Hochschule Bochum

Sommersemester 2022

Dozenten: Prof. Dr.-Ing. Carsten Köhn

Zweitprüfer: Nils Backenköhler

Entwicklung und Evaluierung einer Webapp zum Testen eines Transfermoduls für medizinische Daten

Vorgelegt als Bachelorarbeit

Abgabetermin 29.08.2022

Daniel Tura

10. Semester Informatik

Elisabethstr. 24, 44866 Bochum

daniel.tura@live.de

0177 / 7179764

Matrikelnummer: 017200664

Hochschule Bochum
Bochum University
of Applied Sciences

Inhalt

Abbildungsverzeichnis

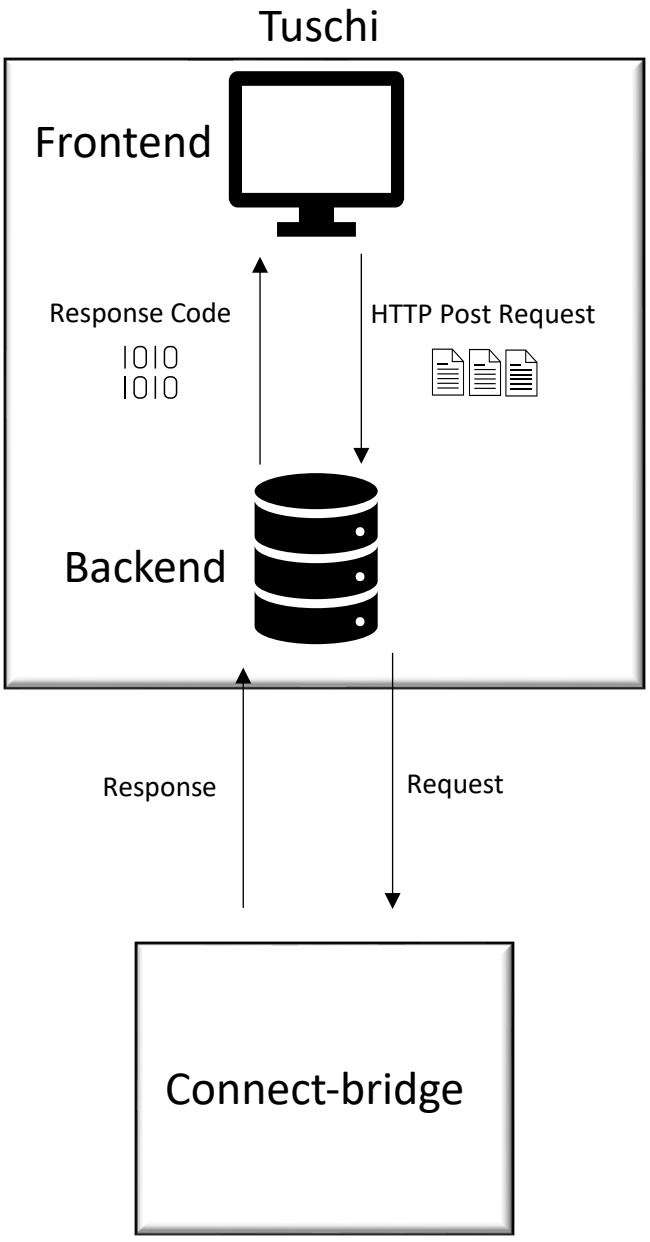

Abbildung 1 Aufbau der Tuschi (Quelle: Eigene Abbildung)

II

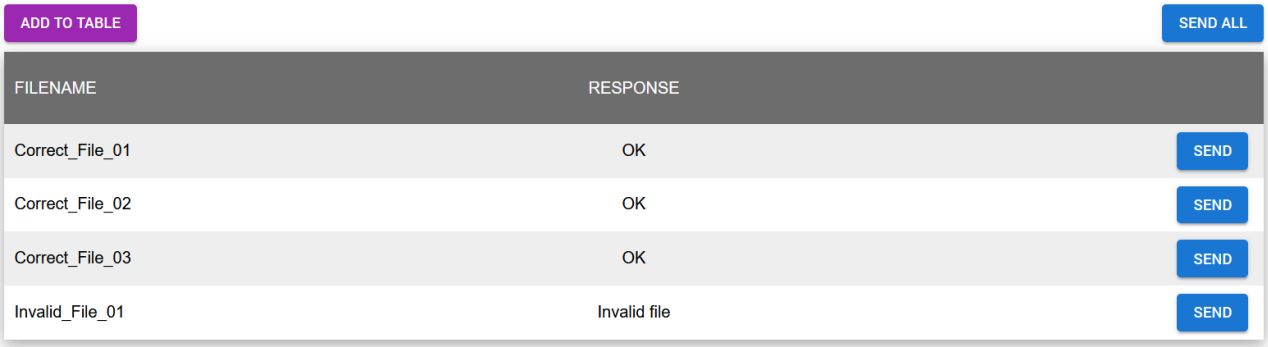

Abbildung 2 Abbildung der Tuschi Frontend UI (Quelle: Eigene Abbildung)

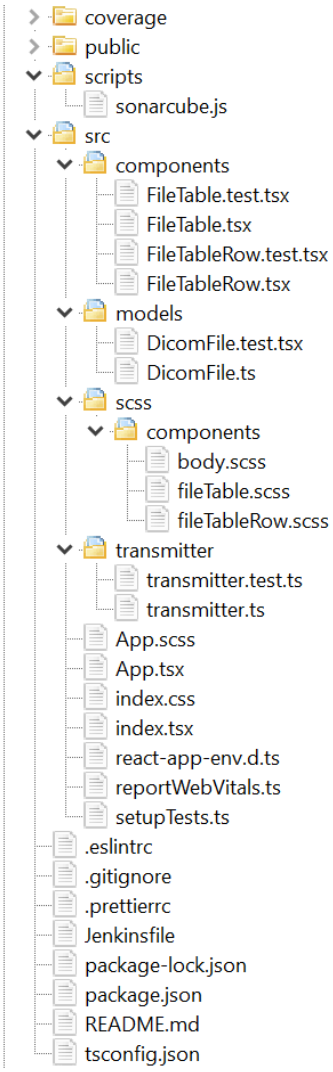

Abbildung 4 Frontend Ordnerstruktur (Quelle: Eigene Abbildung)

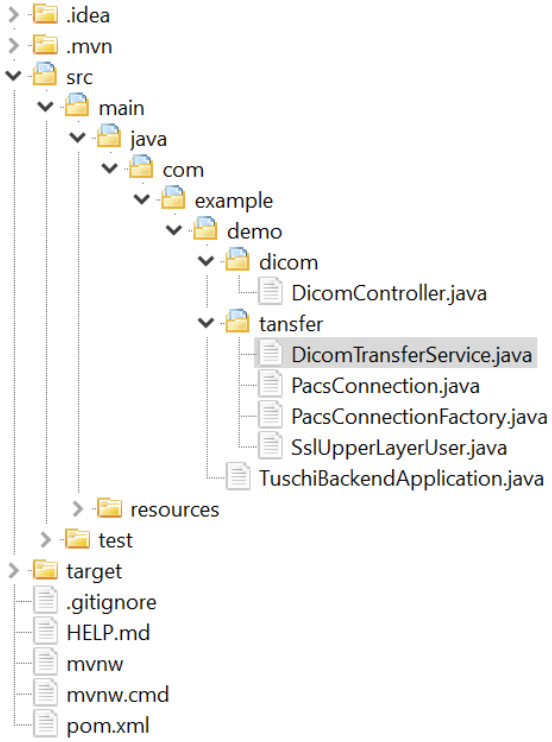

Abbildung 3 Backend Ordnerstruktur (Quelle: Eigene Abbildung)

