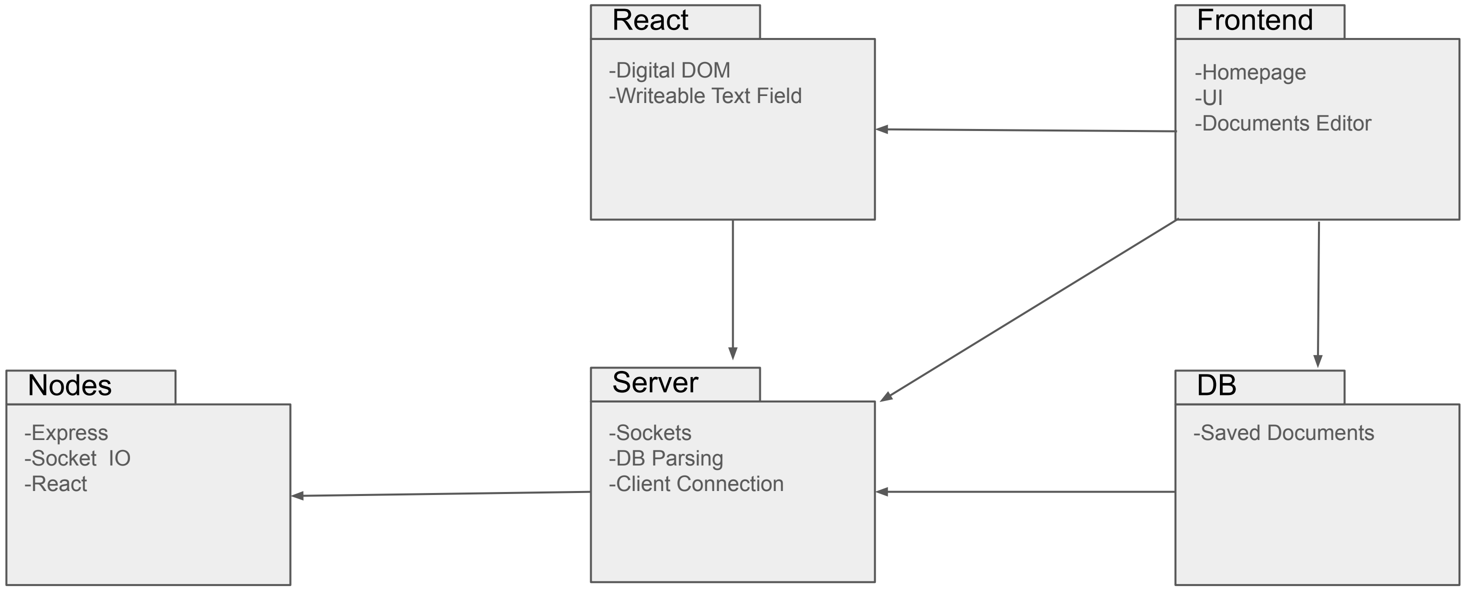
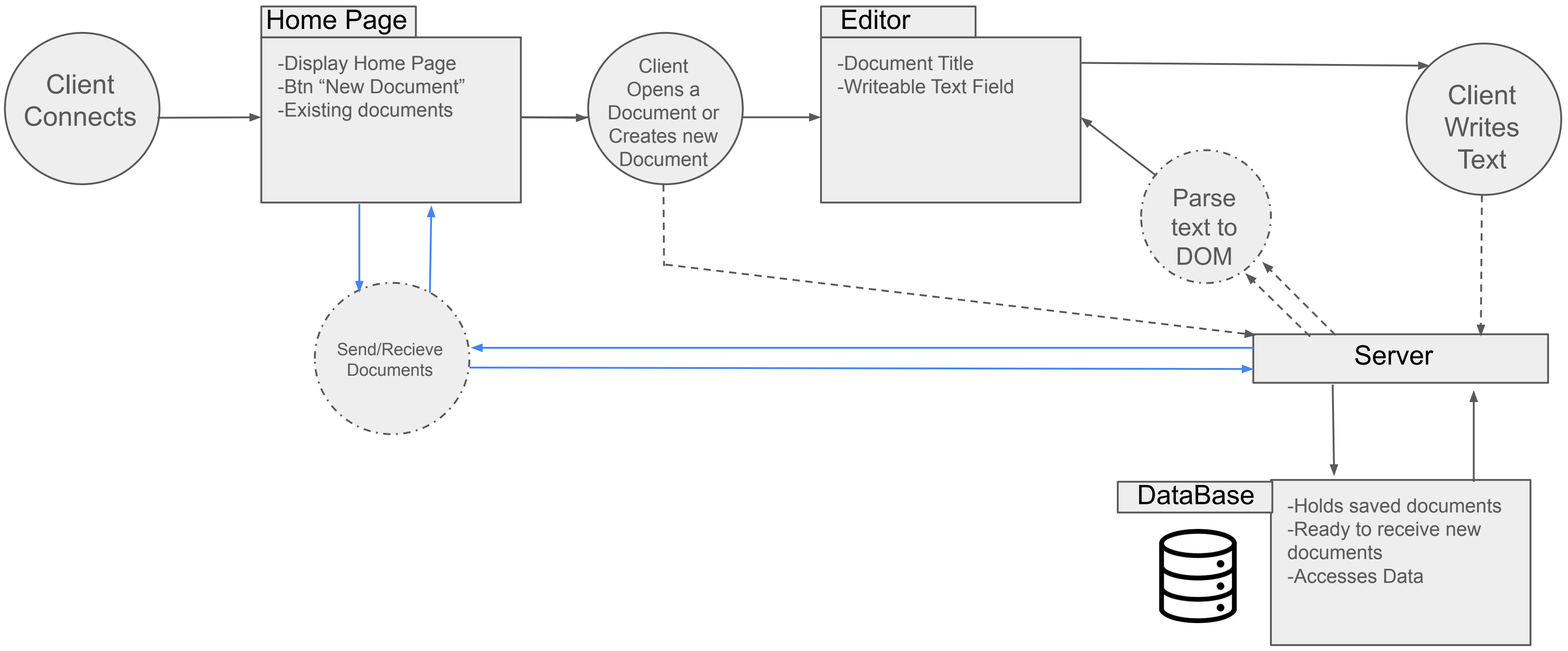
Whiteboard

