Implementationsdokumentation – Sistershift

Die Anwendung Sistershift ist eine Webbasierte Anwendung, bei der die Anwendungslogik auf Dienstgeber und Dienstnutzer verteilt ist. Die Anwendungslogik beinhaltet das faire Erstellen eines Dienstplans mit Berücksichtigung von Mitarbeiterwünschen, dem Tauschen von Schichten und das Ersatzfinden und Eintragen von Mitarbeitern bei etwaigen Abwesenheiten. In der folgenden Dokumentation wird die Implementierung dieser beschrieben. Zusätzlich wird auf Abweichungen zu geplanten Umsetzungen aus Meilenstein 2 eingegangen.

Implementierung Dienstgeber:

Die Anwendungslogik auf dem Dienstgeber beinhaltet das Erstellen eines Dienstplans für einen gesamten Monat. Dieser Dienstplan soll Mitarbeiterwünsche zur Einsatzplanung berücksichtigen. Zusätzlich sollen hier die möglichen Tauschpartner bei einem Schichttausch ermittelt und die Aktualisierungen der Daten nach einem erfolgreichen Tausch erfolgen. Ursprünglich (lt. Systemarchitektur) sollte das Tauschen von Schichten und die damit verbundene Anwendungslogik auf dem Dienstnutzer erfolgen. Die Ersatzplanung sollte dafür noch auf dem Dienstgeber stattfinden. Im aktuellen Stand des Systems, erfolgt die Ersatzplanung mit der damit verbundenen Anwendungslogik jedoch auf dem Dienstnutzer. Da beide Funktionalitäten eine ähnliche Anwendungslogik haben, ist der genannte Wechsel der Ausführungsortes nicht relevant. Die Anbindung an die My-SQL-Datenbank erfolgt weiterhin über den Dienstgeber.

Auf das Implementieren der Schichttauschfunktion und der Ermittlung potentieller Tauschpartner wurde auf Grund des zeitlich sehr beschränkten Projektzeitraums verzichtet. Das Hauptaugenmerk und auch die Kernfunktion der Anwendung ist das automatisierte Erstellen eines Dienstplans. Der Nutzer des Systems muss lediglich einen Monat, eine Station (als ID) und ein Jahr angeben und das System erstellt auf Basis dieser Eingaben einen Dienstplan für den gesamten Monat. Ein besonderes Alleinstellungsmerkmal bei der Dienstplanerstellung ist neben der Automatisierung das Beachten von Mitarbeiterwünschen. Mitarbeiter haben die Möglichkeit Wünsche zu kommenden Dienstplänen abzugeben. Das System prüft die Machbarkeit und löst etwaige Konflikte der Wünsche. Ist ein Tausch mit einem anderen Mitarbeiter möglich, so wird ein Wunsch gewährt und der Dienstplan entsprechend angepasst.

1) Erstellung Dienstplan unter Einbezug von Mitarbeiterwünschen

```
// Erstellen eines neuen Dienstplan
router.post('/', bodyParser.json(), (req, res) => {

  controller.erstelleDienstplan(req.body.monat, req.body.jahr, req.body.stationID)
    .then(function(dienstplan) {
      res.status(201).send(dienstplan)

    }).catch(function(err) { //
      res.status(400).send(err); //
    })

});
```

Durch einen POST – Request auf den Dienstgeber wird die Funktion “erstelleDienstplan” aufgerufen, in der die Anwendungslogik zu der Erstellung ausgelagert wurde.

Die Funktion erstelleDienstplan:

```
var tagZaehler = require("../helper/tagberechnung.js").getDaysInMonth;
var sqlHandler = require("../helper/sqlHandler.js");
var wuensche = require("../helper/wuensche.js");

// Anwendungslogik hinter einem POST auf Dienstplan
var erstelleDienstplan = function(monat, jahr, stationID) {

  return new Promise(function(resolve, reject) {

    var mitarbeiterListe;
    var anzahlSchichten = 3; // (Frühschicht, Spaetschicht, Nachtschicht)
    var anzahlMitarbeiterSchicht = 4;
    var fruehschicht = "Fruehschicht";
    var spaetschicht = "Spaetschicht";
    var nachtschicht = "Nachtschicht";

    // Die folgenden 3 Objekte werden mit Werten gefüllt und als Parameter für die SQL-Funktionen benötigt. Mit Hilfe dieser werden die Datensätze in der Datenbank angelegt.
    const dienstplan = {
      stationID: stationID,
      monat: monat,
      jahr: jahr,
      monatsTage: new Array()
    };

    var schichtzuweisung = {
      datum: "",
      schichtArt: "",
      mitarbeiterID1: "",
      mitarbeiterID2: "",
      mitarbeiterID3: "",
      mitarbeiterID4: ""
    }

    var tag = {
      schichtzuweisungID1: "",
      schichtzuweisungID2: "",
      schichtzuweisungID3: "",
      datum: ""
    }
```

Ein Dienstplan besteht aus 31 Tagen, einer StationID, dem Monat und einem Jahr. Ein Tag wiederum besteht aus 3 Schichten und dem jeweiligen Datum. Eine Schicht, bzw. Schichtzuweisung besteht aus dem Datum, der Schichtart (Früh-, Spät-, oder Nachtschicht) und den ID´s der Mitarbeiter, welche in dieser Schicht eingeteilt sind.

Diese Verschachtelung muss bei der Erstellung eines Dienstplans beachtet werden.

Zunächst werden Mitarbeiter einer Schicht zugeordnet. Da die Mitarbeiter mal in allen Schichten und nicht immer nach demselben Dienstplanmuster arbeiten sollen, wurden verschiedene Zyklen implementiert, nach denen die Mitarbeiterbesetzung rotiert.

```
var mitarbeiterZuweisung;

// Zyklen für Januar,April,Juli,Oktober
var zyklus1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11];
var zyklus2 = [12, 13, 14, 15, 16, 17, 2, 3, 4, 5, 0, 1];
var zyklus3 = [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17];


// Zyklen für Februar,Mai,August,November
var zyklus4 = [12, 13, 14, 15, 16, 17, 2, 3, 4, 5, 0, 1];
var zyklus5 = [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17];
var zyklus6 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11];


// Zyklen für März,Juni,September,Dezember
var zyklus7 = [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17];
var zyklus8 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11];
var zyklus9 = [12, 13, 14, 15, 16, 17, 2, 3, 4, 5, 0, 1];
```