Listingverzeichnis

```
1.  import axios from 'axios';
2.  import DicomFile from '../models/DicomFile';
3.
4.  const backendConfig = {
5.    dicomHost: 'localhost',
6.    dicomPort: '104',
7.    calledAeTitle: 'connect-box',
8.    maxPduSize: '32768',
9.  };
10.
11. export const transmit = async (dicomFiles: DicomFile[]) => {
12.   const url = 'http://localhost:8080/transmit';
13.   const formData = new FormData();
14.   dicomFiles.forEach(dicomFile => {
15.     formData.append('files[]', dicomFile.file);
16.   });
17.
18.   const config = {
19.     headers: {
20.       'content-type': 'multipart/form-data',
21.       dicomHost: backendConfig.dicomHost,
22.       dicomPort: backendConfig.dicomPort,
23.       calledAeTitle: backendConfig.calledAeTitle,
24.       maxPduSize: backendConfig.maxPduSize,
25.     },
26.   };
27.
28.   return axios
29.     .post(url, formData, config)
30.     .then(
31.       response =>
32.         // response.data is of Type Array<Object>
33.         response.data
34.     )
35.     .catch(() => 'No Connection');
36. };
```

Listing 1 transmitter.ts (eigener Code)

```
1.  class DicomFile {
2.    _file: File;
3.
4.    _transmissionResponse: string;
5.
6.    constructor(file: File) {
7.      this._file = file;
8.      this._transmissionResponse = '';
9.    }
10.
11.   get name() {
12.     return this.file.name;
13.   }
14.
15.   get file() {
16.     return this._file;
17.   }
18.
19.   get transmissionResponse() {
20.     return this._transmissionResponse;
21.   }
22.
23.   set transmissionResponse(res: string) {
```

```
24.     this._transmissionResponse = res;
25.   }
26.
27.   public static arrayFrom(fileList: File[]): DicomFile[] {
28.     return fileList.map(fileElement => new DicomFile(fileElement));
29.   }
30. }
31.
32. export default DicomFile;
```

Listing 2 DicomFile.ts (eigener Code)

```
1.  import React, { useState } from 'react';
2.  import { Button } from '@mui/material';
3.  import DicomFile from '../models/DicomFile';
4.  import { transmit } from '../transmitter/transmitter';
5.
6.  const FileTableRow: React.FC<{
7.    dicomFile: DicomFile;
8.    tableIsTransmitting: boolean;
9.  }> = ({ dicomFile, tableIsTransmitting }) => {
10.   const [rowIsTransmitting, setRowIsTransmitting] = useState<boolean>(false);
11.
12.   type Response = {
13.     name: string;
14.     transmissionResponse: string;
15.   };
16.
17.   const setTransmissionResponse = (df: DicomFile, r: Response) => {
18.     if (r.transmissionResponse === '0') {
19.       df.transmissionResponse = 'OK';
20.     } else {
21.       df.transmissionResponse = r.transmissionResponse;
22.     }
23.   };
24.
25.   const handleSetResponse = () => {
26.     setRowIsTransmitting(true);
27.     transmit([dicomFile]).then(data => {
28.       setTransmissionResponse(dicomFile, data[0]);
29.       setRowIsTransmitting(false);
30.     });
31.   };
32.
33.   return (
34.     <div
35.       className="table__content-table-row"
36.       key={dicomFile.name}
37.       role="listitem"
38.     >
39.       <div className="table__row-table-data table-data--filename">{dicomFile.name}</div>
40.       <div className="table__row-table-data">{dicomFile.transmissionResponse}</div>
41.       <div className="table__row-table-data table-data--actions">
42.         <Button
43.           variant="contained"
44.           disabled={tableIsTransmitting || rowIsTransmitting}
45.           onClick={handleSetResponse}
46.         >
47.           SEND
48.         </Button>
49.       </div>
50.     </div>
51.   );
52. };
53.
```

```
54. export default FileTableRow;
```

Listing 3 FileTableRow.tsx (eigener Code)

```
1.  import React, { useState } from 'react';
2.  import { Button } from '@mui/material';
3.  import DicomFile from '../models/DicomFile';
4.  import { transmit } from '../transmitter/transmitter';
5.  import FileTableRow from './FileTableRow';
6.
7.  const FileTable: React.FC = () => {
8.    const [fileStore, setFileStore] = useState<DicomFile[]>([]);
9.    const [tableIsTransmitting, setTableIsTransmitting] = useState<boolean>(false);
10.
11.   type Response = {
12.     name: string;
13.     transmissionResponse: string;
14.   };
15.
16.   const containsFile = (file: DicomFile, list: DicomFile[]) =>
17.     list.some(elem => elem.name === file.name);
18.
19.   const mergeFiles = (currentFiles: DicomFile[], uploadedFiles: DicomFile[]): DicomFile[]
    => {
20.     const newFiles = uploadedFiles.filter(elem => !containsFile(elem, currentFiles));
21.
22.     return [...currentFiles, ...newFiles];
23.   };
24.
25.   const handleAddFiles = (uploadedFiles: DicomFile[]) => {
26.     setFileStore(mergeFiles(fileStore, uploadedFiles));
27.   };
28.
29.   const setTransmissionResponse = (df: DicomFile, r: Response) => {
30.     if (r.transmissionResponse === '0') {
31.       df.transmissionResponse = 'OK';
32.     } else {
33.       df.transmissionResponse = r.transmissionResponse;
34.     }
35.   };
36.
37.   const handleSetResponse = () => {
38.     setTableIsTransmitting(true);
39.     transmit(fileStore).then(data => {
40.       data.forEach((response: Response) => {
41.         fileStore.forEach(dicomFile => {
42.           if (dicomFile.name === response.name) {
43.             setTransmissionResponse(dicomFile, response);
44.           }
45.         });
46.       });
47.       setTableIsTransmitting(false);
48.     });
49.   };
50.
51.   return (
52.     <>
53.       <div className="actions">
54.         <div className="actions__element">
55.           <Button
56.             variant="contained"
57.             component="label"
58.             color="secondary"
59.           >
60.             Add files
```

```
            <input
              id="file-upload"
              type="file"
              multiple
              onChange={e => {
                if (e.target.files) {
                  const files = Array.from(e.target.files);
                  handleAddFiles(DicomFile.arrayFrom(files));
                }
              }}
            />
          </Button>
        </div>
        <div className="actions__element--send-all">
          <Button
            variant="contained"
            component="label"
            disabled={tableIsTransmitting}
            onClick={handleSetResponse}
          >
            Send All
          </Button>
        </div>
      </div>
      <div
        className="container"
        data-testid="table-container"
      >
        <div className="table">
          <div className="table__header">
            <div className="table__header-item table__header-item--filename">
              <div className="header-text header-text-filename">Filename</div>
            </div>
            <div className="table__header-item">
              <div className="header-text">Response</div>
            </div>
            <div className="table__header-item">
              <div className="header-text" />
            </div>
                </div>
                <div className="table__content">
                  {fileStore.map(file => (
                    <FileTableRow
                      key={file.name}
                      dicomFile={file}
                      tableIsTransmitting={tableIsTransmitting}
                    />
                  ))}
                </div>
              </div>
            </div>
          </>
        );
      };

      export default FileTable;
```

Listing 4 FileTable.tsx (eigener Code)

```
1.  import React from 'react';
2.  import { render, screen } from '@testing-library/react';
3.  import userEvent from '@testing-library/user-event';
4.  import FileTable from './FileTable';
5.
6.  describe('FileTable', () => {
```

```
    const mockFile1 = new File([''], 'dateiname1', { type: 'dicom' });
    const mockFile2 = new File([''], 'dateiname2', { type: 'dicom' });
    const mockFile3 = new File([''], 'dateiname3', { type: 'dicom' });

    it('rendersWithoutCrashing', () => {
      render(<FileTable />);
    });

    it('test_FileTable_whenFileAdded_expectItToBeInTheDocument', () => {
      render(<FileTable />);
      const fileInput = screen.getByLabelText('Add files');
      userEvent.upload(fileInput, mockFile1);
      const tableRow = screen.getByText('dateiname1');

      expect(tableRow).toBeInTheDocument();
    });

    it('test_FileTable_whenFilesAdded_expectNoDoubleEntries', () => {
      render(<FileTable />);
      const fileInput = screen.getByLabelText('Add files');
      userEvent.upload(fileInput, [mockFile1, mockFile2, mockFile3]);
      userEvent.upload(fileInput, mockFile1);
      const tableRows = screen.queryAllByText('dateiname1');

      expect(tableRows.length).toBe(1);
    });

    it('test_FileTable_whenFilesAdded_expectSameAmountOfEntries', () => {
      render(<FileTable />);
      const fileInput = screen.getByLabelText('Add files');
      userEvent.upload(fileInput, [mockFile1, mockFile2, mockFile3]);
      const tableRows = screen.getAllByRole('listitem');

      expect(tableRows).toHaveLength(3);
    });
});
```

