FULLSTACK OPEN NOTES:

* Core Web Elements:
  + HTML file defines structure and elements.
  + CSS file defines the aesthetic qualities of the HTML elements.
  + Javascript(JS) files can re-render the HTML dynamically on the client browser without having to request a new HTML from the server.
  + Node.js backend technology on the server side.
* HTML:
* CSS:
* React Library:
  + JS library
  + Aims to separate the application by components rather than by content (HTML), style (CSS), and functionality (JS). To do this the HTML and CSS can be included with the component’s functionality in JS. These components should be as independent and reusable as possible.
  + Create a new react app:
    - Npx create-react-app <name of app>
  + When using react all contents to be rendered tend to be defined in react rather than HTML, although this is not a rule.
  + Must have an export default call at the bottom.
    - 
  + console.log() will print to the browser console. Great for debugging.
  + React components must be capitalized.
  + In ES6 functions are assigned to variables:
    - Text

      Description automatically generated with medium confidence
  + Code within curly braces is a reference to a variable.
    - Text

      Description automatically generated
    - A picture containing text

      Description automatically generated
  + The previous pieces of HTML-like code are in fact JSX and not HTML. JSX returned by react components is compiled into JS code under the hood. The above compiles to:
    - Text

      Description automatically generated
  + In JSX all tags must be closed. Ie. <br /> in JSX while in HTML it could be <br> since it is an empty tag (line break).
  + The root react component must wrap what it returns in a root tag, which is generally chosen to be an <div> tag or an empty tag <> as seen below. It can also return them as a list (in brackets []) without the outermost tag but this is seen as ugly and overly complex.
    - *Text

      Description automatically generated*
  + You can use a component inside the JSX returned by another component by using the following syntax: (Hello is its own JS function return JSX markup)
    - Text

      Description automatically generated with low confidence
  + Props:
    - Used to pass data to components.
    - It’s a catch all “class instance”-like argument for a component which can have many class parameters within it.
    - Text

      Description automatically generated
  + Helper Functions:
    - Can have internal functions within components defined with the same syntax as a normal component.
    - These internal helper functions have access to the same variables as its parent function.
    - Text

      Description automatically generated
  + Destructuring:
    - The props passed to a component can be split into its member variables. This can be done individually or in a single line using the following syntax.
    - 
    - It can even be done in the component declaration as follows:
    - 
  + Command Line Arguments
    - You can use command line arguments in a program by invoking the process.argv array.
      * 
  + Environmental Variables / Private Variables:
    - Variables that you want to keep private (API keys) should not be declared in the code that you will publicly upload to a repository.
    - They can be declared as local environmental variables which are loaded into the project at build time.
    - Env variable names must start with ‘REACT\_APP\_’ to be pulled into the project.
    - They can be accessed inside the JS files by using the ‘process.env.REACT\_APP\_VARIABLE\_NAME’ object.
      * 
    - By default there is a ‘process.env.NODE\_ENV’ variable which contains different values depending on if the project is built as development, test, or production.
    - More info at:
      * <https://create-react-app.dev/docs/adding-custom-environment-variables/>
    - To create an env variable through a shell
  + Re-Rendering:
    - To update an element after changes have been made to its props call the render method again as follows.
    - Text

      Description automatically generated
    - We can add a delay by using the setInterval method:
    - Text

      Description automatically generated
    - Re-rendering is better performed using stateful components than calling render directly.
  + Stateful Components:
    - State Hooks
      * Needs to import the ‘useState’ function from react.
      * Text

        Description automatically generated
      * Calls the useState function with an initial value (0 in this case) and deconstructs its output (returns an array [int, function reference]) into two variables:
      * 
      * Call the setTimeout function to call the setCounter function after one second (1000ms).
      * Text

        Description automatically generated
      * Every time the setCounter function is called the counter component is re-rendered automatically.
      * **Calling a function that changes the state causes the component to rerender.**
      * The full function looks as follows:
      * Text

        Description automatically generated
      * No for loops are needed since when the component gets re-rendered it is the whole App component.
    - Effect Hooks:
      * Must import the useEffect function from react:
        + 
      * Has 2 arguments the first being a function reference which will be the call back and the second being an array that specifies when to run the effect.
        + Text

          Description automatically generated
      * By default the effect is run after each render.
      * If the second argument is an empty array it will only run after the first render.
    - Rules of Hooks:
      * useState or useEffect functions must not be called from inside of a loop or conditional statement. These functions must always be called in the same order otherwise the program can behave erratically.
  + Event Handlers:
    - An event handler calls a function when an event occurs. In this case when a click occurs the handleClick function is called.
    - Text

      Description automatically generated
    - Handler functions can also be declared in line within the JSX inside the return statement.
    - Cant define the handler as follows since setCounter(counter+1) is a function call (because of the parenthesis and arguments), this will cause an error. This is because an event handler should be a function reference (ie. pointer) and if it is a call would cause an infinite loop since it gets called the first time its rendered and then re-renders ad infinitum:
      * 
    - It must be wrapped in another arrow function as follows:
      * 
    - When a piece of JSX with a component with an event handler is rendered the event handler statement is evaluated and its return value is set as its event handler. This is why a function call will not work unless it returns a function reference itself. A function that returns function references can be useful as a way to create custom event handlers for specific components dynamically.
    - It is a good idea to know what the default action of an event handler is and to immediately stop it if you want to override it with your own event handler:
      * 
  + Passing state to child components:
    - It is recommended to break up your React application into modular, small, self-contained components.
    - For example in the previous app the buttons and display functions can be broken out into their own components as follows:
      * Text

        Description automatically generated
    - The state is contained in the parent component and passed down to the children components as function references.
    - All children components called from within a parent component will be re-rendered when the parent is re-rendered.
    - **Do not define components within components. (Everytime the parent is re-rendered it creates a new unique child component creating bloat and making it hard for the React to optimize the application)**
  + Complex States:
    - Most of the time the best way of creating more complex states is to use multiple useState() calls in the same component to create separate pieces of state.
    - It is also possible to declare/change multiple pieces of state with a single call to useState():
      * Text

        Description automatically generated
    - And the function to update the clicks would be called like this:
      * Text

        Description automatically generated
    - This makes it tedious since every time we want to change one parameter we have to list them all out.
    - A way to make it simpler to write is to use the “**object spread**” syntax as follows:
      * Text

        Description automatically generated
    - The three dots copy all the parameters within the clicks variable except those that we explicitly re-write.
    - This makes a shallow copy of all the parameters we don’t change so if the parameters are objects themselves they are pointing to the same object.
    - **It is forbidden in React to change a state directly since it can have unintended side effects. To change a state you must create a new variable with the new state value and then pass that to the state setter function explicitly or implicitly(inline in the arguments).**
  + Conditional Rendering:
    - A component renders different things depending on the current state of a piece.
  + Debugging React:
    - You can type the keyword “debugger” in your program at any point and this will act like a breakpoint in the browser.
    - You can also add breakpoints in the traditional way within the sources tab of the developer console of your browser.
    - Variable contents will be displayed in the Scope section.
    - The React developer tools that can be added to your browser also add a Components tab that allows you to inspect components individually and their state and props. The state of the state hooks is shown in the same order as they were declared.
  + Module best practices:
    - It is recommended to declare each module as its own file as an ES6-module
    - The file should be named after the component
    - They should be placed within the ‘src/components/’ directory.
    - Each module file should end with a line exporting the declared module.
      * 
    - The file that is using a module must import it by calling import and providing the path to the module:
      * 
    - Modules that provide services rather than components should be stored in the “src/services” directory.
      * The functions within the services should be exported as follows:
        + Text