Sind die Mitarbeiter auf die verschiedenen Zyklen aufgeteilt, so werden die Zyklen für eine variable Verteilung der Schichten auf die Tage eines Monats verwendet.

```
// Variable Verteilung der Schichten auf die Tage
for (let i = 1; i <= anzahlTage; i++) {

  if (i < 3 || (i >= 7 && i < 9) || (i >= 13 && i < 15) || (i >= 19 && i < 21) || (i >= 25 && i < 27)) {
    if (monat == 1 || monat == 4 || monat == 7 || monat == 10) {
      mitarbeiterZuweisung = zyklus1;
    } else if (monat == 2 || monat == 5 || monat == 8 || monat == 11) {
      mitarbeiterZuweisung = zyklus4;
    } else if (monat == 3 || monat == 6 || monat == 9 || monat == 12) {
      mitarbeiterZuweisung = zyklus7;
    }

  } else if (i < 5 || (i >= 9 && i < 11) || (i >= 15 && i < 17) || (i >= 21 && i < 23) || (i >= 27 && i < 29)) {
    if (monat == 1 || monat == 4 || monat == 7 || monat == 10) {
      mitarbeiterZuweisung = zyklus2;
    } else if (monat == 2 || monat == 5 || monat == 8 || monat == 11) {
      mitarbeiterZuweisung = zyklus5;
    } else if (monat == 3 || monat == 6 || monat == 9 || monat == 12) {
      mitarbeiterZuweisung = zyklus8;
    }
  } else if (i < 7 || (i >= 11 && i < 13) || (i >= 17 && i < 19) || (i >= 23 && i < 25) || (i >= 29 && i <= 31)) {
    if (monat == 1 || monat == 4 || monat == 7 || monat == 10) {
      mitarbeiterZuweisung = zyklus3;
    } else if (monat == 2 || monat == 5 || monat == 8 || monat == 11) {
      mitarbeiterZuweisung = zyklus6;
    } else if (monat == 3 || monat == 6 || monat == 9 || monat == 12) {
      mitarbeiterZuweisung = zyklus9;
    }
  }
```

Nach der Verteilung auf die Tage, werden anschließend die 3 verschiedenen Schichten nach demselben Muster in der Datenbank erstellt. Sind diese erstellt, so werden diese direkt einem Tag zugeordnet.

```
// Erstellung von Schichtzuweisungen, die einem Tag zugewiesen werden
for (let j = 0; j < anzahlSchichten; j++) {

  if (j == 0) {
    let datum = i + "-" + monat + "-" + jahr; //Formatieren des Datums für die DB-Speicherung
    if (i < 10) {
      datum = "0" + i + "-" + monat + "-" + jahr;
    }

    // Erstellen einer Schichtzuweisung "Frühschicht" mit Mitarbeitern und datum x
    schichtzuweisung.datum = datum;
    schichtzuweisung.schichtArt = "Fruehschicht";
    schichtzuweisung.mitarbeiterID1 = mitarbeiterListe[mitarbeiterZuweisung[0]].id;
    schichtzuweisung.mitarbeiterID2 = mitarbeiterListe[mitarbeiterZuweisung[1]].id;
    schichtzuweisung.mitarbeiterID3 = mitarbeiterListe[mitarbeiterZuweisung[2]].id;
    schichtzuweisung.mitarbeiterID4 = mitarbeiterListe[mitarbeiterZuweisung[3]].id;
    sqlHandler.neueSchichtzuweisung(schichtzuweisung)
      .then(function(schichtzuweisung) {
        if (schichtzuweisung === undefined) console.log("Schichtzuweisung konnte nicht erstellt werden");
      })
      .catch(function(err) {
        console.log(err);
      }).then(function() {
        sqlHandler.getSchichtzuweisung(datum, fruehschicht)
          .then(function(schicht) {
            tag.schichtzuweisungID1 = schicht[0].id;

          })
      })
  }
```

Sobald alle Schichten aufgeteilt sind, werden die Tage anschließend auch durch den entsprechenden SQL-Befehl in der Datenbank angelegt. Sofern die Tage und Schichtzuweisungen angelegt sind, wird der Dienstplan in der Datenbank angelegt.

```
sqlHandler.neuerDienstplan(dienstplan)
  .then(function(dienstplanDB) {
    if (dienstplanDB === undefined) reject("Dienstplan konnte nicht erstellt werden");
    else {

      // Nach Erstellung eines Dienstplans, wird der Monat auf Mitarbeiterwünsche kontrolliert.
      wuensche.korrigiereSchichtzuweisungen(dienstplan).then(function(finalerDienstplan) {
        if (finalerDienstplan == -1) {
          // Falls keine Wünsche vorhanden sind, wird der normale Dienstplan resolvt
          sqlHandler.getDienstplanByDate(dienstplan.monat, dienstplan.jahr)
            .then(function(dienstplanOhneWuensche) {
              resolve(dienstplanOhneWuensche);
            })

        } else {
          resolve(finalerDienstplan) // Dienstplan mit gewährten Wünschen
        }
```

Nach dem erfolgreichen Anlegen des unbearbeiteten Dienstplans erfolgt die Einbeziehung der Mitarbeiterwünsche, sofern zu dem Monat, für den der Dienstplan erstellt wird welche vorliegen.

Dies geschieht mit der Funktion “korrigiereSchichtzuweisungen”. Die Anwendungslogik für diese Einbeziehung ist wiederum ausgelagert.

Bei der genannten Funktion werden zunächst alle Wünsche, die betroffen sind ermittelt. Sobald Wünsche vorliegen, werden konfliktäre Wünsche nach dem Wunschrating eines Mitarbeiters gelöst und eine Liste mit zu erfüllenden Wünschen erstellt. Das genannte Wunschrating wird nach Gewähren oder Verweigern eines Wunsches beim jeweiligen Mitarbeiter angepasst.

```
// Lesen aller Wünsche einer Station für den Monat x
sqlHandler.getWuenscheStation(dienstplan.stationID, monat)
  .then(function(wunschListe) {
    if (wunschListe.length == 0) {

      console.log("Keine Wuensche auf dieser Station vorhanden!");
      resolve(-1);
    } else {
      mitarbeiterWuenscheListe = wunschListe;

      var wunschSuche = new Promise(function(resolve, reject) {

        // Filtern von konfliktären Wünschen (wenn selbes Datum!) anhand des Wunschratings eines Mitarbeiters
        for (let j = 0; j < mitarbeiterWuenscheListe.length; j++) {
          for (let z = j + 1; z < mitarbeiterWuenscheListe.length; z++) {

            // Datum-Abgleich der Wünsche
            if (mitarbeiterWuenscheListe[j].datumWunsch == mitarbeiterWuenscheListe[z].datumWunsch) {
              sqlHandler.getMitarbeiterById(mitarbeiterWuenscheListe[j].mitarbeiterID)
                .then(function(mitarbeiterJ) {
                  sqlHandler.getMitarbeiterById(mitarbeiterWuenscheListe[z].mitarbeiterID)
                    .then(function(mitarbeiterZ) {

                      // Bei selben Datum -> Abgleich Wunschrating der verschiedenen Mitarbeiter -> Mitarbeiter dem der Wunsch gewährt wird Anpassung des Wunschratings um -1 ansonsten +1
                      if (mitarbeiterJ[0].wunschRating >= mitarbeiterZ[0].wunschRating) {

                        mitarbeiterWuenscheListe.splice(z, 1);
                        sqlHandler.updateWunschRating(mitarbeiterJ[0].id, -1);
                        sqlHandler.updateWunschRating(mitarbeiterZ[0].id, 1);

                      } else {

                        mitarbeiterWuenscheListe.splice(j, 1);
                        sqlHandler.updateWunschRating(mitarbeiterJ[0].id, 1);
                        sqlHandler.updateWunschRating(mitarbeiterZ[0].id, -1);

                      }
```

Sobald die Wunschliste zu einem Dienstplan vorliegt, werden die Mitarbeiter, welchen ein Wunsch gewährt wird aus der jeweiligen Schicht am Wunschdatum ausgetragen. Nach dem Austragen erfolgt die Ermittlung eines Mitarbeiters, welcher seine Schicht mit der nun unterbesetzten Schicht tauscht. Dies geschieht in der Funktion “tauscheSchicht”.