Af Simon Ebbesen Asmussen og Mathias Sørensen

UML - Package Diagram



UML - State Diagram



HTML, CSS, JS, Typescript

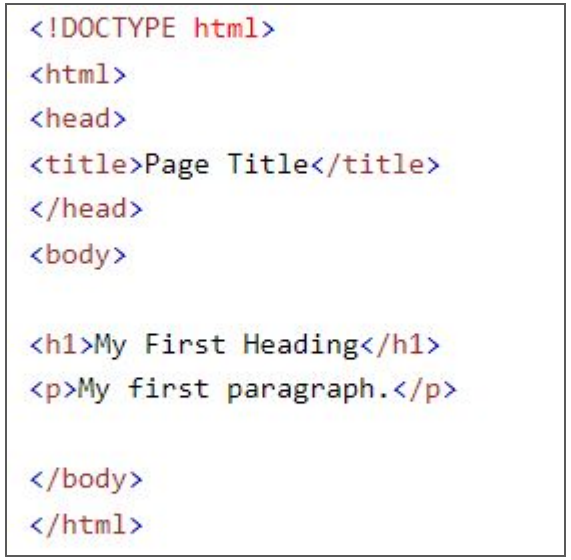
- En Web-App består af en kombination af HTML, CSS, og JavaScript (typiskt)

- Mange Web-Apps er statiske, men de kan også være Real-Time

- Real-Time defineres som: “Kommunikation som foregår inden for en tidsramme, som føles øjeblikkelig eller næsten øjeblikkelig”

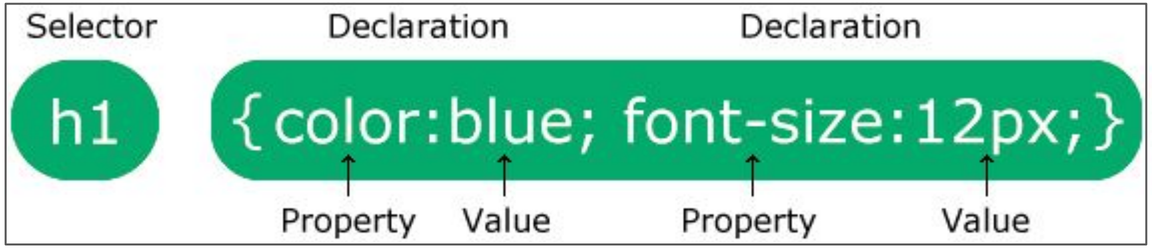
- Typescript er en udvidelse af JS som gør det muligt at programmere objektorienteret. Typescript gør det muligt at lave types, classes, mm.

HTML, CSS, JS, Typescript 2

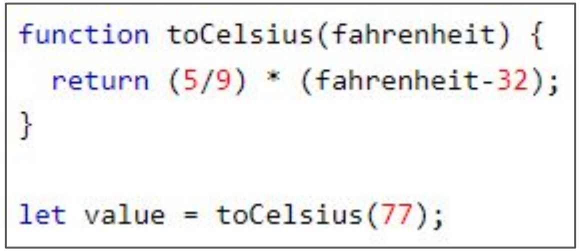


```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

HTML Syntax



CSS Syntax

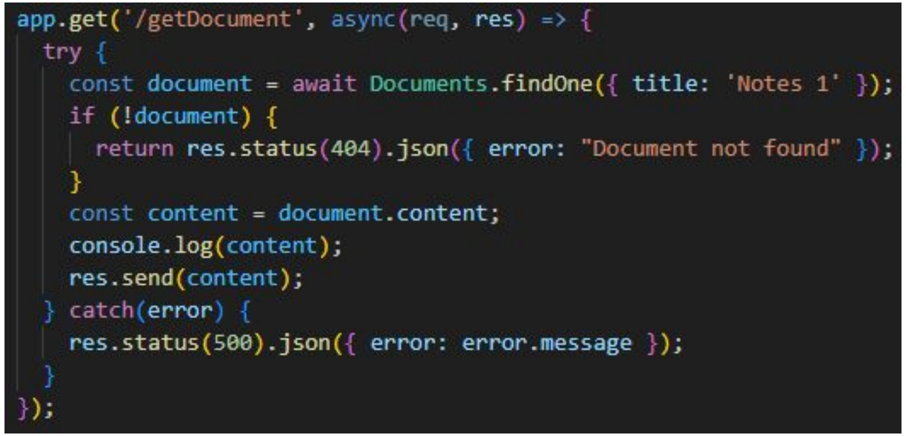