          Description automatically generated
        + Text

          Description automatically generated
      * The labels on the left are the keys to the objects (used from inside the caller module) whereas the ones on the right are variables that are defined inside the module.
      * When the keys and variables are the same names are the same you can simplify the syntax as follows.
      * Text

        Description automatically generated
  + Controlled Components:
    - An example of a controlled component with which the user can interact is a form:
      * Graphical user interface, text

        Description automatically generated
        + The value param is the initial value of the input field.
        + The onChange handler param is the handler function that updates the value variable everytime it is changed by the user.
        + Without an onChange handler the value in the input field is read only.
        + Its not necessary to call the even.preventDefault() method since there is no default method for an input change (always refer to the docs).
        + The handler function looks something like this:

Text

Description automatically generated

* + - * + The handler function for the onSubmit parameter relies on the newNote variable set by the handleNoteChange handler since its event argument does not contain the value of the string in the input component. It also resets the value of newNote to a blank string.

Text

Description automatically generated

* + Alert:
    - A rudimentary way to alert the user to an error or exception.
      * Text

        Description automatically generated with medium confidence
  + Inline Styles:
    - Any React component or element can be provided with a set of CSS properties as a JavaScript object through the [style](https://reactjs.org/docs/dom-elements.html#style) attribute.
      * In react the strings must be in quotes and the properties are camelCase rather than kebab-case.
      * Text

        Description automatically generated
    - Limitations: so called [pseudo-classes](https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes) cant be used straightforwardly unlike in CSS.
  + Project Structure:
    - Text

      Description automatically generated
    - The express app and the code taking care of the web server should be separated into individual files (index.js and app.js).
    - The handling of environmental variables should be extracted into a separate utils/config.js file.
      * Text

        Description automatically generated
    - Route handlers should also be separated into their module called at the ‘controllers/notes.js’ file. To declare routes separate from the express app you should use the express Route object:
      * Text

        Description automatically generated
      * Once imported they should be declared as middleware with a base path in the app.js file. All subsequent addresses for the routes declared in the notes.js file are now relative to the base path declare in the app.js file.
        + A screenshot of a computer

          Description automatically generated with medium confidence
    - Custom Middleware should also be separated into its own module and placed in its own inside the utils folder:
    - Mongoose models should be initialized inside the app.js file and their schemas should be declared in their own modules inside the ‘models’ folder.
  + Logging:
    - Extract into its own module as shown above.
    - Easier to maintain this way.
  + Flux-architecture:
    - A way to separate the state from the React components.
    - Graphical user interface, diagram, application, Teams

      Description automatically generated
  + Redux:
    - A simpler library implementation of the flux architecture.
    - Install:
      * 
    - Store:
      * The whole state of the app is stored in a single JavaScript object in the store.
      * Different things in the state are saved as individual object fields in the store.
      * 
      * 
      * counterReducer is a reducer function.
    - Actions:
      * The state of the store is changed with actions.
      * Actions are objects which have at least one field specifying the type of the action.
      * Text

        Description automatically generated
      * If there is data associated with the action it can also be stored as separate object fields.
      * The data is usually named ‘payload’:
      * Text

        Description automatically generated
      * Usually action creator can be abstracted into their own function.
      * Text

        Description automatically generated
      * 
    - Reducers:
      * The impact of the action on the application is defined using a reducer which is a function that is given the current state and an action as parameters. It returns a new state.
      * Text

        Description automatically generated
      * The reducer returns a new state based on the action type.
      * Generally, use a switch statement in a reducer.
      * Reducers should not change state variables but create and return new state variables.
      * Define a default value for state in case it hasn’t been initialized yet.
      * Reducers are never supposed to be called directly from the app.
      * Reducers are given as parameters to the createStore(deprecated) or configureStore function that creates a store.
      * Text

        Description automatically generated
      * Multiple reducers can be declared and then merged using the combineReducers function:
      * Text

        Description automatically generated
    - Dispatcher:
      * Actions are dispatched or sent to the store using the dispatch method.
      * 
      * You can retrieve the state of the store using the method getState.
      * 
    - Subscriber:
      * The subscribe method can be used to create a callback function the store calls whenever an action is dispatched to the store.
      * Text

        Description automatically generated
    - Hooks API react-redux library
      * 
      * Include the main App component inside of a Provider tag with the store attribute defined.
        + A screenshot of a computer

          Description automatically generated with medium confidence
      * App must pass the store as a prop to all sub components.
      * The store is accessed using the useSelector hook.
        + Text

          Description automatically generated
    - Redux-toolkit:
      * 
      * Includes the configureStore function which is preferred over the createStore function. This removes the need for the combineReducers function.
      * Text

        Description automatically generated
      * Additionally reducers and action creators can be easily generated using createSlice function.
      * Text

        Description automatically generated
      * You can use the reducers declared in the slice as follows:
      * 
      * The slice also contains automatically generated actions based on the reducers that were declared. These actions should be exported as follows.
      * Graphical user interface, text

        Description automatically generated
      * The actions should then be imported and used as follows.
      * 
      * When using the createSlice function you can simply mutate the state and the Immer library will make a new immutable object behind the scenes or you can return a new state object like we’ve done before.
    - Redux-Thunk:
      * <https://github.com/reduxjs/redux-thunk>
      * Enables implementation of action creators which return a function instead of an object.
      * Enables the use of asynchronous action creators which wait for the completion of a task and then dispatch an action.
      * Text

        Description automatically generated
      * Thunk action creators can also take a second getState argument which reads the current state in the store.
      * From there individual states can be accessed with selectors:
      * https://redux.js.org/usage/deriving-data-selectors
  + React Query
    - A library to store and manage data retrieved from the server
    - 
    - To use React query import the following components into the index.js file.
    - A screenshot of a computer

      Description automatically generated with medium confidence
    - Still uses axios to perform the request wrapped inside a useQuey method.
    - The first parameter of the function call is a string ‘notes’ which acts as a key to the query defined
    - Text

      Description automatically generated
    - The return value of the query is an object which represents the status of the query.
    - Mutations are used to modify existing objects on the server. <https://tanstack.com/query/latest/docs/react/guides/mutations>
    - Use the useMutation function to declare a mutation using React Query.
    - Graphical user interface, text