Bei dieser werden zunächst alle Schichten derselben Schichtart gefiltert. Danach werden die Kollegen derselben Schicht und die ID des Mitarbeiters mit dem Wunsch selbst als potentielle Tauschpartner gefiltert.

```
// Filtern nach allen Schichten der selben Schichtart
for (let i = 0; i < schichten.length; i++) {
  if (schichten[i].schichtArt == schichtzuweisung.schichtArt)
    passendeSchichten.push(schichten[i]);
}

// Filtern der Tauschpartner von Kolllegen der selben Schicht und ID des Wunschgebers selbst (Mitarbeiter, welche in der Schicht arbeitern, welche sich frei gewünscht wu
for (let j = 0; j < passendeSchichten.length; j++) {
  if (passendeSchichten[j].mitarbeiterID1 != schichtzuweisung.mitarbeiterID1 && passendeSchichten[j].mitarbeiterID1 != schichtzuweisung.mitarbeiterID2 && passendeSchicht

    if (passendeSchichten[j].mitarbeiterID1 != mitarbeiterID) {
      var passendeMitarbeiterSchicht1 = new passendeMitarbeiterSchicht(passendeSchichten[j].mitarbeiterID1, passendeSchichten[j].datum, 1, schichtzuweisung.schichtArt);

      passendeMitarbeiter.push(passendeMitarbeiterSchicht1);
    }
  }
  if (passendeSchichten[j].mitarbeiterID2 != schichtzuweisung.mitarbeiterID1 && passendeSchichten[j].mitarbeiterID2 != schichtzuweisung.mitarbeiterID2 && passendeSchicht

    if (passendeSchichten[j].mitarbeiterID2 != mitarbeiterID) {
      var passendeMitarbeiterSchicht2 = new passendeMitarbeiterSchicht(passendeSchichten[j].mitarbeiterID2, passendeSchichten[j].datum, 2, schichtzuweisung.schichtArt);

      passendeMitarbeiter.push(passendeMitarbeiterSchicht2);
    }
  }
  if (passendeSchichten[j].mitarbeiterID3 != schichtzuweisung.mitarbeiterID1 && passendeSchichten[j].mitarbeiterID3 != schichtzuweisung.mitarbeiterID2 && passendeSchicht
    if (passendeSchichten[j].mitarbeiterID3 != mitarbeiterID) {
      var passendeMitarbeiterSchicht3 = new passendeMitarbeiterSchicht(passendeSchichten[j].mitarbeiterID3, passendeSchichten[j].datum, 3, schichtzuweisung.schichtArt);

      passendeMitarbeiter.push(passendeMitarbeiterSchicht3);
    }
  }
  if (passendeSchichten[j].mitarbeiterID4 != schichtzuweisung.mitarbeiterID1 && passendeSchichten[j].mitarbeiterID4 != schichtzuweisung.mitarbeiterID2 && passendeSchicht
    if (passendeSchichten[j].mitarbeiterID4 != mitarbeiterID) {
      var passendeMitarbeiterSchicht4 = new passendeMitarbeiterSchicht(passendeSchichten[j].mitarbeiterID4, passendeSchichten[j].datum, 4, schichtzuweisung.schichtArt);

      passendeMitarbeiter.push(passendeMitarbeiterSchicht4);
    }
```

Sofern die ersten beiden Filterungen erfolgreich waren, folgt die Filterung der Mitarbeiter, welche am selben Tag des Wunschdatums arbeiten und somit als Tauschpartner nicht in Frage kommen.

```
// Rausfiltern der am selben Tag tätigen Mitarbeiter
var schichtenAmTagX = new Array();
for (let x = 0; x < schichten.length; x++) {
  if (schichtzuweisung.datum == schichten[x].datum) {

    schichtenAmTagX.push(schichten[x]);

  }
  if (x + 1 == schichten.length) {

    var unpassendeMitarbeiterIDs = [
      schichtenAmTagX[0].mitarbeiterID1, schichtenAmTagX[0].mitarbeiterID2, schichtenAmTagX[0].mitarbeiterID3, schichtenAmTagX[0].mitarbeiterID4,
      schichtenAmTagX[1].mitarbeiterID1, schichtenAmTagX[1].mitarbeiterID2, schichtenAmTagX[1].mitarbeiterID3, schichtenAmTagX[1].mitarbeiterID4,
      schichtenAmTagX[2].mitarbeiterID1, schichtenAmTagX[2].mitarbeiterID2, schichtenAmTagX[2].mitarbeiterID3, schichtenAmTagX[2].mitarbeiterID4
    ]

    passendeMitarbeiter.forEach(function(mitarbeiter) {

      for (let y = 0; y < unpassendeMitarbeiterIDs.length; y++) {

        if (mitarbeiter.mitarbeiterID != unpassendeMitarbeiterIDs[y]) {

          passendeMitarbeiter.splice(passendeMitarbeiter.indexOf(mitarbeiter), 1);
        }

      }
    })

  }
}
```

Nach dieser Filterung erfolgt eine zufällige Auswahl eines verfügbaren Tauschpartners.

```
          // Zufällige Auswahl eines in Frage kommenden Mitarbeiters
          var tauschenderMitarbeiter = passendeMitarbeiter[getRandomInt(passendeMitarbeiter.length)];

          var resolveObject = {
            tauschenderMitarbeiter: tauschenderMitarbeiter,
            elements: schichten
          }

          resolve(resolveObject);
        }

      }

    })

    tausch.then(function(resolveObject) {
        // Aktualisieren der jeweiligen Schichten -> Tausch der Mitarbeiter
        sqlHandler.updateSchichtzuweisungWunsch(mitarbeiterID, resolveObject.tauschenderMitarbeiter);
```

Ist ein Mitarbeiter gefunden, der für den ausgetragenen Mitarbeiter einspringen kann, wird dieser in die Schicht am Wunschdatum eingesetzt. Der Mitarbeiter übernimmt dann die ursprüngliche Schicht des einspringenden Kollegen.

```
// 1)Folgend wird Mitarbeiter mit Wunsch ausgetragen
// 2)Ermittlung eines Mitarbeiters, welche Schicht tauscht, in der Funktion tauscheSchicht ()
// 3)Eintragen der ID´s der jeweiligen Mitarbeiter in den entsprechenden Schichten
//*****************************************************************************************************************************************************

for (let j = 0; j < mitarbeiterWuenscheListe.length; j++) {
  for (let i = 0; i < elements.length; i++) {

    // 1)
    if ((mitarbeiterWuenscheListe[j].datumWunsch == elements[i].datum) && (mitarbeiterWuenscheListe[j].mitarbeiterID == elements[i].mitarbeiterID1)) {
      let schichtzuweisungUpdate = new schichtzuweisungUpdateObject(-1, elements[i].mitarbeiterID2, elements[i].mitarbeiterID3, elements[i].mitarbeiterID4)

      // 2)
      tauscheSchicht(elements[i], elements, mitarbeiterWuenscheListe[j].mitarbeiterID).then(function(resolveObject) {
        return new Promise(function(resolve, reject) {

          resolve(resolveObject);

        }).then(function(resolveObject) {

          // 3)
          schichtzuweisungUpdate.mitarbeiterID1 = resolveObject.tauschId;
          sqlHandler.updateSchichtzuweisung(resolveObject.elements[i].datum, resolveObject.elements[i].schichtArt, schichtzuweisungUpdate)
            .then(function(schichtzuweisung) {
              if (schichtzuweisung === undefined) console.log("Schichtzuweisung konnte nicht aktualisiert werden");
            })
            .catch(function(err) {
              console.log(err);
            })

        })

      })
```

Nach dem Aktualisieren der Schichten, wird der finale Dienstplan zurückgegeben.