Listing 5 FileTable.test.tsx (eigener Code)

```
package com.example.demo.dicom;

import com.example.demo.tansfer.DicomTransferService;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ArrayNode;
import com.fasterxml.jackson.databind.node.ObjectNode;
import com.visustt.dicomTk.CommonDicomException;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.util.List;

@RestController
public class DicomController {

    private static final Logger LOG = LogManager.getLogger(DicomController.class);
    private final DicomTransferService dicomTransferService;
    private final ObjectMapper mapper;

    @Autowired
```

```
27.     public DicomController(
28.         DicomTransferService dicomTransferService,
29.         ObjectMapper jacksonObjectMapper
30.     ) {
31.         this.dicomTransferService = dicomTransferService;
32.         this.mapper = jacksonObjectMapper;
33.     }
34.
35.     @CrossOrigin(origins = "http://localhost:3000")
36.     @PostMapping("transmit")
37.
38.     public ResponseEntity<Object> forwardDicomData(
39.             @RequestHeader("dicomHost") String host,
40.             @RequestHeader("dicomPort") int port,
41.             @RequestHeader("calledAeTitle") String calledAeTitle,
42.             @RequestHeader("maxPduSize") int maxPduSize,
43.             @RequestParam("files[]") List<MultipartFile> files
44.     ) {
45.         ArrayNode responseNode = mapper.createArrayNode();
46.         for (MultipartFile file : files) {
47.             try {
48.                 Integer responseCode = dicomTransferService.transfer(
49.                                                 host,
50.                                                 port,
51.                                                 calledAeTitle,
52.                                                 maxPduSize,
53.                                                 file.getBytes()
54.                                             );
55.                 if (responseCode == null) {
56.                     addToResponseNode(responseNode, file, "Invalid file");
57.                     continue;
58.                 }
59.                 addToResponseNode(responseNode, file, responseCode.toString());
60.             } catch (IOException e) {
61.                 addToResponseNode(responseNode, file, e.getMessage());
62.             } catch (CommonDicomException e) {
63.                 addToResponseNode(responseNode, file, e.getMessage());
64.             }
65.         }
66.         return new ResponseEntity(responseNode, HttpStatus.OK);
67.     }
68.
69.     private void addToResponseNode(
70.         ArrayNode responseNode,
71.         MultipartFile file,
72.         String transmissionResponse
73.     ) {
74.         ObjectNode objectNode = mapper.createObjectNode();
75.         objectNode.put("name", file.getOriginalFilename());
76.         objectNode.put("transmissionResponse", transmissionResponse);
77.         responseNode.add(objectNode);
78.     }
79. }
```

Listing 6 DicomController (eigener Code)

```
1.  public Integer transfer(
2.          String host, int port, String calledAeTitle, int maxPduSize, byte[] data
3.      ) throws CommonDicomException {
4.          PacsConnectionFactory connectionFactory =
5.                  new PacsConnectionFactory()
6.                          .addDestination(host, port)
7.                          .addDicomSendMetaData(
8.                              callingAeTitle, calledAeTitle, maxPduSize
9.                      );
10.
11.         LOG.debug("Beginning transfer via dicom send process");
12.         DicomObject dicomObject = new DicomObject(data);
13.         Integer dicomResponseCode = null;
14.
15.         try (PacsConnection connection = openConnection(connectionFactory, dicomObject)) {
16.             connection.connect();
17.
18.             PresentationContext selectedPresentationContext = connection
19.                                                     .getUser()
20.                                                     .getSelectedPresentationContext();
21.
22.             CStoreRQ cStoreRQ = new CStoreRQ(
23.                 DimseHandler.getMessageID(),
24.                 getAffectedSOPClass(dicomObject),
25.                 dicomObject.getDicomElement(BaseDicomTagNames.TAG_sopInstanceUID)
26.                 .getValueAsString(0), data,
27.                 selectedPresentationContext
28.             );
29.
30.             CStoreRP dicomResponse = connection.sendDicomMessage(cStoreRQ,
31.                     FileMetaHeader.getFileMetaHeaderLenght(data));
32.             dicomResponseCode = dicomResponse.getStatus().getStatusCode();
33.             connection.getUser().release();
34.         } catch (CommonDicomException e) {
35.             return dicomResponseCode;
36.         } catch (NullPointerException e) {
37.             return dicomResponseCode;
38.         }
39.         return dicomResponseCode;
40.     }
```

Listing 7 transfer Methode (Unternehmensproprietär)

1. Einleitung

Im Rahmen der Health IT sind die Anforderungen an Anwendungen weitaus höher als in anderen Bereichen der Informatik. Computer werden in immer höherem Mase verwendet, um medizinische Daten zu verarbeiten. Die Entwicklung und Evaluation von Anwendungen, die für diesen Bereich entwickelt werden, ist enorm wichtig, denn Fehler in der Verarbeitung von Daten können von leichten bis hin zu tödlichen Folgen an Patienten führen. Um dies zu vermeiden, müssen medizinische Anwendungen Standards erfüllen. Hierbei ist das Testen dieser Anwendungen einer der Aspekte, der für die Erfüllung jener Standards eine bedeutende Rolle spielt. Ziel dieser Arbeit ist die Entwicklung und Evaluierung einer Webapp zum Testen eines Transfermoduls für medizinische Daten. Grund für die Entwicklung ist der Bedarf an einer Testsoftware, um die laufende Entwicklung der connect-bridge zu begleiten. Dafür soll eine Webbasierte-GUI erstellt werden. Um der Health IT entsprechend eine qualitativ hochwertige Anwendung zu entwickeln, wird ein Evaluationskriterium benötigt, welches den Ansprüchen für eine Anwendung in diesem Bereich angemessen ist.

Die Basis unserer Evaluation bildet die SQuaRE Reihe, welche von der Internationalen Organisation für Standards (ISO) aufgestellt wurden. Die Reihe stellt ein Produktqualitätsmodell für Software vor, welches aus 32 Qualitätsmerkmalen besteht. Außerdem stellt die Reihe eine Anleitung zur Verfügung, wie Anforderungen ermittelt und Evaluationen durchgeführt werden können. Die SQuaRE Reihe soll für diese Arbeit eine Richtlinie bilden, an der wir uns orientieren können, wenn es um jegliche Art von Entscheidungen geht, die mit der Planung, Entwicklung und Analyse unserer Anwendung zu tun haben.

Die vorliegende Arbeit prüft jedes einzelne Qualitätsmerkmal des Produktqualitätsmodells und erstellt daraus Anforderungen für unsere Webapp. Anhand dieser Anforderungen werden User Stories für das Projekt erstellt. Der praktische Teil dieser Arbeit, ist die Entwicklung der Webapp, welche als mögliche Lösung für die Erfüllung der Anforderungen erforderlich ist. Abschließend werden die ermittelten Ergebnisse zusammen gefasst und betrachtet ob und wie die Anforderungen an das Projekt erfüllt werden. Hierbei sollen auch weiterführende Gedanken und Anregungen für Verbesserungen mit aufgenommen werden.

2. Technologie Stack

2.1. DICOM

Bei DICOM handelt es sich um einen internationalen Standard zum Senden, Speichern, Abrufen, Drucken, Verarbeiten und Anzeigen medizinischer Bildinformationen. DICOM wird von der ISO als eigener Standard anerkannt und kann unter ISO 12052 eingesehen werden (dicomstandard.org).

2.2. connect-bridge

Die connect-bridge, ist die von Visus Health IT GmbH entwickelte Anwendung in Form einer Spring Boot Anwendung. Ihr Zweck ist die Prüfung und der Transfer von DICOM Daten. Die connect-bridge wird im Produktionsbetrieb als Windows-Dienst ausgeführt.