      Description automatically generated
    - Text

      Description automatically generated
    - A screenshot of a computer

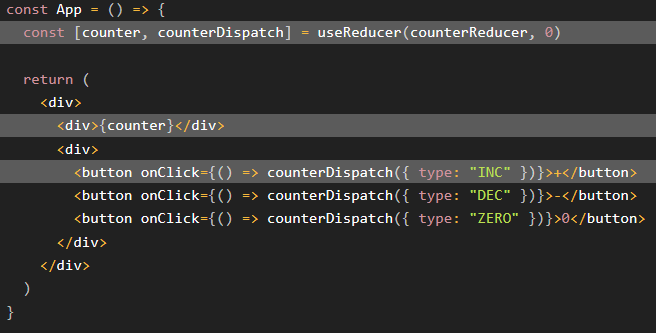
      Description automatically generated with medium confidence
    - The mutation is defined by making calling the useMutation function. The passed parameter is a functions which posts a new object to the backend.
    - By calling the .mutate method with an object as an argument, the object is added to the backend.
    - To update the object on the screen you must invalidate the previous object when a new note is added, this can be done by declaring the onSuccess calback in the useMutation functions arguments.
    - Text

      Description automatically generated
    - Calling the .invalidateQueries(‘note’) method causes ReactQuery to invalidate and update the query with the key ‘notes’.
    - You can also update the objects in the backend through the mutate function by passing a function that performs a put request as an argument to useMutation.
    - 
    - Text

      Description automatically generated
  + Optimizing performance usings ReactQuery:
    - Instead of invalidating the query every time an addition is made which causes a refeching of all the data from the server, the ReactQuery object can be updated manually.
    - Text

      Description automatically generated
    - By default ReactQuery will refetch queries whose status is ‘stale’ when the window focus (the active element of the app) changes, this behaviour may be undesirable.
    - This functionality can be disabled by creating a query with the following options:
    - Graphical user interface, text

      Description automatically generated
    - ReactQuery is a library that mantains the server state in the frontend and can in cases eliminate the need for the data on the server to be saved in the state of the front end.
    - Redux on the other hand can be used to maintain the state of the frontend and sometimes the server data on the frontend.
  + UseReducer
    - useReducer is a built-in react hook that provides a Redux like way of managing state.
    - Text

      Description automatically generated
    - The parameters for creating a state is the reducer function that handles state changes and the initial value of the state as seen above.
    - To use the change state functions passed they can be called as follows.
    - 
    - You can also include additional data like in Redux using the payload field of the action parameter in the Reducer function.
  + Passing state context to components
    - State change reducers can be passed to components as props but when stage management is complex the amount of props being passed to components increases as they call their own components etc. This is called prop-drilling.
    - React has a built in Context API which provides a different better method to pass state context down the DOM.
    - React’s context unifies the state reducers and makes them accessible to components.
    - Text

      Description automatically generated
    - Text

      Description automatically generated
    - The context provider value is set to be an array containing the value of the counter and its dispatch(reducer) function.
    - The components can now access the context like so.
    - Text

      Description automatically generated
    - Everything related to the state and context of one state can be put in its own file and the countercontext object can be exported directly.
    - Text

      Description automatically generated
    - Now it can be used simply by doing this
    - Text

      Description automatically generated
    - Not all components require all parts of the context so we can isolate the context components by declaring some helper functions inside of that Context’s JS file.
    - Text

      Description automatically generated
    - Then the context value and dispatcher can be requested individually by the components.
    - Graphical user interface, text, chat or text message

      Description automatically generated
    - Text

      Description automatically generated
  + When to use what state management method?
    - useState, useEffect, useReducer and other built in React Hooks:
      * Basis and perfectly serviceable.
      * Good for simple apps or when a state is self contained in a single component.
    - ReactQuery:
      * When a state in the front end need to be kept in sync with data in the backend.
    - Redux:
      * Oldest and most complex/Rigid but also most comprehensive.
      * Better and easier to use now with the ReduxToolkit.
    - \*\*Any of the above can be used concurrently in the same app although it is preferable to streamline the tech stack.
  + Uncontrolled Forms:
    - <https://goshakkk.name/controlled-vs-uncontrolled-inputs-react/>
    - The state of the app is not bound to the state of the form fields.
    - A screenshot of a computer

      Description automatically generated with medium confidence
    - The value of the field can be found using the event.target.note.value field where note is the name of the input form.
  + React Router:
    - Library for managing navigation in a react app.
    - 
    - Use the Router (BrowserRouter) component to create routes to the components you want.
    - All paths are relative and will be appended to the site at which the main app is being hosted.
    - Text

      Description automatically generated
    - We can define a parametized route by doing the following:
    - A screenshot of a computer

      Description automatically generated with medium confidence
    - We can access information (ie. id) about the current address of the app by using the useParams() function as seen bellow
    - Text

      Description automatically generated
    - We can also keep a user state and if the user has logged in use conditional rendering to remove the log in link.
    - Text

      Description automatically generated
    - We can utilize the hook useNavigate() to programmatically change the router’s URL.
    - Text

      Description automatically generated
    - We can implement redirects so that route conditionally redirects to a different route if some condition is not met. To do this we can use Navigate.
    - 
    - We can use the useMatch function as a conditional to verify if we are in a current address.
    - Text

      Description automatically generated
  + Custom Hooks:
    - React uses 15 built in hooks <https://reactjs.org/docs/hooks-reference.html>
    - Don’t call hooks inside loops conditions or nested functions.
    - Don’t call hooks from regular JS functions, only from react function components.
    - You can call hooks from custom hooks.
    - Custom hooks can facilitate the reuse of logic inside components.
    - Custom hooks are regular JS functions that can use any other hooks.
    - The name of a custom hook must start with the word ‘use’.
    - This is an example of a custom hook to manage an input field.
    - Text

      Description automatically generated
    - This custom hook can be implemented is a program like so:
    - Text

      Description automatically generated
  + React-bootstrap:
    - Frontend framework with pre-made components.
    - 
    - Add a link for loading the CSS style sheet for bootstrap inside of the head tag in your index.html file.
    - Text

      Description automatically generated
    - In bootstrap all of the contents of the app are rendered inside a container. Declare it by giving the root div element of the app the container class attribute.
    - Text

      Description automatically generated
    - React bootstrap components have to be imported individually from react-bootstrap
    - 
    - React bootstrap components documentation <https://react-bootstrap.github.io/getting-started/introduction/>
  + Material UI
    - An implementation of googles material design language In react.
    - 
    - Render the contents in a container similarly to bootstrap.
    - Text

      Description automatically generated
    - Every component in MaterialUI has to be imported separately.
    - A picture containing text

      Description automatically generated
    - MUI documentation; <https://mui.com/material-ui/getting-started/overview/>
  + Other UI frameworks:
    - Text

      Description automatically generated
  + Styled components:
    - Another way to declare the style for components.
    - 
    - Declare the CSS rules in backticks and assign them to objects.
    - Text

      Description automatically generated
    - Can be used as normal components.
    - A screenshot of a computer