Für das Rendern eines Dienstplans werden alle erforderlichen Informationen der Schichten Dienstplan, Tage und Schichten benötigt.

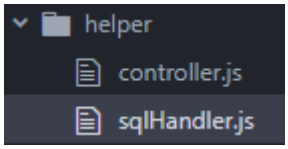
Diese Informationen können über einen GET-Request abgefragt werden. Das Ergebnis dieses ist ein Objekt, welches in Metadaten (Dienstplan Informationen, wie StationID, Monat und Jahr), Tage und Schichtzuweisungen untereilt ist.

Der Dienstgeber bietet neben den beiden im Detail beschriebenen Funktionen noch folgende weiter Funktionen:

- Löschen eines Dienstplans
- Abrufen aller Abwesenheiten
- Abrufen einer bestimmten Abwesenheit
- Erstellen einer neuen Abwesenheit
- Aktualisieren einer Abwesenheit
- Löschen einer Abwesenheit
- Erstellen von Ersatzanfragen
- Abrufen der Ersatzanfragen eines Mitarbeiters
- Löschen einer Ersatzanfrage
- Löschen einer Ersatzanfrage für einen bestimmten Mitarbeiter
- Erstellen von Ersatzeintragungen
- Abrufen der Ersatzeintragungen eines Mitarbeiters
- Abrufen aller Mitarbeiter
- Abrufen eines Mitarbeiters
- Aktualisieren eines Mitarbeiters
- Erstellen eines Mitarbeiters
- Löschen eines Mitarbeiters
- Abrufen des Wunschratings eines Mitarbeiters
- Aktualisieren des Wunschratings
- Abrufen der Überstunden eines Mitarbeiters
- Aktualisieren der Überstunden eines Mitarbeiters
- Erstellen eines Mitarbeiterwunsches (zur Dienstplanung)

- Abrufen aller Tauschanfragen
- Abrufen einer bestimmten Tauschanfrage
- Erstellen einer neuen Tauschanfrage
- Aktualisieren einer Tauschanfrage
- Löschen einer Tauschanfrage

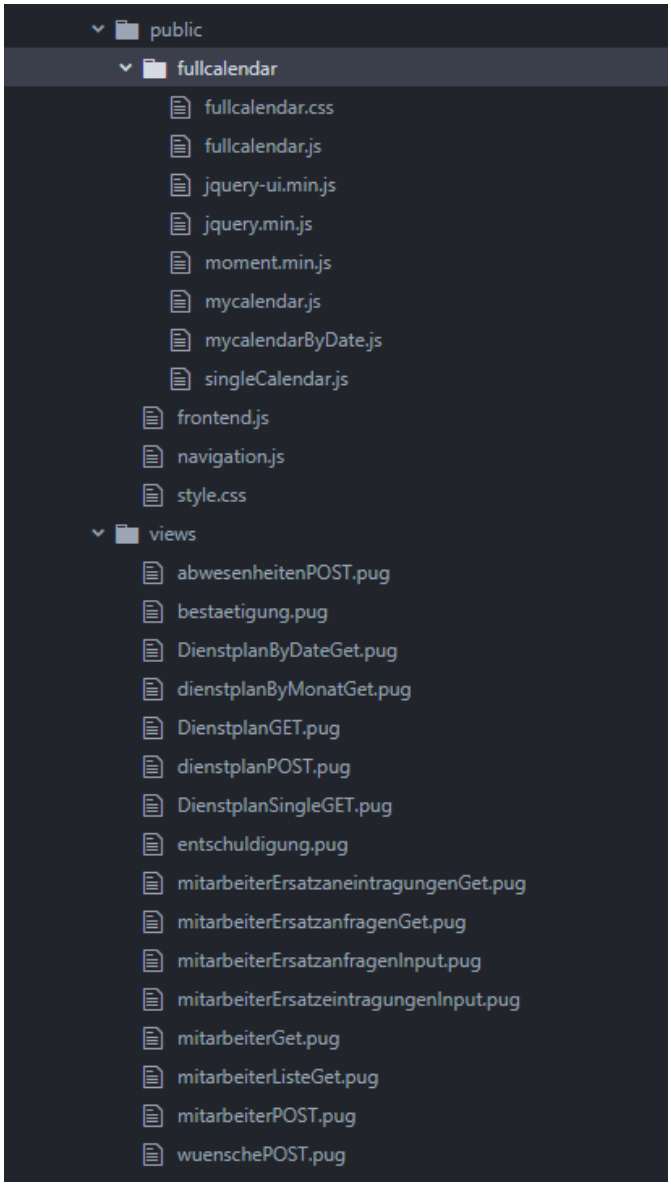
Alle Datenbankzugriffe wurden in extra Funktionen ausgelagert.



Zusätzlich arbeiten alle Funktionen mit Statuscodes. Genauere Infos sind in der REST-Modellierung zu finden.

Implementierung Dienstnutzer:

Der Dienstnutzer bildet die Schnittstelle zwischen dem User und dem System.
Auf diesem wird die gesamte Oberfläche des Systems gerendert. Dies geschieht mit Hilfe des pug-modules. Zusätzlich wird auf diesem das Frontend-JavaScript ausgeführt.