```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}

let value = toCelsius(77);
```

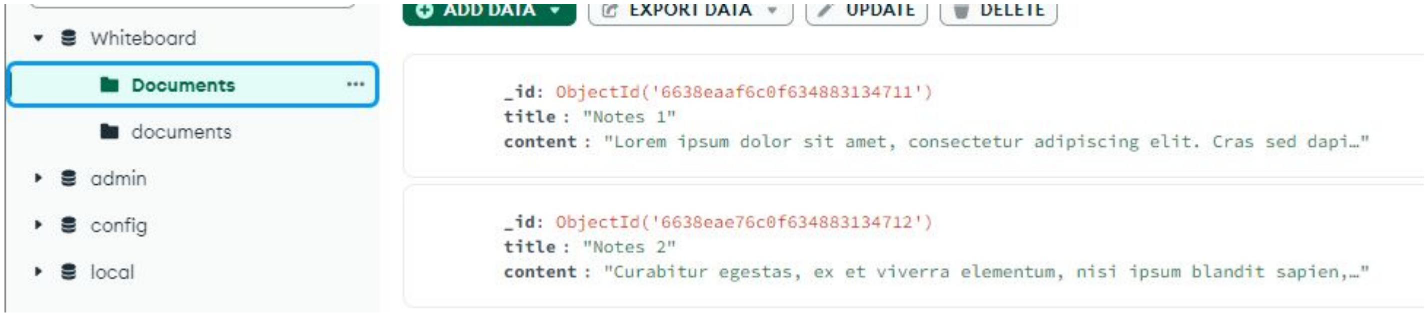
JS Syntax

MongoDB (database)

- NoSQL
- Whiteboard (Database)
    - Documents (Collection)
        - BSON (Document)
            - _ID: ObjectId
            - Title: String
            - Content: String



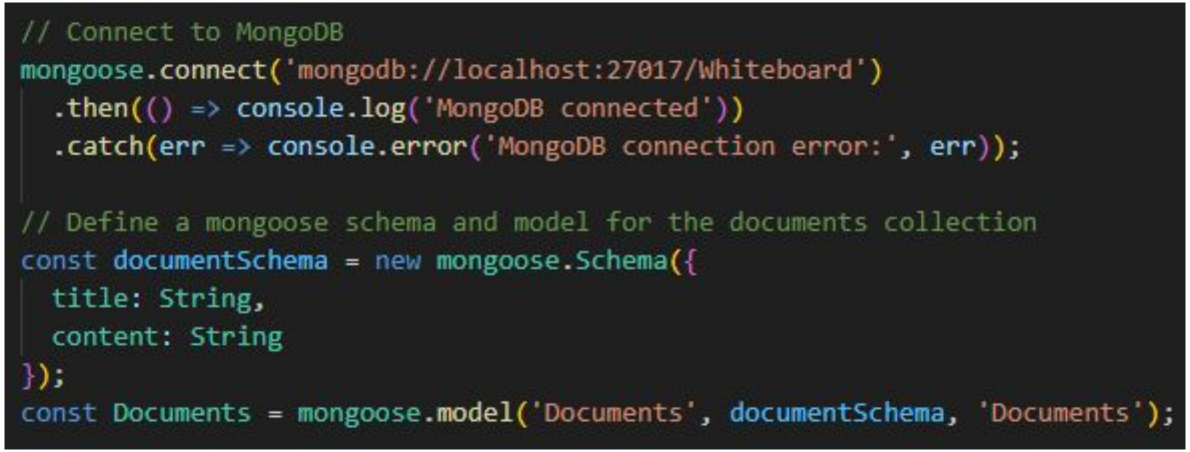
```
app.get('/getDocument', async(req, res) => {
  try {
    const document = await Documents.findOne({ title: 'Notes 1' });
    if (!document) {
      return res.status(404).json({ error: "Document not found" });
    }
    const content = document.content;
    console.log(content);
    res.send(content);
  } catch(error) {
    res.status(500).json({ error: error.message });
  }
});
```



```
_id: ObjectId('6638eaaf6c0f634883134711')
title : "Notes 1"
content : "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras sed dapi…"