2.3. React

Bei React handelt es sich um eine vom Unternehmen Meta entwickelte JavaScript-Bibliothek zur Erstellung von Webbasierten Benutzeroberflächen (reactjs.org 2022). React kann sowohl in JavaScript als auch TypeScript programmiert werden. Für die vorliegende Arbeit wird die TypeScript Variante verwendet.

2.4.Spring Boot

Spring Boot ist Teil des Spring Frameworks und bietet Entwicklern die Möglichkeit einen auf Java basierten Webserver aufzusetzen (spring.io). Das Framework wird als Backend für die Tuschi verwendet und bildet die Schnittstelle bei der Kommunikation zwischen dem Benutzer und dem Zielsystem.

2.5.Git

Git ist eine kostenlose open source Anwendung, welche für die Versionierung von Projekten verwendet wird, was bedeutet, dass mehrere Entwickler gleichzeitig an dem Projekt arbeiten können (git-scm.com).

3. Evaluationsmethode

Bei Evaluation handelt es sich nach Balzer um eine Bewertung bzw. Begutachtung von Projekten, Prozessen und Funktionseinheiten. (Balzer, Frey und Nenniger 1999). Das Projekt ist die Entwicklung der Anwendung zum Testen der Transfermoduls. Als Prozesse sind jene Handlungen abgebildet, die zur Erfüllung der Projektanforderungen dienen. Im Falle der vorliegenden Arbeit sind das:

- Wahl der Tools und Frameworks
- Vorgehen bei der Entwicklung

Die Funktionseinheiten, welche wir betrachten, seien die Ergebnisse dieser Handlungen. Im vorliegenden Fall ist dies der Programmcode.

Es gibt unterschiedliche Modelle, auf Basis derer Anwendungen bewertet werden können. Lange präsentiert hierfür in einer Ausarbeitung zu Softwarequalitätsmodellen zwei Modelle, die zur Bewertung von Software verwendet werden können. Zum einen das Qualitätsmodell nach McCall. Das Softwarequalitätsmodell nach McCall stammt aus dem Jahre 1977 und ist ein dreistufiges Modell, bestehend aus elf Qualitätshauptzielen, den Qualitätsfaktoren, aus Qualitätskriterien zu jedem Faktor und aus Kenngrößen und Metriken. (Lange 2010) Die Qualitätshauptziele sind folgende:

Korrektheit, Zuverlässigkeit, Effizienz, Integrität, Benutzbarkeit, Wartbarkeit, Testbarkeit, Flexibilität, Portabilität, Wiederverwendbarkeit, Verknüpfbarkeit

Das zweite Modell, welches Lange ausarbeitet ist das Qualitätsmodell nach Norm ISO 9126. Dieses Modell ist allerdings nicht mehr aktuell und wurde in den ISO Standard 25010 überführt und überarbeitet, weshalb sich weitere Erwähnungen auf den aktuelleren Standard beziehen. ISO 25010 definiert ein Produktqualitätsmodell, welches aus 8 Hauptmerkmalen besteht, welche sich wiederum in 32 Untermerkmale aufteilen:

- Funktionale Eignung (Angemessenheit, Korrektheit, Vollständigkeit)
- Wartbarkeit (Modularität, Modifizierbarkeit, Wiederverwendbarkeit, Analysierbarkeit, Prüfbarkeit)
- Kompatibilität (Koexistenz, Interoperabilität)
- Benutzbarkeit (Verständlichkeit, Wiedererkennbarkeit, Erlernbarkeit, Bedienbarkeit, Fehlervermeidung, Ästhetik, Barrierefreiheit)
- Leistungseffizienz (Kapazität, Zeitverhalten, Verbrauchsverhalten)
- Zuverlässigkeit (Reife, Fehlertoleranz, Wiederherstellbarkeit, Verfügbarkeit)
- Übertragbarkeit (Installierbarkeit, Austauschbarkeit, Anpassbarkeit)
- Sicherheit (Zurechenbarkeit, Nachweisbarkeit, Authentizität, Vertraulichkeit, Datenintegrität)

Die beiden Modelle sind sich sehr ähnlich. Beide Modelle definieren Kriterien, anhand derer Software bewertet werden kann. Für diese Arbeit wird das Modell nach ISO verwendet, da es alle Kriterien aus dem Modell von McCall aufnimmt und die Aufteilung in die vielen Unterkategorien eine facettenreichere Evaluation unserer Anwendung erlauben. Darüber hinaus bietet der Standard eine Vorlage, die die Anwendung des Modells auf die Realität beschreibt. Wichtig sei hervorzuheben, dass das Modell eher als Richtlinie zu betrachten ist, die der Anwender seinen Bedürfnissen anpassen kann (ISO 20510).

3.1. Abbildung des Modelles auf die Tuschi

Wie in Abschnitt 2 bereits erarbeitet, kann das Qualitätsmodell auf die Bedürfnisse der Nutzer der Modelle zugeschnitten werden. Nicht alle Qualitätsmerkmale eigenen sich für die Anwendung auf Die Tuschi (ISO/IEC 25010). ISO/IEC 25020 der SQuaRE Reihe beschreibt den Prozess zur Sicherstellung eines Qualitätsmerkmals. Demnach werden für alle benötigten Qualitätsmerkmale eine oder mehrere Qualitätsmessungen durchgeführt. Dabei kann jede Messung aus einer oder mehreren Messfunktionen bestehen. Mögliche Messfunktionen werden hierbei in ISO/IEC 25023 aufgelistet. Aufgrund der Flexibilität der Modelle besteht keine Einschränkung in Hinsicht auf die Erstellung eigener Messfunktionen. Abschnitt 2.2.1 definiert die Qualitätsmerkmale und mappt die für nötig empfundenen Merkmale auf die Tuschi, um daraus Anforderungen und Messungen zur Erfüllung dieser Merkmale bereit zu stellen.

3.1.1. Mapping der Kriterien

Funktionale Eignung: Hauptmerkmal. Grad, in dem ein Produkt oder System Funktionen bereitstellt, die den angegebenen und impliziten Bedürfnissen entsprechen, wenn es unter bestimmten Bedingungen verwendet wird.

Vollständigkeit: Untermerkmal von funktionaler Eignung. Grad, in dem der Satz von Funktionen alle spezifizierten Aufgaben und Benutzerziele abdeckt.

Anforderungen:

- Erfüllung der Akzeptanzkriterien der User Stories

User Story 1:

Als Entwickler möchte ich eine oder mehrere Dateien von meiner Festplatte in eine Liste hinzufügen können, um später eine, mehrere oder alle davon an die connect-bridge senden zu können.

Akzeptanzkriterien:

- A-1: Es muss keine Prüfung des Dateityps stattfinden, da das Zielsystem falsche Dateien ablehnt.
- A-2: Doppelt hinzugefügte Dateien sollen nicht erneut zur Liste hinzugefügt werden.
- A-3: Es soll nur der Dateiname - ohne Pfad - in der Liste angezeigt werden.

User Story 2:

Als Entwickler möchte ich eine Datei auswählen und an die connect-bridge senden, um die laufende Entwicklung der connect-bridge zu testen.

Akzeptanzkriterien:

- A-4: Falls das empfangende System (connect-bridge) mit einer Fehlermeldung antwortet, möchte ich diese direkt angezeigt bekommen.
- A-5: Die Fehlermeldung vom empfangenden System kann 1:1 weitergegeben werden und muss nicht übersetzt werden o.ä.
- A-6: Unabhängig von Erfolg oder Fehler, soll die zuletzt ausgewählte Datei ausgewählt und der Fokus auf dem Absenden-Button bleiben, so dass ich die Datei sofort wieder absenden kann.
- A-7: Im Erfolgsfall soll eine Meldung "OK" erscheinen

User Story 3:

Als Entwickler möchte ich mehrere Dateien auf einmal auswählen können, um sie mit einem Klick nacheinander an die connect-bridge senden zu können.

Akzeptanzkriterien:

- A-8: Funktionalität wie beim einzelnen Senden, aber nur eine Datei auf einmal soll gesendet werden.
- A-9: Während eine Übertragung von mehreren Dateien läuft, soll der Button gesperrt sein, damit keine parallelen Übertragungen gestartet werden können.