Das Rendern erfolgt dynamisch während der Laufzeit. Folgend ist das Rendern des Dienstplans zu sehen. Mit Hilfe des pug-Moduls wird folgende pug-Datei nach den User-Eingaben gerendert.

```
1  doctype html
2  html
3    head
4      title Dienstplan
5      link(rel='stylesheet', href='/static/fullcalendar/fullcalendar.css')
6      script(src='/static/fullcalendar/jquery.min.js')
7      script(src='/static/fullcalendar/moment.min.js')
8      script(src='/static/fullcalendar/fullcalendar.js')
9      script(src='/static/fullcalendar/mycalendar.js')
10   body
11     #calendar
12
```

```
// Darauf warten das das document geladen ist.
$(document).ready(function() {
  var url = "http://sistershift.ddns.net/Dienstplaene/" + getIndexVonDienstplan();
  // Dienstplan Informationen von Dienstgeber holen
  $.get(url, function(dienstplan) {
    var schichten;
    var arrayOfSchichten = new Array();
    // Daten aufbereiten -> da im Response eine nested Array steht Arry[[],[],[],[]]
    for (let i = 0; i < dienstplan.schichten.length; i++) {

      schichten = {
        art: dienstplan.schichten[i]
      };
      arrayOfSchichten.push(schichten)
    }

    var einzelSchichten = new Array();

    arrayOfSchichten.forEach(function(element) {

      //Fruehschicht
      var schicht1 = new schicht(element.art[0].schichtArt + " " + element.art[0].mitarbeiterID1 + " " + element.art[0].mitarbeiterID2 +

      //Spaetschicht
      var schicht2 = new schicht(element.art[1].schichtArt + " " + element.art[1].mitarbeiterID1 + " " + element.art[1].mitarbeiterID2 +

      //Nachtschicht
      var schicht3 = new schicht(element.art[2].schichtArt + " " + element.art[2].mitarbeiterID1 + " " + element.art[2].mitarbeiterID2 +
      einzelSchichten.push({
        schicht1,
        schicht2,
        schicht3
      })

    })
```

```
    var events = new Array();
    console.log(einzelSchichten)
    einzelSchichten.forEach(function(element) {

      events.push(element.schicht1, element.schicht2, element.schicht3)

    })

    // page is now ready, initialize the calendar..
    $('#calendar').fullCalendar({

      eventSources: [

        // your event source
        {
          // das events Array übergeben und die enthaltenen Informationen rendern
          events: events
          // an option!
        }

        // any other event sources...

      ]

    });
    // Springe zu dem Monat in dem Der Dienstplan beginnt.
    $('#calendar').fullCalendar('gotoDate', einzelSchichten[0].schicht1.start)


  });
})
```

Hierbei werden mit Hilfe von Jquery die erforderlichen Dienstplaninformationen beschafft. Nach der Beschaffung werden diese aufbereitet, und mit Hilfe des “fullcalender”-templates einen Dienstplan im Kalenderformat gerendert. Es wird die ID #calender verwendet um den gerenderten Dienstplan richtig zu positionieren.

Die Anwendungslogik auf dem Dienstnutzer beinhaltet das Ermitteln eines Mitarbeiters, welcher als Ersatz bei der Einreichung einer Abwesenheit in Frage kommt.

```
// POST auf eine Abwesenheit -> Damit verbunden: Ermittlung und Erstellung entsprechender Ersatzanfragen an die Mitarbeiter
router.post('/abwesenheiten', (req, res) => {

  var abwesenheitsErstellung = new Promise(function(resolve, reject) {

    const abwesenheit = {
      stationID: req.body.stationID,
      mitarbeiterID: req.body.MitarbeiterID,
      datumBeginn: req.body.datumBeginn,
      datumEnde: req.body.datumEnde
    };

    // POST Request an Dienstgeber -> Anlegen einer Abwesenheit in der Datenbank
    let resourceURI = serviceURL + '/Abwesenheiten';

    console.log(resourceURI);

    var options = {
      uri: resourceURI,
      method: 'POST',
      headers: {
        'Accept': 'application/json'
      },
      json: abwesenheit
    }

    request(options, (err, res, body) => {

      if (err) {
        console.log(err);
        return;
      } else {
        var abwx2 = {
          abwesenheitDB: body[0],
          abwesenheit: abwesenheit
        }
        resolve(abwx2);
      }

    });


  }) // end of Promise abwesenheitsErstellung
```

Die für die Abwesenheit-POST-Request erforderlichen Daten, werden vom Frontend-JavaScript geliefert.