_id: ObjectId('6638eae76c0f634883134712')
title : "Notes 2"
content : "Curabitur egestas, ex et viverra elementum, nisi ipsum blandit sapien,…"
```

MongoDB (Mongoose)

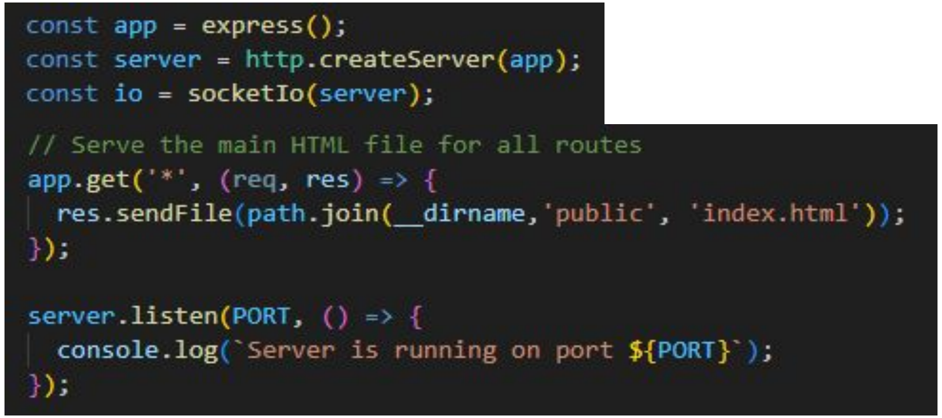
- Connecting to DB
- Mongoose Schema
- Mongoose Model
  - Save
  - Find
  - Update
  - Delete



```
// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/Whiteboard')
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.error('MongoDB connection error:', err));

// Define a mongoose schema and model for the documents collection
const documentSchema = new mongoose.Schema({
  title: String,
  content: String
});
const Documents = mongoose.model('Documents', documentSchema, 'Documents');
```

Node (Server)

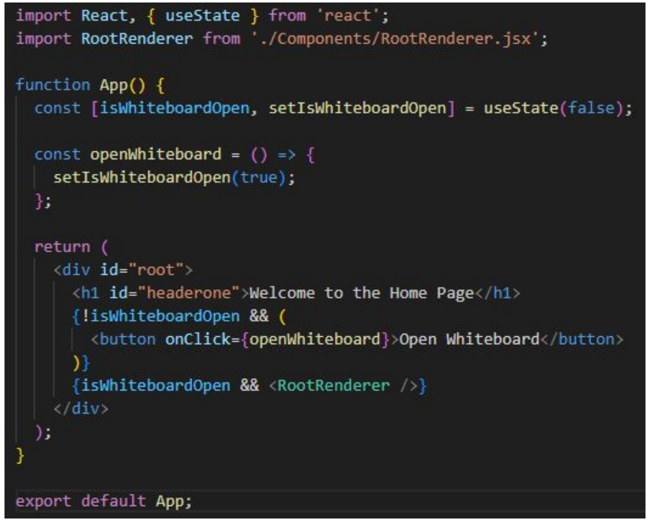
- Node er et runtime-environment som gør det muligt at teste og køre webapps uden for client browseren.
- Node er open source, og har et kæmpe library af tilføjelses programmer og andre packages.
- Node gør det muligt at køre en server, og igennem node kan man håndtere HTTP Requests og sockets.



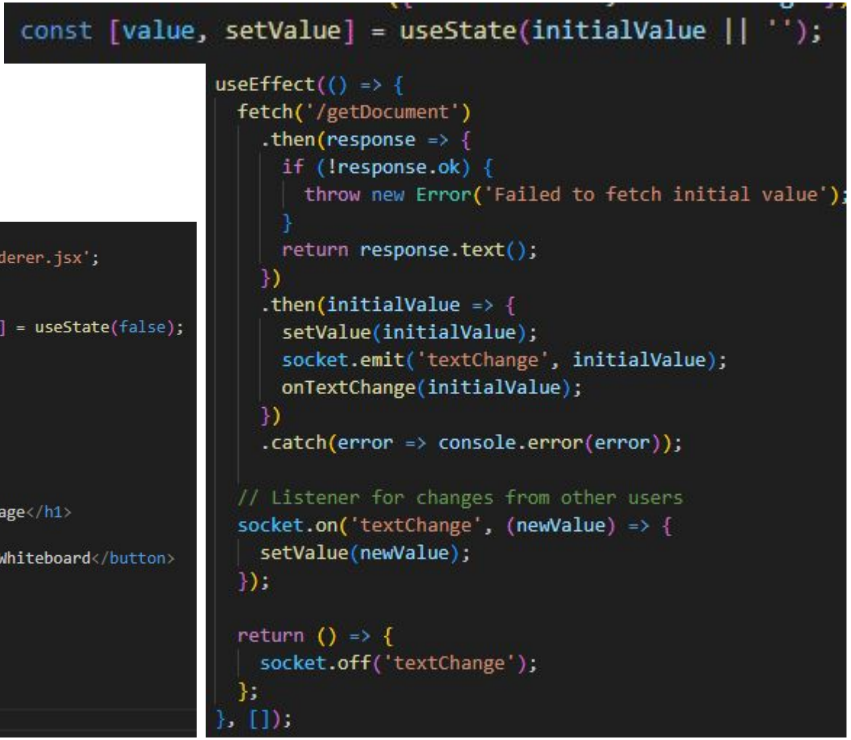
```
const app = express();
const server = http.createServer(app);
const io = socketIo(server);