Messung:

- Anzahl erfüllter Akzeptanzkriterien / Anzahl vorhandener Akzeptanzkriterien

Korrektheit: Untermerkmal von funktionaler Eignung. Untermerkmal von Funktionale Eignung. Grad, in dem ein Produkt oder System die richtigen Ergebnisse mit dem erforderlichen Grad an Präzision liefert.

Anforderungen:

- Funktionale Korrektheit wird erreicht, indem Unit-Tests für die Implementierungen der Funktionen geschrieben werden.

Messung:

- Prüfung der Testabdeckung. Festlegung eines Prozentsatzes der widerspiegelt, wie hoch der Anteil an abgedecktem Code sein muss.

Angemessenheit: Untermerkmal von funktionaler Eignung. Das Ausmaß, in dem die Funktionen das Erreichen Bestimmter Aufgaben und Ziele erleichtern.

- Funktionale Angemessenheit wird erreicht, indem der Benutzer die Tuschi in einem möglichst Ressourcen sparenden Kontext nutzen kann

Leistungseffizienz: Hauptmerkmal. Leistung im Verhältnis zur Menge der verwendeten Ressourcen unter bestimmten Bedingungen. Ressourcen können andere Softwareprodukte- und Hardwarekonfiguration des Systems und Materialen wie Speichermedien umfassen.

Zeitverhalten: Untermerkmal von Leistungseffizienz. Grad, in dem die Reaktions- und Verarbeitungszeiten sowie die Durchsatzraten eines Produkts oder Systems bei der Ausführung seiner Funktionen den Anforderungen entsprechen.

Anforderungen:

- Aufgrund der Benutzergruppe ist das Zeitverhalten der Tuschi nicht relevant.

Verbrauchsverhalten: Untermerkmal von Leistungseffizienz. Grad, in dem die Mengen und Arten von Ressourcen, die von einem Produkt oder System bei der Ausführung seiner Funktionen verwendet werden, den Anforderungen entsprechen.

Kapazität: Untermerkmal von Leistungseffizienz. Grad, in dem die Höchstwerte eines Produkt- oder Systemparameters den Anforderungen entsprechen.

Anforderungen:

- DICOM Daten können mehrere Gigabytes groß werden. Es ist bei der Entwicklung darauf zu achten, dass eine obere Grenze bei der Hardware der Benutzer der Tuschi vorliegt.

Messung: Berechnung der maximal erlaubten Dateigroße, die in die Tuschi geladen werden darf.

User Story 4:

Als Entwickler möchte ich sicher sein, dass keine zu großen Dateien an die connect-bridge gesendet werden können, um einen Absturz der Tuschi zu verhindern.

Akzeptanzkriterien:

- A-10: Die maximale Dateigroße, die der Benutzer auswählen darf beträgt den errechneten maximalen Betrag.

Kompatibilität: Hauptmerkmal. Grad, in dem ein Produkt, System oder Bauteil Informationen mit anderen Produkten, Systemen oder Bauteilen austauschen und/oder seine erforderlichen Funktionen ausführen kann, während es dieselbe Hardware- oder Softwareumgebung nutzt.

Koexistenz: Untermerkmal von Kompatibilität. Grad, in dem ein Produkt seine erforderlichen Funktionen effizient ausführen kann, während es sich eine gemeinsame Umgebung und Ressourcen mit anderen Produkten teilt, ohne dass dies nachteilige Auswirkungen auf ein anderes Produkt hat.

Interoperabilität: Untermerkmal von Kompatibilität. Grad, in dem zwei oder mehr Systeme, Produkte oder Komponenten Informationen austauschen und die ausgetauschten Informationen nutzen können.

Benutzbarkeit: Hauptmerkmal. Grad, in dem ein Produkt oder System von bestimmten Nutzern verwendet werden kann, um bestimmte Ziele mit Effektivität, Effizienz und Zufriedenheit in einem bestimmten Nutzungskontext zu erreichen.

Anforderungen:

- Relevant, da vor allem neue Entwickler, die sich an der laufenden Entwicklung der connect-bridge beteiligen, von einer leicht verständlichen UI zum Testen profitieren.

Verständlichkeit: Untermerkmal von Benutzbarkeit. Grad, in dem die Nutzer erkennen können, ob ein Produkt oder System für ihre Bedürfnisse geeignet ist.

Anforderungen:

- Ein Entwickler verwendet die Tuschi, um die Datenübertragung mit der connect-bridge zu testen. Dabei möchte er ohne weitere Hilfestellung die Tuschi bedienen können.

Messung:

- Usability Tests
- Für die Tests werden mindestens 5 Testpersonen ausgewählt, welche der Benutzergruppe entsprechen
- Testdurchführung:
    - Die Testperson soll eine beliebige Anzahl an Daten in die Liste laden
    - Die Testperson soll eine Datei an die connect-bridge senden
    - Die Testperson soll alle Dateien an die connect-bridge senden
    - Die Testperson soll den Übertragungsstatus der Dateien vorlesen/aufschreiben

Wiedererkennbarkeit: Untermerkmal von Benutzbarkeit. Grad, in dem die Nutzer erkennen können, ob ein Produkt oder System für ihre Bedürfnisse geeignet ist.

Anforderungen:

- Ähnlich wie bei der Verständlichkeit soll der Benutzer die Elemente und ihre Bedeutung möglichst schnell erkennen können.

Erlernbarkeit: Untermerkmal von Benutzbarkeit. Grad, in dem ein Produkt oder System von bestimmten Nutzern verwendet werden kann, um bestimmte Ziele zu erreichen, nämlich zu lernen,

das Produkt oder System mit Effektivität, Effizienz, Risikofreiheit und Zufriedenheit in einem bestimmten Nutzungskontext zu verwenden.

Anforderungen:

- Erlernbarkeit zweitrangig aufgrund der Expertise der Benutzergruppe.

Bedienbarkeit: Untermerkmal von Benutzbarkeit. Grad, in dem ein Produkt oder System Eigenschaften aufweist, die seine Bedienung und Kontrolle erleichtern.

Anforderungen:

- Wichtig eventuell für neue Entwickler ansonsten geht so da sich alle damit auskennen.

Fehlervermeidung: Untermerkmal von Benutzbarkeit. Grad, in dem ein System die Benutzer vor Fehlern schützt.

Anforderungen:

- Äußerst relevant, da Entwickler sich sicher sein müssen, welche Anwendung an bestimmten Stellen Fehler wirft.

Messung:

- Wie bei der funktionalen Korrektheit auch, kann die Anforderung erfüllt werden, durch Erstellung von Unit- oder UI-Tests für die Tuschi.

Ästhetik: Untermerkmal von Benutzbarkeit. Ausmaß, in dem eine Benutzeroberfläche eine für den Benutzer angenehme und zufriedenstellende Interaktion ermöglicht

Anforderungen:

- Ästhetik zweitrangig aufgrund des Nutzungskontext.

Barrierefreiheit: Untermerkmal von Benutzbarkeit. Grad, in dem ein Produkt oder System von Menschen mit den unterschiedlichsten Eigenschaften und Fähigkeiten genutzt werden kann, um ein bestimmtes Ziel in einem bestimmten Nutzungskontext zu erreichen.

Anforderungen:

- Barrierefreiheit zweitrangig aufgrund des Nutzungskontext.

Zuverlässigkeit: Hauptmerkmal. Grad, in dem ein System, Produkt oder Bauteil bestimmte Funktionen unter bestimmten Bedingungen über einen bestimmten Zeitraum erfüllt.

Anforderungen:

- Die Tuschi soll alle Funktionen zur Verfügung stellen, wenn sie von dem Benutzer benötigt werden, solange er sie benötigt.

Verfügbarkeit: Untermerkmal von Zuverlässigkeit. Grad, in dem ein System, ein Produkt oder eine Komponente betriebsbereit und zugänglich ist, wenn es für die Nutzung erforderlich ist.