Reicht ein Mitarbeiter eine neue Abwesenheit ein, so ermittelt die Anwendungslogik des Dienstnutzers alle Mitarbeiter, welche als Ersatz für die ersatzbedürftige Schicht in Frage kommen. Dabei wird auf die Einhaltung der gesetzlichen Rahmenbedingungen geachtet.

```
// Auslagerung der Anwendungslogik des Dienstnutzers

var request = require('request');


// Funktion zur Ermittlung der Mitarbeiter, welche eine Schicht übernehmen können, die durch eine Abwesenheit unterbesetzt wäre
var ersatzAnfrage = function ersatzAnfrage(informationen, abwesenheit) {

  return new Promise(function(resolve, reject) {

    var ersatzBeduerftigeSchichtenErmittlung = new Promise(function(resolve, reject) {
      // nested Array auflösen
      var elements = new Array();

      for (var p = 0; p < informationen.dienstplan.schichten.length; p++) {
        informationen.dienstplan.schichten[p].forEach(function(element) {
          elements.push(element)
        })

        if (p + 1 == informationen.dienstplan.schichten.length) {
          var substringDatum;
          var ersatzBeduerftigeSchichtInfos = {
            alleSchichten: elements,
            mitarbeiter: informationen.mitarbeiterListe,
            schichten: [],
            abwesenderID: abwesenheit.mitarbeiterID
          }

          // Filtern der Schichten, welche Ersatz benötigen:
          for (let i = 0; i < elements.length; i++) {
            if (elements[i].datum == abwesenheit.datumBeginn) {
              var tmp = i;
              while (elements[i].datum != abwesenheit.datumEnde) {
                ersatzBeduerftigeSchichtInfos.schichten.push(elements[i]);
                i++;

                if (i == elements.length) {
                  i = tmp;
                  break;
                }

              }
            }

            if (i + 1 == elements.length) {
              resolve(ersatzBeduerftigeSchichtInfos);
            }
          } // for i - Schleife
```

```
    }
  }
}) // End of Promise ersatzBeduerftigeSchichtenErmittlung

// Ermittlung Mitarbeiter, welche an diesen Tagen frei haben:
ersatzBeduerftigeSchichtenErmittlung.then(function(ersatzBeduerftigeSchichtInfos) {

  var ermittlungVerfuegbarerMitarbeiter = new Promise(function(resolve, reject) {

    var verfuegbareMitarbeiter = new Array();
    var alleMitarbeiter = ersatzBeduerftigeSchichtInfos.mitarbeiter;

    var erforderlicheInfos = {
      alleSchichten: ersatzBeduerftigeSchichtInfos.alleSchichten,
      schichten: [],
      abwesenderID: ersatzBeduerftigeSchichtInfos.abwesenderID
    }

    for (let i = 0; i < ersatzBeduerftigeSchichtInfos.schichten.length; i++) {
      for (let j = 0; j < alleMitarbeiter.length; j++) {

        if (alleMitarbeiter[j].id != ersatzBeduerftigeSchichtInfos.schichten[i].mitarbeiterID1 && alleMitarbei
          verfuegbareMitarbeiter.push(alleMitarbeiter[j]);
```

```
}
if (j + 1 == alleMitarbeiter.length) {
  erforderlicheInfos.schichten.push(new schicht(ersatzBeduerftigeSchichtInfos.schichten[i].schichtArt, ersatzBeduerftigeSchichtInfos.schichten[i].datum, verfuegbareMitarbeiter));
  verfuegbareMitarbeiter = [];
}
```

```
      } // end of j-Schleife
      if (i + 1 == ersatzBeduerftigeSchichtInfos.schichten.length) {


        resolve(erforderlicheInfos);
      }
    } // end of i- Schleife

  }) // end of Promise ermittlungNichtVerfuegbarerMitarbeiter

  ermittlungVerfuegbarerMitarbeiter.then(function(erforderlicheInfos) {

    // Filtern der verfuegbaren Mitarbeitern nach den gesetzlichen Vorgaben:
    var filternGesetzlicheBedingungen = new Promise(function(resolve, reject) {

      for (let i = 0; i < erforderlicheInfos.schichten.length; i++) {
        var vars = erforderlicheInfos.schichten[i].datum.split("-");
        var tagVorher = parseInt(vars[0]) - 1;
        var tagNachher = parseInt(vars[0]) + 1;
        var monat = vars[1];
        var jahr = vars[2];
        if (tagVorher < 10) {
          tagVorher = "0" + tagVorher;
        }
        if (tagNachher < 10) {
          tagNachher = "0" + tagNachher;
        }
        var datumVorher = tagVorher + "-" + monat + "-" + jahr;
        var datumNachher = tagNachher + "-" + monat + "-" + jahr;

        // Abgleich mit Schichten davor:
        if (erforderlicheInfos.alleSchichten[i].datum == datumVorher) {
          for (let z = 0; z < erforderlicheInfos.schichten.length; z++) {
            for (let y = 0; y < erforderlicheInfos.schichten[z].mitarbeiter.length; y++) {

              if (erforderlicheInfos.schichten[z].mitarbeiter[y].id == erforderlicheInfos.alleSchichten[i].mitarbeiterID1 || er

                // Vor einer Fruehschicht nur Fruehschicht
                if (erforderlicheInfos.schichten[z].schichtArt == "Fruehschicht") {
                  if (erforderlicheInfos.alleSchichten[i].schichtArt != "Fruehschicht") {
                    erforderlicheInfos.schichten[z].mitarbeiter.splice(y, 1);
                  }
```

```
                }
              }
              // Vor Spaetschicht darf nur Frueh oder Spaet gearbeitet werden
              if (erforderlicheInfos.schichten[z].schichtArt == "Spaetschicht") {
                if (erforderlicheInfos.alleSchichten[i].schichtArt == "Nachtschicht") {
                  erforderlicheInfos.schichten[z].mitarbeiter.splice(y, 1);
                }
              }
              // Vor einer Nachtschicht darf Frueh oder Spaet gearbeitet werden
              if (erforderlicheInfos.schichten[z].schichtArt == "Nachtschicht") {
                if (erforderlicheInfos.alleSchichten[i].schichtArt == "Nachtschicht") {
                  erforderlicheInfos.schichten[z].mitarbeiter.splice(y, 1);
                }
              }

            }
          }
        }
      }

      // Abgleich mit Schichten danach:
      if (erforderlicheInfos.alleSchichten[i].datum == datumNachher) {
        for (let z = 0; z < erforderlicheInfos.schichten.length; z++) {
          for (let y = 0; y < erforderlicheInfos.schichten[z].mitarbeiter.length; y++) {

            if (erforderlicheInfos.schichten[z].mitarbeiter[y].id == erforderlicheInfos.alleSchichten[i].mitarbeiterID1 || er

              // Nach einer Spaetschicht darf nur Spaet oder Nacht gearbeitet werden
              if (erforderlicheInfos.schichten[z].schichtArt == "Spaetschicht") {
                if (erforderlicheInfos.alleSchichten[i].schichtArt == "Fruehschicht") {
                  erforderlicheInfos.schichten[z].mitarbeiter.splice(y, 1);
                }
              }
              // Nach einer Nachtschicht darf nur oder Nacht gearbeitet werden
              if (erforderlicheInfos.schichten[z].schichtArt == "Nachtschicht") {
                if (erforderlicheInfos.alleSchichten[i].schichtArt != "Nachtschicht") {
                  erforderlicheInfos.schichten[z].mitarbeiter.splice(y, 1);
                }
              }

            }
          }
        }
      }

      if (i + 1 == erforderlicheInfos.schichten.length) {
        resolve(erforderlicheInfos);
      }
    } // ende i - Schleife
```

```
}) // end of Promise

filternGesetzlicheBedingungen.then(function(erforderlicheInfos) {

  // Ermittelte Mitarbeiter pro Schicht zusammenfassen -> Verfuegbare Mitarbeiter pro Tag -> Wenn 3 mal als verfuegbar ermittelt = Für den gesamten Tag verfuegbar
  var alleFreienMitarbeiter = new Array();
  for (let i = 0; i < erforderlicheInfos.schichten.length; i++) {
    for (let j = 0; j < erforderlicheInfos.schichten[i].mitarbeiter.length; j++) {
      alleFreienMitarbeiter.push(erforderlicheInfos.schichten[i].mitarbeiter[j])

    }
    if (i + 1 == erforderlicheInfos.schichten.length) {
      var ersatzMitarbeiter = new Array();

      alleFreienMitarbeiter.sort((function(a, b) {
        return a.id - b.id
      }))
      for (let x = 0; x < alleFreienMitarbeiter.length; x++) {
        if (x < alleFreienMitarbeiter.length - 2) {
          if (alleFreienMitarbeiter[x].id == alleFreienMitarbeiter[x + 1].id && alleFreienMitarbeiter[x].id == alleFreienMitarbeiter[x + 2].id) {
            ersatzMitarbeiter.push(alleFreienMitarbeiter[x])
          }
        }
        if (x + 1 == alleFreienMitarbeiter.length) {
          for (let z = 0; z < erforderlicheInfos.schichten.length; z++) {
            erforderlicheInfos.schichten[z].mitarbeiter = ersatzMitarbeiter;
            if (z + 1 == erforderlicheInfos.schichten.length) {
              resolve(erforderlicheInfos.schichten); // Array von Schicht-Objekten mit Mitarbeitern, welche als Ersatz in Frage kommen (Konstruktor "schicht" (siehe unten))
            }
          }
        }

      }

    }

  }
```

```
// Konstruktor Schicht
function schicht(schichtArt, datum, mitarbeiter) {
  this.schichtArt = schichtArt;
  this.datum = datum;
  this.mitarbeiter = mitarbeiter; // Mitarbeiter, die die genannte Schicht, am genannten Datum übernehmen können
}
```

Das Ergebnis der Ermittlung ist ein Objekt, welches die Schichtart, das Datum der Schicht und ein Array mit den Mitarbeitern, welche die Schicht übernehmen könnten, enthält.