// Serve the main HTML file for all routes
app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname,'public', 'index.html'));
});

server.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

React (Front end)

- Components
  - JSX
- Hooks
  - UseState
  - UseEffect
- Build
  - JSX → JS



```
import React, { useState } from 'react';
import RootRenderer from './Components/RootRenderer.jsx';

function App() {
  const [isWhiteboardOpen, setIsWhiteboardOpen] = useState(false);

  const openWhiteboard = () => {
    setIsWhiteboardOpen(true);
  };

  return (
    <div id="root">
      <h1 id="headerone">Welcome to the Home Page</h1>
      {!isWhiteboardOpen && (
        <button onClick={openWhiteboard}>Open Whiteboard</button>
      )}
      {isWhiteboardOpen && <RootRenderer />}
    </div>
  );
}

export default App;
```

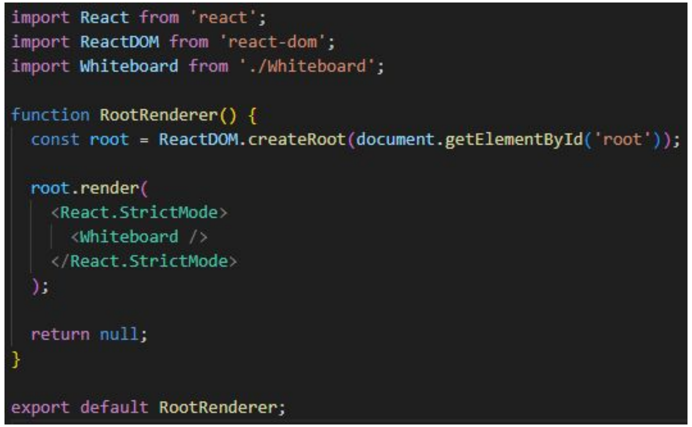


```
const [value, setValue] = useState(initialValue || '');
```

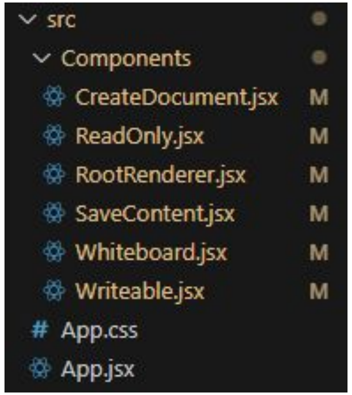
```
useEffect(() => {
  fetch('/getDocument')
    .then(response => {
      if (!response.ok) {
        throw new Error('Failed to fetch initial value');
      }
      return response.text();
    })
    .then(initialValue => {
      setValue(initialValue);
      socket.emit('textChange', initialValue);
      onTextChange(initialValue);
    })
    .catch(error => console.error(error));

  // Listener for changes from other users
  socket.on('textChange', (newValue) => {
    setValue(newValue);
  });

  return () => {
    socket.off('textChange');
  };
}, []);
```

React

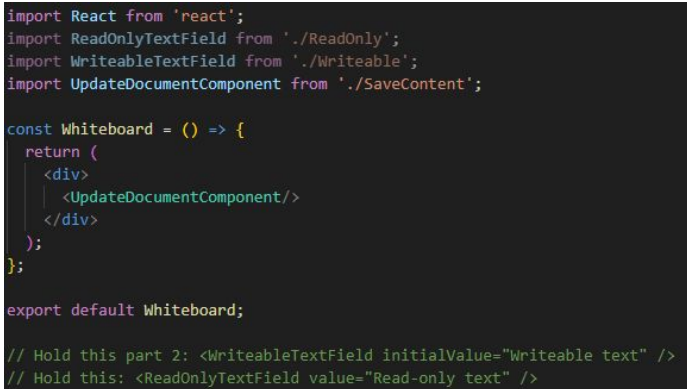
- Hooks
  - useReducer
  - useMemo
  - useRef
  - useCallback



```jsx
import React from 'react';
import ReactDOM from 'react-dom';
import Whiteboard from './Whiteboard';

function RootRenderer() {
  const root = ReactDOM.createRoot(document.getElementById('root'));

  root.render(
    <React.StrictMode>
      <Whiteboard />
    </React.StrictMode>
  );

  return null;
}

export default RootRenderer;
```