      Description automatically generated with medium confidence
* Webpack:
  + Code that is divided into modules must be bundled for browsers. This creates a single file that contains all of the app code (including the code for dependencies).
  + Usually you can use the npm run build command when using npm and it will generate the following file structure.
  + A black background with white text

    Description automatically generated with low confidence
  + The entry point becomes the index.html file containing the following which loads the bindled js file using the script tag. It also loads its associated bundled css file.
  + Text

    Description automatically generated
  + Bundling is done to minimize the overhead in loading multiple css and js files.
  + To create a custom webpack configuration define the file webpack.config.js in the root of your app.
  + Include the following to initialize the webpack config.
  + Text

    Description automatically generated
  + JS Loaders:
    - By default webpack only knows how to deal with plain JS.
    - Loaders inform webpack of the files that need to be processed before they are bundled.
    - To add a loader to a webpack config add the following
    - Text

      Description automatically generated
    - Loader definitions have 3 parts. The test property defines the type of file associated with this loader.
    - Text

      Description automatically generated
    - To transpile ES6 and forward code into ES5 standard so that it can be understood by all browsers you must include the following preset.
    - Graphical user interface, text, application

      Description automatically generated
    - To install babel:
    - 
    - To install the pre-set:
    - 
    - Currently if the code contains any await/async functionality you must install two additional dependencies and import them at the top of your index.js file.
    - 
    - Text

      Description automatically generated
  + CS Loaders:
    - The job of the [css loader](https://webpack.js.org/loaders/css-loader/) is to load the *CSS* files and the job of the [style loader](https://webpack.js.org/loaders/style-loader/) is to generate and inject a *style* element that contains all of the styles of the application.
    - Text

      Description automatically generated
    - 
  + Dev server
    - Allows for concurrent development in the browser.
    - 
    - Add the webpack start script to the package.json file.
    - Graphical user interface, text

      Description automatically generated
    - Add the following devServer config to the wbpack.config.js file.
    - Text

      Description automatically generated
    - This config will start the server in the localhost:3000 port.
  + Source map
    - Generating a source map allows for errors to be traced back to their original code in the dev file structure.
    - Webpack can generate a source map by adding the following to its config file.
    - A screenshot of a computer

      Description automatically generated with low confidence
  + Minifying
    - Optimizes the bundled code to reduce size.
    - UglifyJS is one of the leading tools for this process.
    - Webpack will automatically minimize the code when the mode is set to production in the npm script for start in package.json
  + Dev and Production config
    - For environmental variables that must change between modes (such as db ports or urls) you can define env in the config file for webpack.
    - The config function takes two parameters env and arg.
    - A screenshot of a computer

      Description automatically generated with medium confidence
    - You can use the argv variables for condition checks.
    - Additionally you can use the DefinePlugin (<https://webpack.js.org/plugins/define-plugin/>) to define global environmental constants that can be referenced in the app code.
    - Text

      Description automatically generated
    - You can inspect the bundled production version code in the app locally using the following in the build directory:
    - 
  + Polyfill
    - Fills in JS code that a browser may be missing (usually internet explorer).
    - Can be added to the code by importing polyfill libraries in the code and writing the following types of checks:
    - Text

      Description automatically generated
    - List of polyfills <https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-browser-Polyfills>
    - Browser compatibility API check: <https://developer.mozilla.org/en-US/>
  + Eject
    - You can eject an app created with create-react-app but there is no turning back.
    - Once ejected the default config file for webpack will be stored in the config directory and in a modified package.json file and will have to be maintained manually.
* JS
  + JS runtime environment based on Chrome’s V8 JS engine.
  + Files are run by the command:
    - *Node <filename>.js*
  + Comparison Operators:
    - ===
      * Strict equality
      * Requires that the content and the type be the same.
      * Should be predominantly used.
  + Variables:
    - Const-> constant variable
    - Let or var-> normal variable
    - Type can change, it is not fixed.
    - *Text

      Description automatically generated with medium confidence*
  + Arrays;
    - Defined with brackets.
    - Even if defined as const its elements can change.
    - **
    - Map method:
      * Returns a new array (deep copy).
      * Maps each element of the old array to the new array following the function reference given to the map method.
      * Notation:
        + **
      * The index of each element in the array can be passed as props to the function reference in the map function call as ‘i’ in the following manner:
        + 
        + This however is bad practice and should be avoided since the index being the key by which react identifies the object can result in undefined behaviour if the components in the list change such as because of popping or pushing from the array.
    - Concat method:
      * Returns a new list with a new element appended to the end.
      * Useful for changing state arrays since it does not modify state arrays and return a new larger array as is good practice.
      * 
    - Filter method:
      * Returns a new subset of the list based on the function reference passed which should return a Boolean for each entry in the array:
        + 
  + Objects:
    - Object literals:
      * *Text

        Description automatically generated*
      * When the name of the keys and the names of the variables are the same you can remove the keys all together:
        + 
      * Can be referenced using the dot notation or brackets:
      * *A screenshot of a computer

        Description automatically generated with medium confidence*
      * You can add properties to an object on the fly using the same notation.
      * Can also have methods declared. **Methods can use the *this* keyword, but it behaves differently when the function is assigned as a reference to another variable so it is recommended to avoid it**.
        + *Text

          Description automatically generated with medium confidence*
    - Strings:
      * Can be defined as a Java style string using plus signs:
        + ‘String1’ + variable + ‘String2’
      * Can be defined using comma separated lists when passed to a function like console.log()
        + ‘String1’, variable, ‘String2’
      * Can be defined using template string syntax (${}) notation. For this you must use backticks (`) rather than single or double quotation marks (‘ or “)
        + `String1 ${variable} String2`
      * Regular expressions:
        + Sequence of characters that forms a search pattern.
        + <https://www.w3schools.com/js/js_regexp.asp>
  + Functions:
    - Arrow functions
      * See above in React section.
      * *Text

        Description automatically generated*
      * Or (if it’s a one liner)
      * **
    - Function notation:
      * *Graphical user interface

        Description automatically generated*
  + Classes:
    - JS has classes as of the latest version ES6 but they are not used very often.
  + JSON server
    - Used as a db during development.
    - Can be installed with ‘npm install -g json-server’.
    - To start run: (the port is changed so it wont conflict with the port our default app is running on)
      * 
    - The contents of the json-server can now be accessed through the browser at address:
      * Localhost:3001/notes
      * Where notes is the name of the root of the entry we want to visualize.
  + Single-threaded & Asynchronous:
    - Diagram