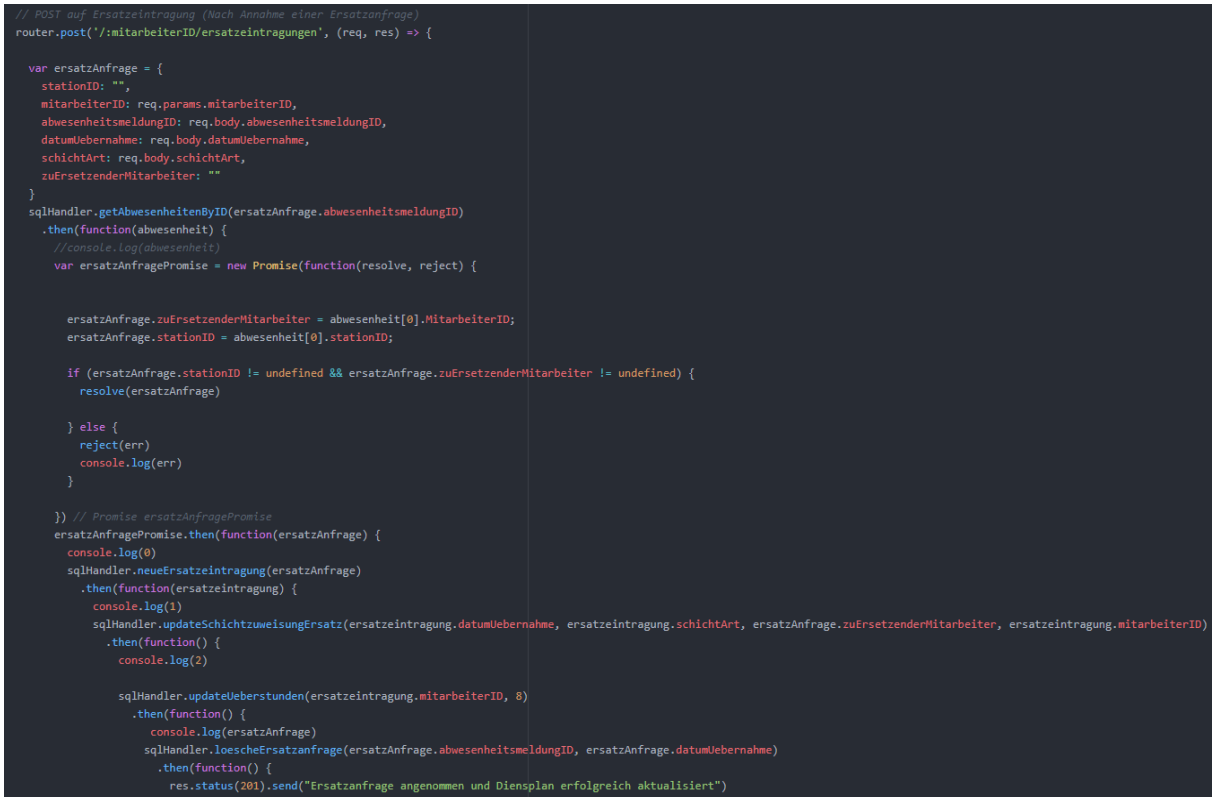
Nach der erfolgreichen Ermittlung werden Ersatzanfragen für diese in Frage kommenden Mitarbeiter generiert. Dies geschieht über folgenden POST-Request an den Dienstgeber.

```
// Dienstgeber Hostadresse
var serviceURL = 'http://sistershift.ddns.net';

// Funktion, welche die Erstellung nach der Ermittlung entsprechenden Ersatzanfragen beim Dienstgeber anstößt
var erstelleErsatzanfragen = function erstelleErsatzanfragen(ersatzAnfrageInfos, abwesenheit) {

  var json = {
    ersatzAnfrageInfos: ersatzAnfrageInfos,
    abwesenheit: abwesenheit
  }

  let resourceURI = serviceURL + '/Abwesenheiten/Ersatzanfragen';

  console.log(resourceURI);

  var options = {
    uri: resourceURI,
    method: 'POST',
    headers: {
      'Accept': 'application/json'
    },
    json: json
  }

  request(options, (err, res, body) => {

    if (err) {
      console.log(err);
      return;
    }
    if (res.status == 201) {
      console.log(body);
    }

  });

} // end of function


exports.erstelleErsatzanfragen = erstelleErsatzanfragen;
exports.ersatzAnfrage = ersatzAnfrage;
```

Dieser gesamte Vorgang ist aus dem Abwesenheit-POST-Request ausgelagert. Die Funktionen werden aber in diesem aufgerufen.

```
      // Funktionsaufrufe löst Kette von Ereignissen aus -> Ziel der Funktionen ist es Ersatzanfragen, für die in Frage kommenden Mitarbeiter zu erstellen

      ersatzAnfrage.ersatzAnfrage(informationen, abwx2.abwesenheit).then(function(ersatzAnfrageInfos) {
        ersatzAnfrage.erstelleErsatzanfragen(ersatzAnfrageInfos, abwx2.abwesenheitDB);

      }).catch(function(error) {
        console.log(error);
      });


    }).catch(function(error) {
      console.log(error);
    });

  }).catch(function(error) {
    console.log(error);
  });

}).catch(function(error) {
  console.log(error);
});

res.status(201).send("Abwesenheit eingereicht!");
```

Mitarbeiter können ihre Ersatzanfragen jeder Zeit abrufen. Jede Ersatzanfrage kann verbindlich angenommen oder abgelehnt werden. Nach dem Annehmen einer Ersatzanfrage, wird eine Ersatzeintragung für den jeweiligen Mitarbeiter erstellt und die Ersatzanfrage, welche auch von anderen Mitarbeitern eingesehen werden konnte gelöscht. Zusätzlich werden dem betreffenden Mitarbeiter Überstunden gutgeschrieben und der Dienstplan wird aktualisiert, sodass der Mitarbeiter in die Schicht eingetragen wird. Dies geschieht mit einem POST-Request auf Ersatzeintragungen an den Dienstgeber.

