState({ data });

fetch('/api/todos', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ text }),

}).then(res => res.json()).then((res) => {

if (!res.success) this.setState({ error: res.error.message || res.error });

else this.setState({ text: '', error: null });

});

}

submitUpdatedTodo = () => {

const { text, updateId } = this.state;

fetch(`/api/todos/${updateId}`, {

method: 'PUT',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ text }),

}).then(res => res.json()).then((res) => {

if (!res.success) this.setState({ error: res.error.message || res.error });

else this.setState({ text: '', updateId: null });

});

}

…

<TodoList

data={this.state.data}

handleDeleteTodo={this.onDeleteTodo}

handleUpdateTodo={this.onUpdateTodo}

/>

Now pass them into Todo.js through TodoList.js

//TodoList.js

import React from 'react';

import PropTypes from 'prop-types';

import Todo from './Todo';

const TodoList = (props) => {

const todoNodes = props.data.map(todo => (

<Todo

key={todo.\_id}

id={todo.\_id}

handleUpdateTodo={props.handleUpdateTodo}

handleDeleteTodo={props.handleDeleteTodo}

>

{ todo.text}

</Todo>

));

return (

<div>

{ todoNodes }

</div>

);

};

TodoList.propTypes = {

data: PropTypes.arrayOf(PropTypes.shape({

id: PropTypes.string,

text: PropTypes.string,

})),

handleDeleteTodo: PropTypes.func.isRequired,

handleUpdateTodo: PropTypes.func.isRequired,

};

TodoList.defaultProps = {

data: [],

};

export default TodoList;

Now Todo.js should look like:

//Todo.js

import React from 'react';

import PropTypes from 'prop-types';

import Todo from './Todo';

const TodoList = (props) => {

const todoNodes = props.data.map(todo => (

<Todo

key={todo.\_id}

id={todo.\_id}

handleUpdateTodo={props.handleUpdateTodo}

handleDeleteTodo={props.handleDeleteTodo}

>

{ todo.text}

</Todo>

));

return (

<div>

{ todoNodes }

</div>

);

};

TodoList.propTypes = {

data: PropTypes.arrayOf(PropTypes.shape({

id: PropTypes.string,

text: PropTypes.string,

})),

handleDeleteTodo: PropTypes.func.isRequired,

handleUpdateTodo: PropTypes.func.isRequired,

};

TodoList.defaultProps = {

data: [],

};

export default TodoList;

Welldone, boys, Grecha is proud of you.