      Description automatically generated
    - The JS runtime is single threaded.
    - It has a single calls stack and if some code is taking too long it stops the runtime from doing anything else until it is done.
    - If a function is synchronous and takes a long time to run it can block the browser and make it unresponsive (single-threaded).
    - Asynchronous IO operations are non-blocking and they are run through external web APIs packaged into the browser which can be thought of as separate threads. After the IO asynchronous operation has completed it pushes its callback/handler onto the callback queue.
    - The Event Loop waits until the call stack is empty and then pushes the first entry in the callback queue onto the stack to be processed.
    - In essence the IO operations which might take some time (user interactions, network requests, db requests, timeouts, ….) are run by the web apis in parallel and their callbacks are then run in single threaded fashion by the JS runtime when it has time for them.
    - An easy way to deprioritize a piece of JS code (have it run when the JS runtime is otherwise idle giving priority to everything else) it to run it as a callback to timeout of length 0:
      * setTimeout(() => console.log(‘Hello World!’), 0)
    - It is important to know which functions/callbacks are async and which are not.
  + ASYNC & AWAIT:
    - The await keyword makes the program wait until an asynchronous function call returns before continuing (function must return a promise). It can only be used inside of an asynchronous function. It is used when calling a function.
    - The async keyword makes a function run asynchronously and the program execution continues. It is used when defining a function.
    - For error handling in async functions one must use try/catch blocks.
    - An alternative is the express-async-errors package <https://github.com/davidbanham/express-async-errors>
      * 
      * It works under the hood catching all exceptions and redirecting to the error handling middleware.
      * Try/catch blocks can be removed.
    - To wait for all aync operations to return before exiting a function one can use the ‘await Promise.all(<individual promise or array>)’ which takes a promise or array.
      * Text

        Description automatically generated
  + Web workers (Browser multi-threading):
    - Can directly run JS code asynchronously.
    - Communicate with the parent JS app through event handler functions.
    - More info at:
      * <https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers>
  + Fetch:
    - Library to request data from server.
    - Supported by all browsers except IE.
    - <https://developer.mozilla.org/en-US/docs/Web/API/fetch>
  + XMLHttpRequest():
    - Object used to interact with servers:
    - Can request data from a url without doing a full refresh.
    - Used heavily in AJAX programming.
    - <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>
  + Promise object:
    - *A Promise is an object representing the eventual completion or failure of an asynchronous operation.*
    - It is an object which represents an asynchronous operation.
    - Can have 3 distinct states:
      * Pending
      * Fulfilled
      * rejected
    - Attaching an event handler to the promise to resolve once the state changes from pending to fulfilled or rejected is done using the method ‘then’:
      * Text

        Description automatically generated
    - The ‘then’ method declares a handler to be executed when the promise becomes fulfilled.
    - Promises can be chained by concatenating multiple ‘then’ method calls.
    - The catch method can also be used to handle a rejected promise. The catch method is declared at the end of the promise chain. The catch handler is called when any of the promises in the chain result in an error:
      * Text

        Description automatically generated
    - Almost always a good idea to add error and exception handling when working with promises.
  + Spread syntax (Three dot …)
    - You can use the three dot spread syntax to transform an array into individual comma separated values.
    - <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax>
    - A good application of this language feature is generating input for functions using an array.
      * 
  + AXIOS
    - External library for browser/server communication.
    - Can be added to a React project using the node package manager (npm)
    - Get() method:
      * Returns a promise.
      * Usually chains a call to the .then method without saving the promise into a variable:
        + Text

          Description automatically generated
    - Post() method:
      * Pushes data to the server at the given address.
      * Takes two arguments, first the url address of the server and second the object to push.
      * Generally it is better to omit the ID property of the object when creating it since its better to let the server generate IDs for us.
      * You can declare a handler function for it by using the then method on the object returned just like in the Get method:
        + Graphical user interface, text

          Description automatically generated
      * Newly created object on the server is stored in the data field of the resource.
      * Since you are sending a JS object the ‘Content-Type Header’ is automatically set to ‘application/json’
    - Put() method:
      * Replaces an existing resource.
      * Equivalent to HTTP Patch.
    - When the frontend and backend are running from the same origin (scheme, url, port) the URLs passed to the axios calls can be relative/partial.
      * For example if both the frontend and backend are located at ‘localhost:3001’ then the request address URL could simply be ‘/api/notes’ rather than ‘localhost:3001/api/notes’.
  + Node Package Manager (npm)
    - Currently almost all JS projects are defined using the npm.
    - Ways to initialize a project:
      * Use ‘npx create-react-app <PROJECT\_NAME>’ to initialize a basic react project.
      * Use ‘npm init’ to initialize a basic npm project with a single project.json file in the current directory.
    - Requires a package.json file located in the root directory for the project
    - Packages:
      * New packages can be defined in the package.json file manually or simply installed with npm through cmd line.
        + 
      * Should always be run in the root directory of the project
      * Installed packages will be listed under the dependencies section in the package.json file along with their version
      * Packages can be installed as development dependencies using the ‘–save-dev‘ option.
        + 
      * The src code for the npm packages can be found in the ‘node\_modules’ directory.
      * When a package is intalled its dependencies will also be installed recursively. These are called transitive dependencies. Therefore, the node-modules directory will contain many more packages than the package.json file will list as dependencies.
      * The version after the name of a package dependency in package.json is important (semantic versioning).
        + 
        + The caret (^) in front of the version number specifies that when updating the package the version will be at least larger than the current version. Additionally, the package version (tertiary) or minor version (secondary) can be larger but the major (primary) version must remain the same.
      * Packages can be updated with the ‘npm update’ command.
        + 
      * If we are installing the project on a different environment we can install all the dependencies listed in the ‘package.json’ file from scratch using the ‘npm install’ command.
        + 
    - Scripts:
      * Scripts work like aliases in bash (shortcuts for more complex commonly used commands)
      * You can add scripts to your npm project by a listing them under the ‘scripts’ node in the package.json file:
        + Text

          Description automatically generated
      * Run them by using the npm run command:
        + 
    - Production Builds:
      * Used to optimize and deploy the application to the web using static files.
      * If the app was created using ‘npx create-react-app ’ creating a production build is as simple as running ‘npm run build’.
      * This creates a directory called build which contains a directory called static holding the minified version of the JS code for the app. This file includes all of the code (including dependencies) put together in a not very readable format.
    - Proxy:
      * When we have several deployment scenarios for an app (ie. Development and production) and the address of the frontend or the backend changes in each scenario we can setup a proxy in the package.json file to redirect relative requests.
        + A screenshot of a computer

          Description automatically generated with medium confidence
        + If the React code does an HTTP request to a server address at [*http://localhost:3000*](http://localhost:3000) not managed by the React application itself (i.e. when requests are not about fetching the CSS or JavaScript of the application), the request will be redirected to the server at [*http://localhost:3001*](http://localhost:3001).
* REST API:
  + Dissertation that REST architecture is based on <https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>
  + Individual data objects are referred to as resources.
  + Every resource has a unique address (URL) associated with it.
  + A convention for creating addresses is to combine the name of the resource type with the resource’s unique address.
    - [*www.example.com/api/notes/10*](http://www.example.com/api/notes/10)
    - Would be the 10th note of the notes object at the [www.example.com/api](http://www.example.com/api) restful api.
  + Example of the operations that can be performed on a REST API (All are HTTP requests).
    - A screenshot of a computer