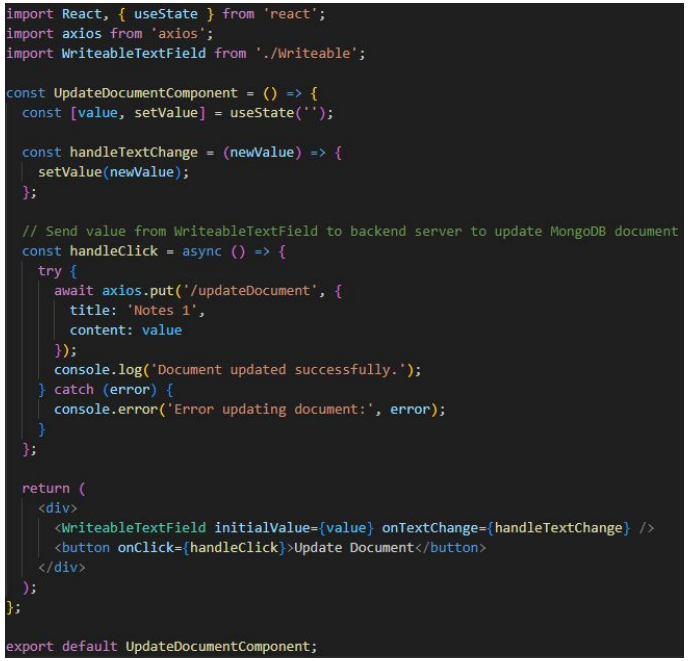
```
src
  Components
    CreateDocument.jsx   M
    ReadOnly.jsx         M
    RootRenderer.jsx     M
    SaveContent.jsx      M
    Whiteboard.jsx       M
    Writeable.jsx        M
  # App.css
  App.jsx
```



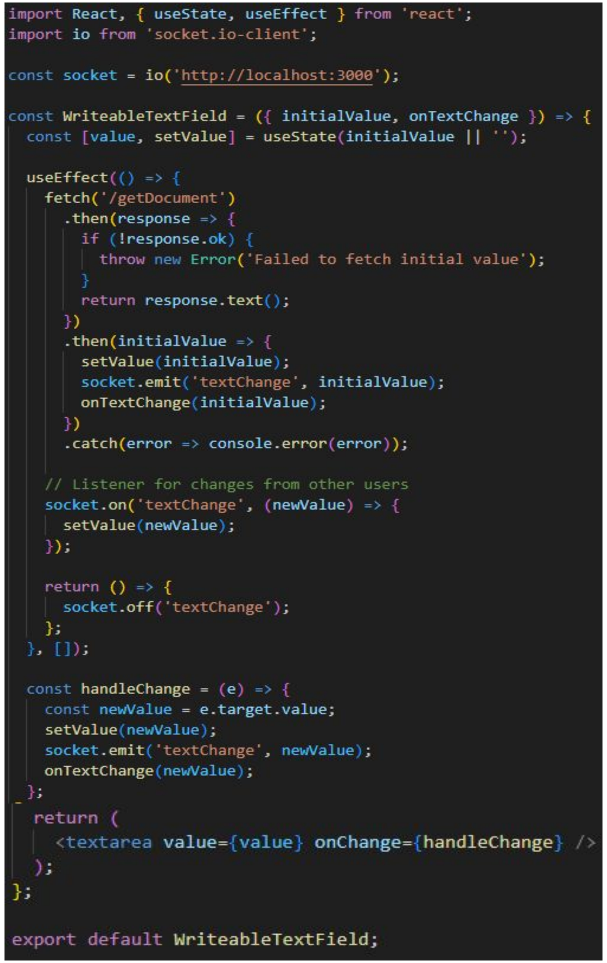
```jsx
import React from 'react';
import ReadOnlyTextField from './ReadOnly';
import WriteableTextField from './Writeable';
import UpdateDocumentComponent from './SaveContent';

const Whiteboard = () => {
  return (
    <div>
      <UpdateDocumentComponent/>
    </div>
  );
};

export default Whiteboard;

// Hold this part 2: <WriteableTextField initialValue="Writeable text" />
// Hold this: <ReadOnlyTextField value="Read-only text" />
```

React components

- Primære components



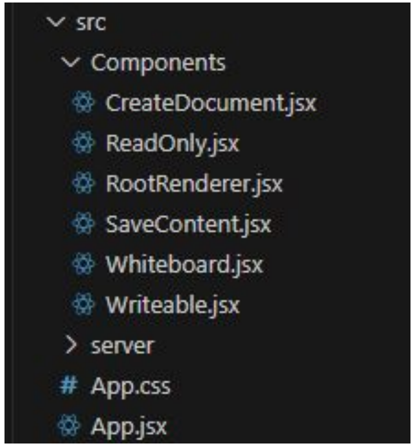
```jsx
import React, { useState } from 'react';
import axios from 'axios';
import WriteableTextField from './Writeable';

const UpdateDocumentComponent = () => {
  const [value, setValue] = useState('');

  const handleTextChange = (newValue) => {
    setValue(newValue);
  };

  // Send value from WriteableTextField to backend server to update MongoDB document
  const handleClick = async () => {
    try {
      await axios.put('/updateDocument', {
        title: 'Notes 1',
        content: value
      });
      console.log('Document updated successfully.');
    } catch (error) {
      console.error('Error updating document:', error);
    }
  };

  return (
    <div>
      <WriteableTextField initialValue={value} onTextChange={handleTextChange} />
      <button onClick={handleClick}>Update Document</button>
    </div>
  );
};

export default UpdateDocumentComponent;
```