Anforderungen:

- Die Tuschi soll ihre Funktionen immer genau dann zur Verfügung stellen, wenn sie grade von einem Entwickler benötigt werden, für den Zeitraum, den er sie nutzt. Ein dauerhafter Betrieb muss nicht gewährleistet sein, da im Notfall der Test über die Konsole noch immer möglich ist.

Wiederherstellbarkeit: Untermerkmal von Zuverlässigkeit. Grad, in dem ein Produkt oder System im Falle einer Unterbrechung oder eines Ausfalls die unmittelbar betroffenen Daten wiederherstellen und den gewünschten Zustand des Systems wiederherstellen kann.

Anforderungen:

- Wiederherstellbarkeit zweitrangig, da die Tuschi nicht der einzige Weg ist die Uschi zu testen.

Fehlertoleranz: Untermerkmal von Zuverlässigkeit. Grad, in dem ein System, ein Produkt oder eine Komponente trotz des Vorhandenseins von Hardware- oder Softwarefehlern wie vorgesehen funktioniert.

Anforderungen:

- Im Absturzfall soll die Tuschi sich neu starten

Reife: Untermerkmal von Zuverlässigkeit. Grad, in dem ein System, Produkt oder Bauteil die Anforderungen an die Zuverlässigkeit bei normalem Betrieb erfüllt.

Sicherheit: Hauptmerkmal. Grad, in dem ein Produkt oder System Informationen und Daten schützt, so dass Personen oder andere Produkte oder Systeme den ihrer Art und Berechtigung entsprechenden Grad an Datenzugriff haben.

Anforderungen:

- Da es sich um ein Testtool handelt, welches nicht im Produktionsbetrieb mit relevanten medizinischen Daten arbeitet, besteht keine Anforderung an die Sicherheit der Anwendung.

Zurechenbarkeit: Untermerkmal von Sicherheit. Grad, in dem die Handlungen einer Einheit eindeutig auf diese Einheit zurückgeführt werden können

Nachweisbarkeit: Untermerkmal von Sicherheit. Grad, in dem Handlungen oder Ereignisse nachweislich stattgefunden haben, so dass die Ereignisse oder Handlungen später nicht mehr bestritten werden können.

Authentizität: Untermerkmal von Sicherheit. Grad, in dem die Identität eines Subjekts oder einer Ressource als die behauptete nachgewiesen werden kann.

Vertraulichkeit: Untermerkmal von Sicherheit. Grad, in dem ein Produkt oder System sicherstellt, dass Daten nur denjenigen zugänglich sind, die dazu berechtigt sind.

Datenintegrität: Untermerkmal von Sicherheit. Grad, in dem ein System, ein Produkt oder eine Komponente den unbefugten Zugriff auf oder die Veränderung von Computerprogrammen oder Daten verhindert.

Wartbarkeit: Hauptmerkmal. Grad der Effektivität und Effizienz, mit der ein Produkt oder System von den vorgesehenen Betreuern verändert werden kann.

Anforderungen:

- Die Tuschi soll von jeder Person mit Zugriff auf den Quellcode, und der Berechtigung diesen zu verändern, gewartet werden können.

Modularität: Untermerkmal von Wartbarkeit. Grad, in dem ein System oder Computerprogramm aus einzelnen Komponenten besteht, so dass die Änderung einer Komponente nur minimale Auswirkungen auf andere Komponenten hat.

Modifizierbarkeit: Untermerkmal von Wartbarkeit. Grad, in dem ein Produkt oder System effektiv und effizient verändert werden kann, ohne dass es zu Fehlern oder einer Verschlechterung der bestehenden Produktqualität kommt.

Anforderungen:

- Wichtig, da sich an der Uschi etwas ändern kann was dann auch an der Tuschi geändert werden muss um die Testen zu können
- Code soll im geringen Aufwand modifizierbar sein, da Anpassungen an der connect-bridge zu Anpassungen an der Tuschi führen können

Wiederverwendbarkeit: Untermerkmal von Wartbarkeit. Grad, in dem ein Vermögenswert in mehr als einem System oder beim Bau anderer Vermögenswerte verwendet werden kann

Analysierbarkeit: Untermerkmal von Wartbarkeit. Grad an Effektivität und Effizienz, mit dem es möglich ist, die Auswirkungen einer beabsichtigten Änderung an einem oder mehreren Teilen eines

Produkts oder Systems zu bewerten, ein Produkt auf Mängel oder Fehlerursachen hin zu diagnostizieren oder zu ändernde Teile zu identifizieren.

Prüfbarkeit: Untermerkmal von Wartbarkeit. Grad der Effektivität und Effizienz, mit dem Prüfkriterien für ein System, ein Produkt oder ein Bauteil festgelegt und Prüfungen durchgeführt werden können, um festzustellen, ob diese Kriterien erfüllt wurden.

Übertragbarkeit: Hauptmerkmal. Grad der Effektivität und Effizienz, mit dem ein System, ein Produkt oder eine Komponente von einer Hardware-, Software- oder sonstigen Betriebs- oder Nutzungsumgebung auf eine andere übertragen werden kann.

Anpassbarkeit: Untermerkmal von Übertragbarkeit. der Grad, in dem ein Produkt oder System effektiv und effizient an unterschiedliche oder sich entwickelnde Hardware-, Software- oder andere Betriebs- oder Nutzungsumgebungen angepasst werden kann.

Austauschbarkeit: Untermerkmal von Übertragbarkeit. Grad, in dem ein Produkt ein anderes spezifiziertes Softwareprodukt für denselben Zweck in derselben Umgebung ersetzen kann.

Installierbarkeit: Untermerkmal von Übertragbarkeit. Grad der Effektivität und Effizienz, mit der ein Produkt oder System in einer bestimmten Umgebung erfolgreich installiert und/oder deinstalliert werden kann.

Anforderungen:

- Tuschi soll in einer Docker Umgebung gestartet werden können.

User Story 5:

Als Entwickler möchte ich einen Konsolenbefehl ausführen, um eine Betriebsbereite Tuschi in einem docker Container zu starten.

4. Erfüllung der Anforderungen

4.1. Allgemeiner Aufbau der Tuschi

Der folgende Teil der Arbeit vermittelt ein tieferes Verständnis für den Aufbau der Tuschi und das Vorgehen bei dessen Entwicklung. Eine Skizzierung des Aufbaus der Tuschi kann Abbildung 1 entnommen werden. Das Frontend besteht aus einer React Anwendung mit Typescript als Programmiersprache. Der Benutzer interagiert nun mit dem React Frontend, um die Dateien von seiner Festplatte in die Tuschi zu laden. Diese Dateien werden anschließend via HTTP Post Request an das Spring Backend gesendet. Anschließend werden die Dateien an die connect-bridge gesendet. Die Verarbeitung der Dateien in der connect-bridge findet mit Unternehmensproprietärer Software statt. Die connect-bridge antwortet auf die Request des Backends nach Verarbeitung der

Daten. Die dadurch erhaltene Response enthält den Responsecode der Übertragung, welcher zurück an das React Frontend gesendet und dem Benutzer angezeigt wird.

Die Entscheidung dafür die React Bibliothek zur Entwicklung der Tuschi zu verwenden, liegt in der einfachen Erweiterbarkeit der Anwendung über node modules, welche über den mit React bereitgestellten node package manager hinzugefügt werden können. Dadurch wird die Tuschi den Qualitätsanforderungen für Wartbarkeit und Modifizierbarkeit gerecht.

Die Entscheidung dafür Spring Boot für die direkte Interaktion mit der connect-bridge einzusetzen ergibt sich aus der Suche nach einer funktional angemesseneren Lösung, die vom Frontend allein übernommen werden kann. Zum Zeitpunkt der Entwicklung existiert keine den Anforderungen entsprechende Lösung, um das Frontend allein mit der connect-bridge kommunizieren zu lassen. Da es bereits eine vom Unternehmen selbst entwickelte Lösung gibt, welche auf Spring Boot basiert und mit der connect-bridge kommuniziert, ist dies die nächst-angemessene Lösung zur Erfüllung der funktionalen Angemessenheit.

Ein Screenshot der UI kann Abbildung 2 entnommen werden. Die Ordnerstrukturen von Frontend und Backend können Abbildungen 3 und 4 entnommen werden.