      Description automatically generated with medium confidence
  + To retrieve a resource at a given object we must send a HTTP GET request.
  + For example the collection of notes on the REST server at localhost:3001 could be accessed directly at address “localhost:3001/notes”
  + Furthermore, an individual note with an id of 3 could be accessed at “localhost:3001/notes/3”
  + To create a new resource at a given address we use a HTTP POST request.
  + The data to include inside a new resource would be attached to the ‘body’ of the HTTP POST request.
  + To modify an existing resource we use a HTTP PATCH request.
  + JSON-server in particular requires all data to be sent in JSON format. This means that the data must be a correctly formatted string and that the request must contain the value ‘applications/json’ in the ‘Content-Type’ field.
  + HTTP principles:
    - The HTTP standard defines that GET and HEAD requests must be safe. Meaning they must have no side-effects on the server. The state of the server must not change.
    - Additionally all request types except POST should be ‘idempotent’. Aside from error and expiration issues the side effects of N > 0 identical requests is the same as for a single request. The methods GET, HEAD, PUT and DELETE share this property. If the request does not generate side effects the response should be identical every time.
    - POST is the only request type that is neither safe nor idempotent.
* CSS:
  + Selector defines the target, declaration sets properties for the target. H1 -> selects all h1 header tags, color: green sets their color to green.
    - Text

      Description automatically generated
  + Class selectors, in JSX we can add a className attribute to each component in order to identify it for css styling:
    - Graphical user interface, text

      Description automatically generated
  + In css class selectors need to be defined with a period in front of them.
    - Text

      Description automatically generated
* Node.js
  + JS runtime based on V8.
  + ‘Npm init’ creates a blank project with a package.json file in the current directory.
  + Import Node’s built in web server module:
    - 
  + Node.js uses CommonJS modules rather than ES6 modules (import export modules) since it needed modules long before ES6 was released and JS supported them.
  + CommonJS modules are imported through the ‘require()’ function as seen above.
  + More info at : <https://nodejs.org/api/modules.html#modules-commonjs-modules>
  + We can then use the createServer method of the http module and assign it a call back function as an argument. This callBack will be called whenever a request is made to the port this is running in.
    - Text

      Description automatically generated
  + In the last two lines we bind this server to a port using its lister method.
    - A picture containing text

      Description automatically generated
  + You can set the header for a response message by using the writeHead method:
    - 
    - To return a json format object return the object with type ‘application/json’
  + You can add the payload to the HTTP ServerResponse Object (<https://www.w3schools.com/nodejs/obj_http_serverresponse.asp>) by using the end method.
    - 
    - The call to ‘JSON.stringify()’ creates a JSON type object.
  + Using Node’s built in HTTP module is cumbersome especially in larger applications. Other libraries like Express have been created to alleviate this burden.
  + You can test Node.js code in the command line (interactive scripting) by typing ‘node-repl’
    - Graphical user interface, text

      Description automatically generated
  + Environmental Variables:
    - Unlike in react to include environmental variables defined in the ‘.env’ file in your program you must install the ‘dotenv’ package.
      * 
    - Create a ‘.env’ file in the root directory next to the package.json file.
    - Add environmental variables to .env (no specific prefix or naming required)
      * 
    - Import the variables by importing the dotenv package
      * 
      * This will import them globally including for any other JS files imported afterwards in the main entry point program.
    - Use them in your program as properties of the process.env object:
      * 
    - Remember to add the .env file to ,gitignore so it wont be uploaded to public repositories.
    - If using Fly.io include a ‘.dockerignore’ file and add the ‘.env’ file to it, otherwise it will upload your environmental variables:
      * 
      * Set the environmental variables from command line:
        + 
  + Debugging:
    - VSCode
      * Setup a launch.json file with your debugger configuration
      * Add a configuration for Run npm start
      * Launch the application in debug mode by Run->Debug and choose nodeJS as the debugger.
    - Chrome Dev Tools
      * Run
        + 
      * In the browser open the dev tools and click on the green square node logo
      * Go to the Sources tab and open the file you want to debug
      * Set your breakpoints as needed.
* Express:
  + A Node.js library.
  + Importing express as a CommonJS module returns a function reference which can then be used to create servers.
    - 
  + Using the resulting object ‘app’ we can define ‘routes ’ to the application using the ‘get’ method. Each route has a relative address and a handler function.
    - Text

      Description automatically generated
    - In the handler the ‘request’ prop (argument) contains all of the information in the HTTP request received by the app.
    - In the handler, the ‘response’ prop (argument) is used to define the response to the request.
    - The ‘response.send’ method is used to respond with a string. The response header status is automatically set to ‘200’ (OK) and the ‘Content-Type’ header is automatically set to ‘text/html’ since a string was passed.
    - Graphical user interface, text

      Description automatically generated
    - To send data formatted as a JSON string we must use the ‘response.json’ method.
    - QOL improvements like automatically converting into JSON string format is why packages like Express are preferred over raw Node HTTP.
  + We can also define parameters for routes in express using the colon syntax inside the address string argument of the get method.
    - Text

      Description automatically generated
    - This syntax will handle all requests to the addresses “*/api/notes/SOMETHING*” where something could be an id or any other string.
  + Handling erroneous requests:
    - The response to an erroneous request should contain the appropriate Header status code and an empty payload.
    - Response status codes info:
      * <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>
    - It is also possible to give more information about the error by overriding the default message. <https://stackoverflow.com/questions/14154337/how-to-send-a-custom-http-status-message-in-node-express/36507614#36507614>
      * Graphical user interface, text, application, chat or text message

        Description automatically generated
    - There should also be constraints on the type of input that is accepted and an input error should correspond to a status code ‘400 bas request’.
      * Graphical user interface, text, application, chat or text message

        Description automatically generated
  + Deleting Resources (Delete Request):
    - Delete requests are handled with the ‘delete’ method of the express object:
      * Text

        Description automatically generated
    - The return status for a succesful delete request should be ‘204’
    - The return status for aa failed delete request is unspecified and can be ‘204’ or ‘404’.
  + Adding data (POST Request):
    - Need to use the JASON parser built into the Express package to convert the payload in a message into a JS object before putting it back into the body field of the request before the handler is called. Otherwise the body field would be undefined in JS.
      * Graphical user interface, text

        Description automatically generated
      * Text

        Description automatically generated
    - If the ‘Content-Type’ header field of the incoming POST request is not set to JSON or the data type the parser in use expects, it will not attempt to parse it and return an empty body.
  + Checking the headers of a request can be performed individually with the request.get method or all at once with the request.headers property.
  + Same Origin Policy and Cross-Origin Resource Sharing (CORS):
    - URLs have several components:
      * Text

        Description automatically generated
      * All three of these components together define the “origin” of the URL.
    - When an HTML includes requests for file from a different origin URL than the HTML was retrieved from can pose a security vulnerability and as such it is restricted.
      * When a browser runs into this situation it looks at the ‘Access-Control-Allow-origin’ field of the response header that returned the HTML. If it contains ‘\*’ or the origin URL it will process the external request otherwise it will not fetch it and throw an error.
    - CORS is a mechanism to enable legitimate cross-origin requests.
    - To enable requests from external origins to a Node.js database you must use the Node CORS middleware.
      * Install CORS package
        + 
      * Import the cors CommonJS package and enable the middleware for all routes.
        + Text