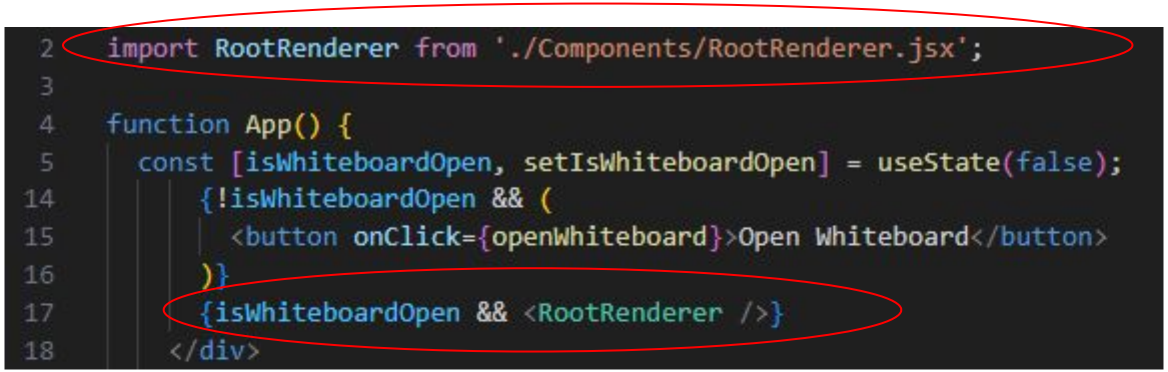
```jsx
import React, { useState, useEffect } from 'react';
import io from 'socket.io-client';

const socket = io('http://localhost:3000');

const WriteableTextField = ({ initialValue, onTextChange }) => {
  const [value, setValue] = useState(initialValue || '');

  useEffect(() => {
    fetch('/getDocument')
      .then(response => {
        if (!response.ok) {
          throw new Error('Failed to fetch initial value');
        }
        return response.text();
      })
      .then(initialValue => {
        setValue(initialValue);
        socket.emit('textChange', initialValue);
        onTextChange(initialValue);
      })
      .catch(error => console.error(error));

    // Listener for changes from other users
    socket.on('textChange', (newValue) => {
      setValue(newValue);
    });

    return () => {
      socket.off('textChange');
    };
  }, []);

  const handleChange = (e) => {
    const newValue = e.target.value;
    setValue(newValue);
    socket.emit('textChange', newValue);
    onTextChange(newValue);
  };

  return (
    <textarea value={value} onChange={handleChange} />
  );
};

export default WriteableTextField;
```

Architecture

- React, som komponentbaseret framework, lægger naturligt op til Component-based architecture.
- Vi har forsøgt at holde os til MVC, men er endt med en arkitektur som minder mere om Component-based.
- At vi kører en component-based arkitektur lader os nemmere skalere i fremtiden. Scalability er et hovedfokus i vores programmerings-process.



```
src
  Components
    CreateDocument.jsx
    ReadOnly.jsx
    RootRenderer.jsx
    SaveContent.jsx
    Whiteboard.jsx
    Writeable.jsx
  server
  # App.css
  App.jsx
```



```
import RootRenderer from './Components/RootRenderer.jsx';

function App() {
  const [isWhiteboardOpen, setIsWhiteboardOpen] = useState(false);
      {!isWhiteboardOpen && (
        <button onClick={openWhiteboard}>Open Whiteboard</button>
      )}
      {isWhiteboardOpen && <RootRenderer />}
    </div>
```

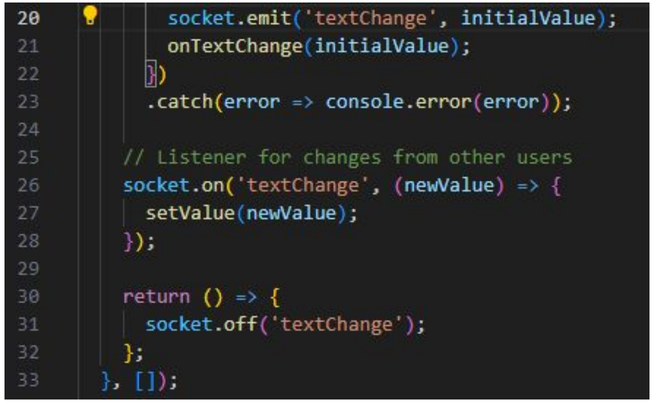
Integration Techniques

Vi har fiflet med både sockets og HTTP requests, dog er der lige nu hovedsageligt ingen HTTP Requests udover til at opstarte server og sockets.

- Vi har sockets til at styre kommunikation på siden.
- Sockets håndterer run time kommunikation, men oprettes af en initial request fra client til server.
- Why not polling?
- Da vi ikke **skal** deploy vores applikation fandt vi det nemmere og mindre ressourcekrævende at oprette sockets og event listeners
- Subscribe/Publish => appen bruger et Subscribe/Publish system til at håndtere text change.