Die Implementation der Tuschi beginnt bei User Story 1 der Funktionalen Anforderungen, bearbeitet dann User Story 2 und anschließend User Story 3. Dementsprechend folgt die Erklärung zur Implementation auch der Reihenfolge, wie sie den User Stories entspricht. Der Umfang dieser Arbeit umfasst auch nur die Implementation der ersten 3 User Stories. Alle weiteren User Stories sind in dieser Arbeit, aufgrund des Umfanges der Implementation, nicht enthalten.

4.2. Implementierung User Story 1

Das Frontend besteht aus zwei funktionalen React Komponenten. Zum einen der FileTable Komponente (Listing 4) und der FileTableRow Komponente (Listing 3). Die beiden Komponenten haben eine 1 zu n Beziehung. 1 FileTable beinhaltet n FileTableRows.

Zunächst wird ein Element zum Interagieren für den Benutzer benötigt, damit er Dateien von seiner Festplatte in die Anwendung laden kann. Hierfür wird ein Button Element angelegt (Listing 4 Zeilen 54-73), welches ein input Element beinhaltet. Dem *type* Attribut des input Elements wird der Wert “file“ zugewiesen. Die Zuweisung dieses Schlüsselwortes weist dem input Element die Funktion zu das File Explorer Fenster des Betriebssystems des Benutzers zu öffnen, wenn es angeklickt wird. Das *multiple* Attribut erlaubt es dem Benutzer mehrere Dateien gleichzeitig auszuwählen. Durch das Weglassen des *accept* Attributes werden alle Dateitypen akzeptiert, was für Akzeptanzkriterium A-1 verlangt wird. Die Funktion in *onChange* wird aufgerufen, sobald der Benutzer Dateien ausgewählt hat. Das Event e in Listing 4 Zeile 65 beinhaltet die vom Benutzer ausgewählten Dateien. In Listing 4 Zeile 67 wird aus allen Dateien ein Array erzeugt. In Listing 4 Zeile 68 wird anschließend der

Handler aufgerufen, welcher die Dateien in einem state der FileTable Komponente speichert. Der State wird hier *fileStore* genannt. Der Handler in Listing 4 Zeile 68 wandelt das Array vom Typ File in ein Array vom Typ DicomFile um. Dieser Datentyp ist eine Hilfe für die weitere Verarbeitung der Dateien im späteren Verlauf der Entwicklung. Der Aufbau des Datentyps kann Abbildung 3 entnommen werden. Um Akzeptanzkriterium A-2 zu erfüllen, muss vor dem Speichern der Dateien in den state eine Überprüfung stattfinden, um bereits vorhandene Dateien aus der Liste auszusortieren. Mit den Funktionen *containsFile* (Listing 4 Zeilen 16-17) und *mergeFiles* (Listing 4 Zeilen 19-23) wird sichergestellt, dass nur dem Namen nach einzigartige Dateien in den state aufgenommen werden.

Listing 4 Zeilen 85 – 111 implementieren die Liste in Form einer Tabelle, in der die Dateien referenziert werden. Listing 4 Zeilen 90 – 100 bilden den Header der Tabelle, welche aus drei Spalten besteht:

- *Filename*: In dieser Spalte wird der Dateiname angezeigt.
- *Response*: Diese Spalte beinhaltet den Übertragungsstatus der Datei, nachdem diese an die connect-bridge gesendet wurde.
- Die dritte Spalte des Headers hat keinen Titel. In dieser Spalte werden die Aktionen aufgeführt, welche mit der Datei durchgeführt werden können.

Listing 4 Zeilen 101 – 109 bilden den Körper der Tabelle. In der Implementation wird über fileStore iteriert, und für jedes Element eine FileTableRow angelegt. Jede FileTableRow hat dabei folgende props:

- *key*: Da es sich hier um eine Liste handelt benötigt React ein key prop, um die Komponenten voneinander unterscheiden zu können. Hierfür kann der name der Datei genommen werden, da dieser innerhalb des filesStore immer einzigartig ist.
- *dicomFile*: die im fileStore gespeicherte Datei wird an die Komponente übergeben.
- *tableIsTransmitting*: Jeder FileTableRow Komponente wird mitgeteilt, ob die Tabelle bereits eine Übertragung aller Dateien an die connect-bridge durchführt.

Eine FileTableRow Komponente besteht dem Header der Tabelle entsprechend aus 3 Elementen (Listing 3 Zeilen 39-50). Das erste Element beinhaltet den Dateinamen. Dies wird zur Erfüllung von Akzeptanzkriterium A-3 benötigt. Das zweite Element beinhaltet die Response der Übertragung der Datei an die connect-bridge, nachdem sie versendet wurde. Das dritte Element beinhaltet einen Button, mit einer onClick Funktion, welche einen Handler zum Setzen des Übertragungsstatus enthält.

4.3. Implementierung User Story 2

4.3.1. Backendimplementierung

Mit der zweiten User Story soll eine Möglichkeit bereitgestellt werden, die vom React Frontend gesammelten Dateien an die connect-bridge zu senden. An dieser Stelle kommt das Spring Boot Backend zum Einsatz. Abbildung 7 zeigt den Quellcode des DicomControllers, welcher die Request an die connect-bridge sendet. Der Controller wird mit einem Service vom Typ DicomTransferService initialisiert (Listing 6 Zeile 23), welcher dafür eingesetzt wird eine Kommunikation mit der connect-bridge aufzubauen. Der Service und alle darin importieren Dateien sind Teil Unternehmensproprietärer Software. Die genaue Funktionsweise des Service wird aufgrund des Rahmens dieser Arbeit nicht weiter vertieft. Das zweite Objekt, welches mit dem Controller initialisiert wird ist ein Objekt vom Typ ObjectMapper (Listing 6 Zeile 24), welches Teil des importierten fasterxml Moduls ist. Dieses Objekt wird benötigt, um die Response für das Frontend zusammenzusetzen.

Der Controller verfügt über genau einen Endpunkt welcher über localhost:8080/transmit angesprochen werden kann. Er bekommt als Parameter eine Liste vom Type MultipartFile übertragen (Listing 6 Zeile 43). Des Weiteren müssen im Header der Request die Adressdaten der connect-bridge übergeben werden (Listing 6 Zeilen 39-42). Die Header übergeben folgende Daten an die Request:

- *dicomHost:* Die Host-Adresse der connect-bridge. Läuft die connect-bridge als Windows-Dienst auf dem ausführendem Betriebssystem, hört sie auf ‚localhost'.
- *dicomPort:* Der Port der connect-bridge. Zum Zeitpunkt der Entwicklung hört die connect-bridge auf Port 104. Dieser Wert kann sich im Laufe der Zeit ändern.
- *calledAeTitle:* Dieser Wert entspricht einer Passphrase und hat zum Zeitpunkt der Entwicklung den Wert ‚connect-box'. Auch dieser Wert kann sich im Laufe der Zeit ändern.
- *maxPduSize:* Dieser Wert beträgt ‚32768'.

Der *files* Parameter aus Listing 6 Zeile 43 ist das Array an Dateien, welches vom Benutzer vom Frontend übertragen wird. In Listing 6 Zeile 45 wird das Objekt *responseNode* vom Typ ArrayNode erzeugt, in das die ResponseCodes der Übertragungen geschrieben werden. Die Response am Ende hat folgenden beispielhaften Aufbau:

```
[
        { filename: “Dateiname1“, transmissionResponse: “0“ },
        { filename: “Dateiname2“, transmissionResponse: “Invalid file“ }
]
```

Es wird über jede Datei innerhalb der Liste iteriert. Während jeder Iteration wird versucht der ResponseCode der Übertragung ermittelt zu werden, indem der dicomTransferService eine Verbindung zur connect-bridge aufbaut und die Daten aus dem Header, sowie die Datei übermittelt (Listing 6 Zeilen 48-54). Der Quellcode der in Listing 6 Zeile 48 aufgerufenen Funktion *transfer* kann Abbildung 8 entnommen werden. Ist die übermittelte Datei keine DICOM Datei wird auch kein ResponseCode zurück übermittelt. In dem Fall wird als Response der String “Invalid file“ eigentragen und zusammen mit dem Dateinamen der responseNode übergeben. Listing 6 Zeilen 60-64 fangen mögliche Fehler bei der Verarbeitung der Daten ab und schreiben diese Rückmeldung dann mit dem Dateiname zusammen in die responseNode. Damit wird Akzeptanzkriterium A-5 umgesetzt. Dieser Prozess wiederholt sich, bis alle Dateien der Liste übertragen wurden. Anschließend wird die responseNode an den Aufrufer des Endpunktes zurückgegeben (Listing 6 Zeile 66).