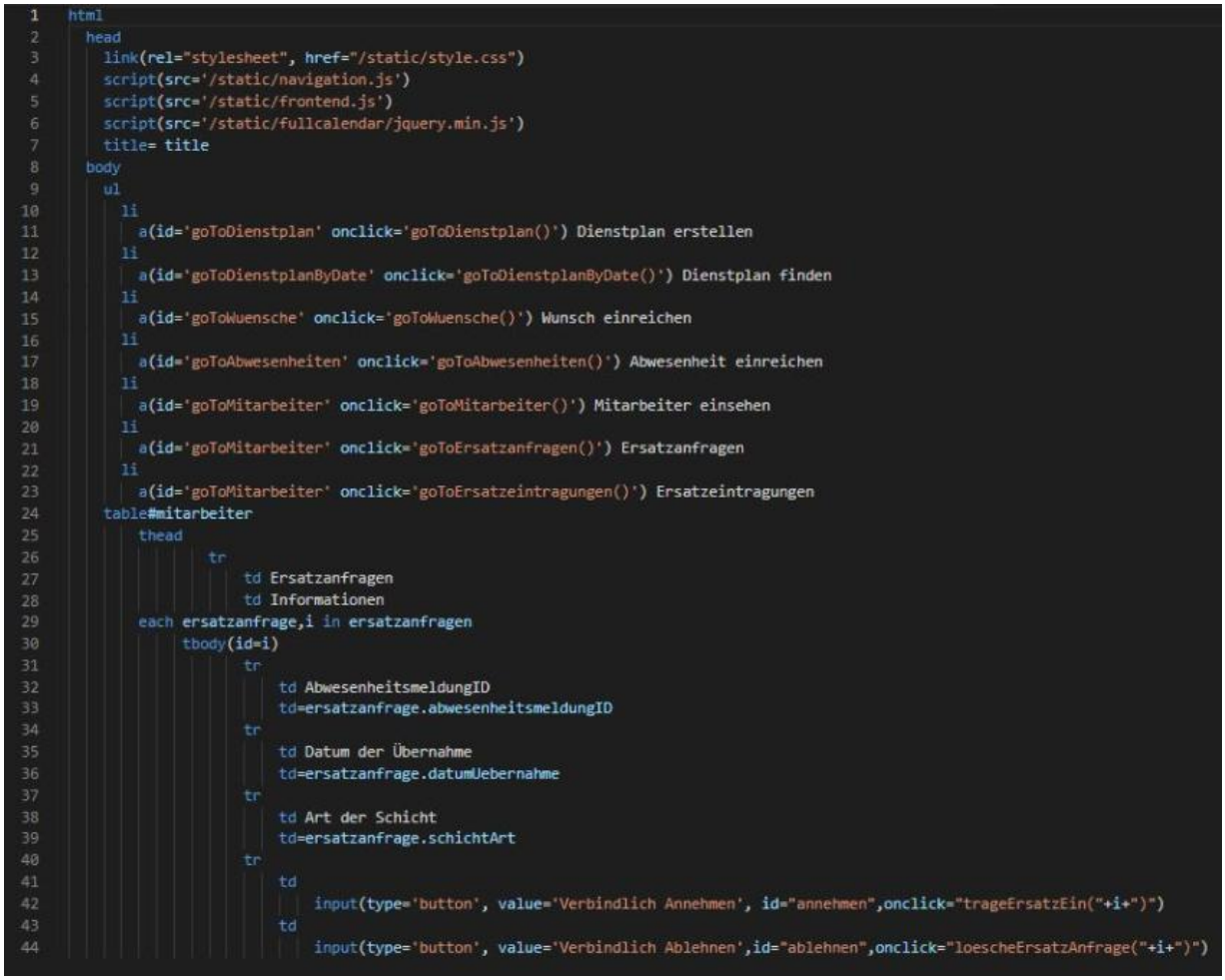


```
// POST auf Ersatzeintragung (Nach Annahme einer Ersatzanfrage)
router.post('/:mitarbeiterID/ersatzeintragungen', (req, res) => {

  var ersatzAnfrage = {
    stationID: "",
    mitarbeiterID: req.params.mitarbeiterID,
    abwesenheitsmeldungID: req.body.abwesenheitsmeldungID,
    datumUebernahme: req.body.datumUebernahme,
    schichtArt: req.body.schichtArt,
    zuErsetzenderMitarbeiter: ""
  }
  sqlHandler.getAbwesenheitenByID(ersatzAnfrage.abwesenheitsmeldungID)
    .then(function(abwesenheit) {
      //console.log(abwesenheit)
      var ersatzAnfragePromise = new Promise(function(resolve, reject) {


        ersatzAnfrage.zuErsetzenderMitarbeiter = abwesenheit[0].MitarbeiterID;
        ersatzAnfrage.stationID = abwesenheit[0].stationID;

        if (ersatzAnfrage.stationID != undefined && ersatzAnfrage.zuErsetzenderMitarbeiter != undefined) {
          resolve(ersatzAnfrage)

        } else {
          reject(err)
          console.log(err)
        }

      }) // Promise ersatzAnfragePromise
      ersatzAnfragePromise.then(function(ersatzAnfrage) {
        console.log(0)
        sqlHandler.neueErsatzeintragung(ersatzAnfrage)
          .then(function(ersatzeintragung) {
            console.log(1)
            sqlHandler.updateSchichtzuweisungErsatz(ersatzeintragung.datumUebernahme, ersatzeintragung.schichtArt, ersatzAnfrage.zuErsetzenderMitarbeiter, ersatzeintragung.mitarbeiterID)
              .then(function() {
                console.log(2)

                sqlHandler.updateUeberstunden(ersatzeintragung.mitarbeiterID, 8)
                  .then(function() {
                     console.log(ersatzAnfrage)
                    sqlHandler.loescheErsatzanfrage(ersatzAnfrage.abwesenheitsmeldungID, ersatzAnfrage.datumUebernahme)
                      .then(function() {
                        res.status(201).send("Ersatzanfrage angenommen und Diensplan erfolgreich aktualisiert")
```

(Dienstgeber)

Wird eine Ersatzanfrage verbindlich abgelehnt, so wird nur die Ersatzanfrage für den betreffenden Mitarbeiter gelöscht.

Da ein Mitarbeiter unterschiedlich viele Ersatzanfragen haben kann, müssen diese dynamisch in Abhängigkeit der Anzahl von Ersatzanfragen gerendert werden. Dies wird mit Hilfe folgender pug-Datei und einer ‘each’-Schleife realisiert.



```
html
  head
    link(rel="stylesheet", href="/static/style.css")
    script(src='/static/navigation.js')
    script(src='/static/frontend.js')
    script(src='/static/fullcalendar/jquery.min.js')
    title= title
  body
    ul
      li
        a(id='goToDienstplan' onclick='goToDienstplan()') Dienstplan erstellen
      li
        a(id='goToDienstplanByDate' onclick='goToDienstplanByDate()') Dienstplan finden
      li
        a(id='goToWuensche' onclick='goToWuensche()') Wunsch einreichen
      li
        a(id='goToAbwesenheiten' onclick='goToAbwesenheiten()') Abwesenheit einreichen
      li
        a(id='goToMitarbeiter' onclick='goToMitarbeiter()') Mitarbeiter einsehen
      li
        a(id='goToMitarbeiter' onclick='goToErsatzanfragen()') Ersatzanfragen
      li
        a(id='goToMitarbeiter' onclick='goToErsatzeintragungen()') Ersatzeintragungen
    table#mitarbeiter
        thead
                tr
                    td Ersatzanfragen
                    td Informationen
        each ersatzanfrage,i in ersatzanfragen
            tbody(id=i)
                    tr
                        td AbwesenheitsmeldungID
                        td=ersatzanfrage.abwesenheitsmeldungID
                    tr
                        td Datum der Übernahme
                        td=ersatzanfrage.datumUebernahme
                    tr
                        td Art der Schicht
                        td=ersatzanfrage.schichtArt
                    tr
                        td
                            input(type='button', value='Verbindlich Annehmen', id="annehmen",onclick="trageErsatzEin("+i+")")
                        td
                            input(type='button', value='Verbindlich Ablehnen',id="ablehnen",onclick="loescheErsatzAnfrage("+i+")")
```

<u>Abweichungen zu MS2:</u>

Abweichungen zum Meilenstein 2 sind in diesem Bereich größer ausgefallen. Das Einsehen der Ersatzanfragen muss aktiv vom Mitarbeiter selbst getätigt werden. Eine Benachrichtigung wird auf Grund fehlender Session nicht gesendet. Die Implementierung der Anwendungslogik lag bei der Entwicklung im Fokus, sodass auf Grund der begrenzten Zeit, auf das Erstellen einer eigenen Session pro Mitarbeiter und die damit verbundenen Benachrichtigungen verzichtet wurde. Durch das manuelle Abrufen der Ersatzanfragen ist dennoch die volle Funktionalität des Alleinstellungsmerkmals gegeben. Als Zusatz kam das einfache Einsehen aller Ersatzeintragungen (Angenommene Ersatzanfragen) hinzu.