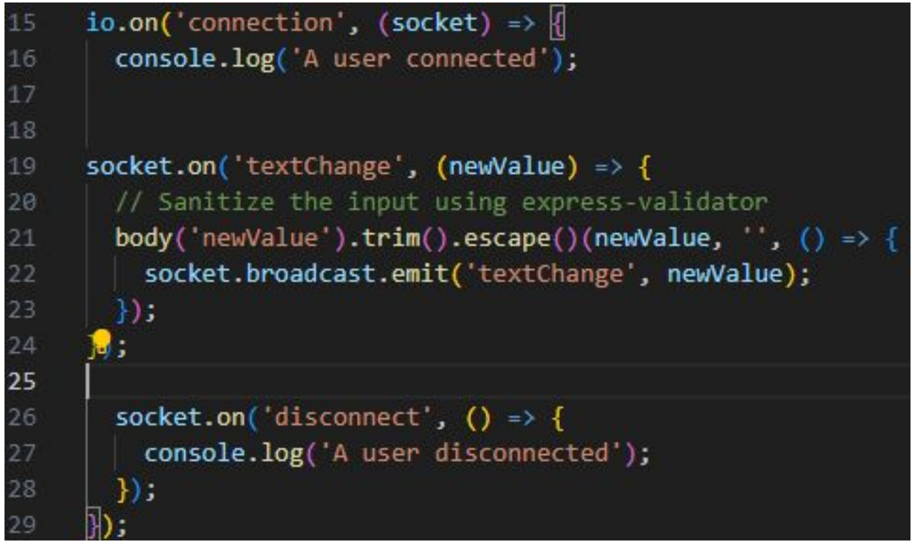
Sockets

- Web sockets fungerer som en to-vejs kommunikations facilitator.
- Web sockets holder kommunikationene mellem client og server åben.
- Der benyttes sockets
  til real-time opdatering
  af tekstfeltet i appen.



```
        socket.emit('textChange', initialValue);
        onTextChange(initialValue);
      })
      .catch(error => console.error(error));

    // Listener for changes from other users
    socket.on('textChange', (newValue) => {
      setValue(newValue);
    });

    return () => {
      socket.off('textChange');
    };
  }, []);
```

React/Client



```
io.on('connection', (socket) => {
  console.log('A user connected');


socket.on('textChange', (newValue) => {
  // Sanitize the input using express-validator
  body('newValue').trim().escape()(newValue, '', () => {
    socket.broadcast.emit('textChange', newValue);
  });
});

  socket.on('disconnect', () => {
    console.log('A user disconnected');
  });
});
```

JS/Server

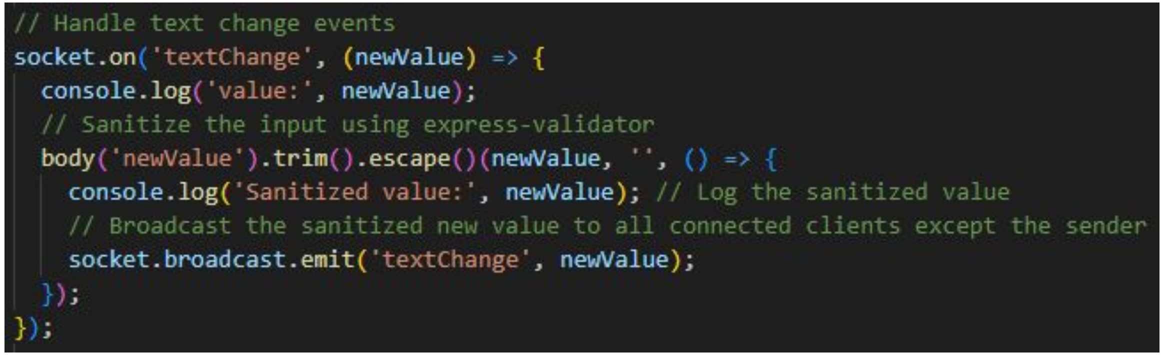
Accessibility

Vi har ikke implementeret accesibility i applikationen, men har roadmappet hvad der skulle implementeres:

- Tab Focus = keyDown event som tilføjer fokus på et DOM element.
- TabIndex = Liste som definerer hvilken rækkefølge DOM elementer bliver fokuseret i.
- ARIA attributter= attributter som kan oplæses af skærmlæsere (Accessible Rich Internet Applications)
- Font Size change = Mulighed for at opskalere font størrelse (tre knapper lign. lille -> mellem -> stor)

Security

- Using MongoDB which is a NoSQL database automatically protects against SQL injection
- Sanitize the input fields value before showing it to other clients
  - Using express-validator



```
// Handle text change events
socket.on('textChange', (newValue) => {
  console.log('value:', newValue);
  // Sanitize the input using express-validator
  body('newValue').trim().escape()(newValue, '', () => {
    console.log('Sanitized value:', newValue); // Log the sanitized value
    // Broadcast the sanitized new value to all connected clients except the sender
    socket.broadcast.emit('textChange', newValue);
  });
});
```

Testing

Forskellige Tests

- Acceptance test ×
- Usability test ✔
  - Accessibility, functionality, stability
- Unit test ×
- End to end test ×
- Security test ✔
  - XSS, SQL injection
- Performance test ×