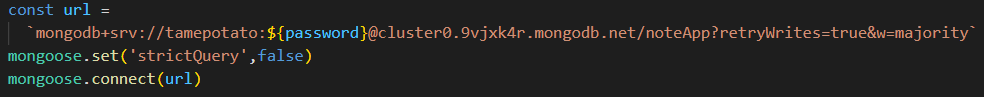
          Description automatically generated
  + Serving static content as an entry point (ie. Serving an app):
    - To serve static content you must use the built in middleware express.static() and pass it the name of the directory (in the root of the app) you want to serve.
      * 
      * Now requests to the address *www.serversaddress.com/index.html* or *www.serversaddress.com* will show the React frontend. GET requests to the address *www.serversaddress.com/api/notes* will be handled by the backend's code.
      * By deault if no path is specified the index.html file will be returned as outlined above.
    - Whenever express receives a GET request it will first check the build directory for the requested file in the address. If the file is not found it will default back to the declared routes.
* Nodemon
  + Node package
  + Automatically restarts node applications when the src files change.
  + Useful in development but not recommended for production.
  + 
  + You can start an app with nodemon by calling the binary from node\_modules and passing it the app entry point.
    - 
  + Can make a shortcut by defining as a script in the package.json file.
    - Graphical user interface, text, application

      Description automatically generated
    - Don’t need to define the path to the binary since node already knows where it is.
* Middleware:
  + An example is the JSON-parser used earlier for the POST request.
  + They are functions that can be used for handling request and response objects.
  + You can use multiple at the same time.
  + They are functions that take three parameters:
    - Text

      Description automatically generated
    - At the end of the function the next() function reference argument is called. This yields control to the next middleware.
  + Middleware is put into use by using the ‘use’ method of the express object.
    - 
    - They are called in the order that they are added to the express object using the ‘use()’ method.
    - If we want them to execute before the route functions and handlers we must declare them before them.
    - If we want them to execute after the route functions and handlers we must declare them after them. These middleware functions will only be called if none of the declared routes match the incoming request.
      * Text

        Description automatically generated
      * This middleware returns an error for requests made to nonexistent routes.
  + Error handling in middleware:
    - Sometimes its better to declare all the error handling in a single place.
    - Text

      Description automatically generated
    - The error is given to the next function as an argument. If next was called with no argument then the execution would move onto the next route or middleware.
    - If next is called with a parameter it moves on to error handling middleware.
    - Middleware error handler functions:
      * Must be the last middlewares declared after the route handlers.
      * Text

        Description automatically generated
  + Logging in middleware:
    - Morgan is a middleware that can log all incoming and outgoing HTTP messages from the backend and is easily customizable.
    - More info at: <https://expressjs.com/en/resources/middleware/morgan.html>
* Web deployment:
  + Fly.io
    - Install flyctl <https://fly.io/docs/hands-on/install-flyctl/>
    - Run ‘fly auth login’ to authenticate your account.
    - Run ‘fly launch’ from the root directory of your project and follow the prompts.
    - The previous step generates a Dockerfile and a fly.toml file in the root directory of the project. These files will define the deployment of the app.
    - Any environmental variables that need to be setup must be defined in the .toml file under [env].
    - Run ‘fly deploy’ in the directory containing the Dockerfile and .toml to deploy the app to the web.
    - Run ‘fly open’ to open the app in your browser.
    - ‘fly logs’ can be used to view server logs.
* Databases:
  + Relational:
    - SQL
    - More complex
  + Non-Relational (Document)
    - MongoDB
    - Lower complexity
    - A good library to use for interacting with the database from your Node application is Mongoose.
      * 
      * Mongoose could be described as an *object document mapper* (ODM), and saving JavaScript objects as Mongo documents is straightforward with this library.
    - Mongoose:
      * First need to connect to the database:
        + 
      * Then we need to set the SCHEMA for the data in the database to tell mongoose how the objects are to be stored and read from the database:
        + Text

          Description automatically generated
        + Validators can be customized by defining a callback function <https://mongoosejs.com/docs/api/schematype.html#schematype_SchemaType-validate>
        + Schema’s have built in methods like toJSON called configurable options. <https://mongoosejs.com/docs/api/schema.html#schema_Schema-set>
        + They can be configured using the following syntax:

Text

Description automatically generated

This can be important because by default Mongoose adds the \_id parameter to the objects and you might prefer the id parameter instead as pictured above.

* + - * + The id parameter of the mongoose object is an object itself rather than a string.
        + <https://mongoosejs.com/docs/guide.html#options>
        + <https://mongoosejs.com/docs/guide.html#toJSON>
        + <https://mongoosejs.com/docs/api.html#document_Document-toObject>
        + <https://mongoosejs.com/docs/api.html#transform>
      * In the *Note* model definition, the first "Note" parameter is the singular name of the model. The name of the collection will be the lowercase plural notes, because the [Mongoose convention](http://mongoosejs.com/docs/models.html) is to automatically name collections as the plural (e.g. notes) when the schema refers to them in the singular (e.g. Note).
        + Models can and should be defined in their own JS files inside a directory called ‘modules’.
        + You can export the reference using the following syntax:



* + - * + And can be imported by as a CommonJS module by:



* + - * Document databases like Mongo are schemaless, meaning that the database itself does not care about the structure of the data that is stored in the database. It is possible to store documents with completely different fields in the same collection.
      * The idea behind Mongoose is that the data stored in the database is given a schema at the level of the application that defines the shape of the documents stored in any given collection.
      * Models are so-called constructor function that create new JS objects based on the provided parameters. (similar to a Class and its constructor)
        + Text

          Description automatically generated
      * To save an object to the database you have to use the model instance’s (document’s) save method.
        + Text

          Description automatically generated
      * To retrieve objects from the database you can use the find method of the model.
        + Text

          Description automatically generated
        + The parameter is an object expressing the search conditions if it is an empty object then all entries in the database are returned.
        + You can also use the findById method
        + Text

          Description automatically generated
      * To delete objects the best option is thefindByIdandRemove method:
        + Text

          Description automatically generated
        + The response to a successful deletion should be 204 ‘no content’.
      * To retrieve objects from a different collection one can use the mongoose populate method:
        + Text

          Description automatically generated
        + This will populate the notes object parameter of the returned objects with the actual note objects rather than just their ID’s.
        + For this to work the type of the object parameter must be declared as a mongoose.Schema.Types.ObjectId (the user parameter in the following example):

Text

Description automatically generated

* + - * To modify a note use the findByIdandUpdate method.
        + Use a normal JS object not an instance of the Note model.
        + Text

          Description automatically generated
        + By default, the *updatedNote* parameter of the event handler receives the original document [without the modifications](https://mongoosejs.com/docs/api/model.html#model_Model-findByIdAndUpdate). We added the optional { new: true } parameter, which will cause our event handler to be called with the new modified document instead of the original.
      * Input validation in the schema.
        + Rules can be set for types in the schema.