4.3.2. Frontendimplementierung

Das React Frontend soll eine Möglichkeit bereit stellen mit dem Spring Boot Backend zu kommunizieren. Die Implementation des Transmitters kann Listing 1 entnommen werden. Listing 1 Zeilen 4-9 beschreiben die Konfiguration der Header der Anfrage. Listing 1 Zeilen 11-37 beschreiben die Funktion *transmit*, welche eine asynchrone Funktion ist, die dem Aufrufer einen Promise mit den Daten der Übertragung zurück gibt, wie in Abschnitt 4.3.1 dargestellt. Für das Erstellen der Request wird “axios“ verwendet, welches über den node package manager dem Projekt hinzugefügt wird. Das Modul erlaubt das Erstellen von http Requests.

Die *transmit* Funktion wird anschließend den Komponenten bereitgestellt. In dem Handler des Send Buttons der FileTableRow Komponente wird die *transmit* Funktion aufgerufen und die Datei, die von der Komponente gehalten wird übergeben (Listing 3 Zeilen 25-31). Bekommt die Anwendung eine Rückmeldung von “0“ ist die Übertragung erfolgreich verlaufen und dem Benutzer soll im Frontend der Wert “OK“ angezeigt werden. In allen anderen Fällen soll die Rückmeldung 1:1 weitergegeben werden. Damit werden die Akzeptanzkriterien A-4 und A-7 erfüllt.

4.4. Implementierung User Story 3

Zum Erfüllen der dritten User Story soll dem Benutzer die Möglichkeit gegeben werden, alle Dateien gleichzeitig an das System zu senden. Der Endpunkt im Spring Boot Backend ist bereits dafür angepasst mehrere Dateien gleichzeitig entgegen zu nehmen. Im Frontend kann wieder die *transmit* Methode aus Listing 1 verwendet werden, da auch diese bereits ein Array an Dateien entgegen nehmen kann. Listing 4 Zeilen 37-49 beschreiben den Handler, der aufgerufen wird, wenn der Send All Button in der UI geklickt wird. Der Handler iteriert über alle im fileStore enthaltenen Dateien und vergleicht die Namen mit denen der Response. Bei einer Übereinstimmung wird die Response in die transmissionResponse des DicomFile geschrieben. Die Implementierung dieser Funktion erfüllt Akzeptanzkriterium A-8.

Zur Erfüllung von Akzeptanzkriterium A-9 wird zunächst der FileTable Komponente ein neuer state hinzugefügt, welcher *tableIsTransmitting* genannt wird und einen boolschen Wert hält, welcher im default fall false ist. Dieser state wird dem disabled Attribut des Send All Buttons hinzugefügt (Listing 4 Zeile 106). Sobald der Handler in der onClick Funktion des Buttons aufgerufen wird, wird der Wert des states auf true gesetzt und der Button kann nicht mehr angeklickt werden, bis die transmit Funktion alle Daten verarbeitet hat. Anschließend wird der state wieder auf false gesetzt und der Button kann wieder angeklickt werden.

4.5. Erfüllung der funktionalen Korrektheit

Die Anforderungen an das Qualitätsmerkmal der funktionalen Korrektheit erfordern eine Testabdeckung der implementierten Funktionen. In React können Unit Tests mit Hilfe der react testing library umgesetzt werden. Listing 5 zeigt den Quellcode der Unit Test Implementierung für die FileTable Komponente. Die react testing library wird in Listing 5 Zeilen 2-3 importiert. Der describe Block (Listing 5 Zeilen 6 bis 42) beschreibt einen zusammenhängenden Testblock. Listing 5 Zeilen 7-9 stellen gemockte Ressourcen zur Verfügung, die von allen Tests verwendet werden können.

Die Syntax für einen Test ist folgende:

```
it('Testbeschreibung', () => { Körper der Testfunktion })
```

Zur Verdeutlichung kann der Test 'test_FileTable_whenFileAdded_expectItToBeInTheDocument' im Detail erklärt werden:

Listing 5 Zeile 16: Die Komponente wird gerendert

Listing 5 Zeile 17: Der Button für den Fileupload, welcher als Teil der FileTable Komponente gerendert wird, kann anhand der screen.getByLabelText Funktion in einer Konstante gespeichert werden.

Listing 5 Zeile 18: das userEvent Objekt hat eine upload Funktion, welche die Situation simuliert, bei der ein Benutzer Dateien von seiner Festplatte über das File Explorer Fenster auswählt.

Listing 5 Zeile 19: Die Funktion getByText des screen Objektes versucht exakt ein Objekt im Dokument zu finden, welches den Wert 'dateiname1' enthält. Sollte mehr als eins gefunden werden schlägt der Test fehl.

Listing 5 Zeile 21: Es erfolgt die Auswertung des Tests. Sollte das zuvor gesuchte Element nicht im Dokument vorhanden sein schlägt der Test fehl.

Die in Listing 5 enthaltenen Tests decken die in User Story enthaltenen Anforderungen an funktionale Korrektheit ab.

4.6. Erfüllung der Verständlichkeit

Zur Erfüllung dieses Qualitätsmerkmals können Usability Tests durchgeführt werden. Ein möglicher Test ist in den Anforderungen in Abschnitt 3.1.1. Dieser wurde der Anleitung entsprechend mit fünf Testpersonen durchgeführt. Alle Testpersonen konnten die Schritte der Anleitung korrekt befolgen. Damit ist die Anforderung an die Verständlichkeit erfüllt.

4.7. Erfüllung der Modifizierbarkeit

Wahl der Frameworks in Kombination mit Git und Bitbucket. TypeScript hat Javascript gegenüber den Vorteil, dass es beim Programmieren Typsicherheit gibt. So können Fehler in der Entwicklung bereits früh abgefangen werden. Git und Bitbucket erlauben es mehrere Entwickler gleichzeitig an verschiedenen Version der Software zu arbeiten.

5. Rückblick auf das Mapping

Anhand der Anforderungen an die Qualitätsmerkmale aus 3.1.1 lässt sich ableiten, inwiefern der momentane Entwicklungsstand diesen Anforderungen gerecht wird.

Mängel in funktionaler Vollständigkeit:

- Es fehlt die Implementierung um das Akzeptanzkriterium A-6 zu erfüllen

Mängel in Funktionale Korrektheit:

- Es fehlt die Testabdeckung für die Funktionen der User Stories 2 und 3

Diese Arbeit kann als Vorlage für die weitere Entwicklung und Evaluierung der Tuschi verwendet werden. Im Laufe der Entwicklung können weitere Anforderungen an bereits vorhandene Qualitätsmerkmale oder Anforderungen für andere noch nicht relevante Merkmale erstellt und bearbeitet werden, damit sie den Benutzeransprüchen genügen.

6. Literaturverzeichnis

7. Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe angefertigt und mich anderer als der in den beigefügten Verzeichnissen angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Der Durchführung einer elektronischen Plagiatsprüfung stimme ich hiermit zu. Die eingereichte elektronische Fassung der Arbeit entspricht der eingereichten schriftlichen Fassung exakt. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Ich bin mir bewusst, dass eine unwahre Erklärung rechtliche Folgen hat.

Ort, den _____

Vorname, Name

8. Sperrvermerk

Diese Arbeit enthält vertrauliche Daten der Firma Visus Health IT GmbH und ist daher einschließlich aller ihrer Teile urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verfassers/ bzw. der Visus Health IT GmbH unzulässig. Bei der elektronischen Prüfung der wissenschaftlichen Arbeit durch den Fachbereich Elektrotechnik und Informatik wird die Geheimhaltung gewahrt.