Text

Description automatically generated

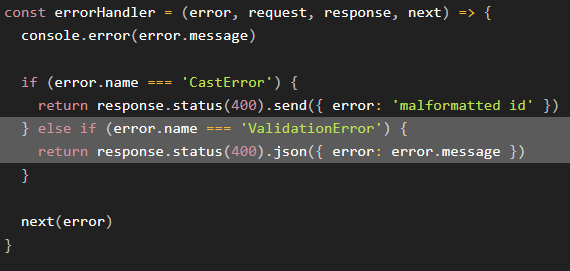
* + - * + There are built-in validators and custom ones can also be declared <https://mongoosejs.com/docs/validation.html#custom-validators>.
        + There is no built-in uniqueness validator but one can be imported using the mongoose-unique-validator package:



* + - * + If you try to store an object which breaks the constraints an exception is thrown and should be caught.

Text

Description automatically generated



By default findOneAndUpdate does not run validators. This can be fixed by changing its default settings through its last argument.

Text

Description automatically generated

* Lint / Linting / Linter:
  + Programs and the action to check code style and syntax as it is being written by highlighting.
  + The leading linter in JS is ESLint.
    - 
    - As development dependency.
  + Define default ESLint config by running the following command and answering all the questions.
    - 
    - The config will be saved in: .eslintrc.js
  + Running ESlint on a file is performed by:
    - 
  + Setting up a script for it to run it on all files is helpful.
    - Text

      Description automatically generated
    - Will check all files in the project.
  + You can define an ESLint ignore file in the root directory of the project called: .eslintignore
    - You should add the build directory to this file as we do not want it to be checked since it is the production version of the front end which should have already been checked with different eslint settings.
  + You can also enable ESLint on your editor so its runs continuously in real time.
    - The .eslintrc.js file still applies for selecting settings.
  + AirBnB is a common ESLint configuration to use.
    - <https://github.com/airbnb/javascript>
    - <https://github.com/airbnb/javascript/tree/master/packages/eslint-config-airbnb>
  + To add specific modules to the .eslintrc use the following syntax:
    - Text

      Description automatically generated
* Testing:
  + Jest is an internal testing library used by Facebook.
  + Install as a development dependency as follows:
    - 
  + Define the test script in the package.json file:
    - 
  + Define the execution environment for Jest in the package.json file:
    - Text

      Description automatically generated
  + It is common to define the current running environment (dev, test, production) as an environmental variable in the package.json script section as NODE\_ENV:
    - You can use the –runInBand option to run the tests in series rather than in parallel.
    - A screenshot of a computer

      Description automatically generated with medium confidence
    - For the NODE\_ENV env variable to work in windows you must use the cross-env package.
      * 
      * Text

        Description automatically generated
  + Write tests in the tests directory and name them as follows ‘\*.test.js’.
  + Tests are declared as follows:
    - A picture containing text

      Description automatically generated
  + Describe blocks can be used for grouping tests into logical collections. This might be necessary when using a shared setup or teardown routine for multiple tests.
    - Text

      Description automatically generated
  + The expect function can be used to verify the output:
    - toBe (same object)
    - toEqual (same value)
    - toHaveLength (arrays)
    - toBeDefined (checks object property is defined)
  + To test apps it is best to run a local database instance either in memory or in a docker container. <https://github.com/nodkz/mongodb-memory-server>
  + Super Test:
    - 
    - Example:
      * Text

        Description automatically generated
    - An object wrapped with the supertest function as shown above for the variable api becomes a superagent object.
    - For the expect method it is better to define the search string as a regex expression in order to avoid comparing the encoding and other attributes of the returned message.
    - A timeout value can be specified as a third argument to the test function:
      * Text

        Description automatically generated
  + Jest contains many functions to initialize and wrap up a testing suite <https://jestjs.io/docs/setup-teardown> .
    - beforeEach runs before each test.
      * Text

        Description automatically generated
    - afterAll runs after all tests have completed.
      * Text

        Description automatically generated
  + To run an individual test specify it as an argument by file or by name:
    - 
    - 
  + To tackle the problem caused by the latest version of Mongoose:
    - Text

      Description automatically generated
    - Add this in tests/teardown.js:
    - Graphical user interface, text

      Description automatically generated
    - Then enable the teardown globally in package.json:
    - Text

      Description automatically generated
* User Administration:
  + In relational databases each set of objects for a user would be their own table and the user owner would be included as a foreign key to the table.
  + In non-relational databases the users are also stored as a separate collection and they are linked to their objects by IDs. These Id’s are held as object properties and can be an individual value/array/list held by the user or by the note:
    - Text

      Description automatically generated note holds user
    - Text

      Description automatically generateduser holds notes
  + You could even nest the whole notes array in the user object:
    - Text

      Description automatically generated
  + Users have a unique username and a password hash which is created by a one-way hash function applied to the user’s password. DON’T STORE USER PASSWORDS IN PLAIN TEXT. Info on password storage and handling <https://codahale.com/how-to-safely-store-a-password/>.
  + Bcrypt is a password hash package:
    - 
    - Text

      Description automatically generated
    - <https://github.com/kelektiv/node.bcrypt.js/#a-note-on-rounds>
  + Token based Authentication:
    - The user’s login information is sent via an HTTP POST message to the backend where it is verified.
    - The backend returns a TOKEN which is signed digitally making it impossible to falsify with cryptographic means.
    - The TOKEN is saved on the client side possibly as a state in a react app.
    - Any further backend requests will have the user’s TOKEN attached to them.
    - The server will then use the TOKEN to identify the user’s requests and store them as linked to the user.
    - A good tool for generating tokens is the JSONWEBTOKEN package:
      * 
      * Tokens are generated and signed with an env variable:
        + 
      * There is also a library which has setup jwt as a middleware for express called express-jwt.
    - A failed authentication is responded to with a 401 UNAUTHORIZED status.
    - A successful authentication is responded to with a 200 OK status and a token.
    - There are multiple ways of attaching a token to an HTTP request from the client. One of them is to attach it to the AUTHORIZATION header of the request. The header also conveys which authentication scheme is used in case the server offers multiple authentication options.
    - We will use the Bearer authentication scheme but there are others: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication#Authentication_schemes>
    - Token expiration period can be set by passing a third object to JSONWEBTOKEN:
      * Text

        Description automatically generated
      * Expiration periods must seek a balance between security and user inconvenience.
    - Server Side Session:
      * Another option for authentication is a server-side session where the information about each token is saved to a backend database and each time a request is made with that token the backend checks that the token is still valid in the db.
      * This can be slow as memory access is potentially very slow.
      * To speed lookups a special key-value database like Redis can be used. <https://redis.io/>
      * This system allows the token rights to be revoked at any time for any reason.
      * The downside aside from speed is backend complexity.
    - ALL LOGIN AND PASSWORD INFORMATION SHOULD BE TRANMITTED OVER HTTPS EXCLUSIVELY.