iHospital

Applicazioni e Servizi Web

Marco Chiavaroli - 0001036980 {marco.chiavaroli@studio.unibo.it}

22 Giugno 2022

Contents

1 Introduzione

L'applicazione descritta si occupa di gestire le informazioni e i processi principali generati da una struttura ospedaliera. Partendo dall'inserimento di un nuovo paziente in struttura, su di esso sarà possibile effettuare tutte le operazioni principali relative alla sua anagrafica, registrare i risultati di ogni analisi effettuata, prescrivere e somministrare farmaci, segnalare lo stato di avanzamento del paziente ed infine programmare e monitorare in real-time le operazioni in sala operatoria. Il corretto funzionamento di ogni funzionalità è garantito da un adeguata divisione dei ruoli. Tale divisione si sviluppa su **tre livelli di utenza** (Direttore ospedaliero, Medico, Infermiere) che verranno meglio descritti nel paragrafo successivo.

2 Requisiti

2.1 Requisiti funzionali

Di seguito sono descritte le funzionalità a disposizione dell'utente che accede al sistema, in base al suo ruolo di registrazione ad esso. A queste funzionalità si aggiungono quelle generiche per ogni livello di utenza (login con credenziali, gestione area riservata, modifica password di accesso).

2.1.1 Direttore Ospedaliero

Il direttore ospedaliero è l'utente in cima all'architettura, che quindi si occupa di **direzionare il carico di lavoro** verso gli altri due livelli di utenza. In particolare può:

- Visualizzare nominativi di medici ed infermieri registrati al sistema.
- Inserire un nuovo paziente al sistema nel momento del suo arrivo in ospedale (check-in).
- Visualizzare la lista completa di tutti i pazienti (attualmente ricoverati e dimessi) registrati al sistema.
- Per ogni paziente può:
    (a) visualizzare e modificare anagrafica;
    (b) solo visualizzare lista dei farmaci prescritti;
    (c) solo visualizzare dettagli delle operazioni;
    (d) visualizzare ed aggiungere registrazioni sui valori vitali;
    (e) chattare nell'area di segnalazioni;
    (f) assegnare il paziente ai medici ed infermieri che quindi lo prenderanno in cura.
- Visualizzare la lista dei farmaci ed aggiungerne di nuovi al sistema.

- Visualizzare i feedback inviati da medici ed infermieri su eventuali problematiche software.

2.1.2 Medico

Il medico può visualizzare e quindi gestire **solo i pazienti che gli vengono assegnati da un direttore ospedaliero**, e comunque finché quest'ultimo non lo rimuoverà dall'incarico. Durante questo periodo può:

- Visualizzare la lista dei propri pazienti in carico (attualmente ricoverati e dimessi).
- Per ogni paziente a lui assegnato potrà:
  1. visualizzare e modificare anagrafica;
  2. visualizzare e prescrivere nuovi farmaci;
  3. visualizzare, modificare i dati di una operazione e firmare il verbale di avvenuto intervento;
  4. visualizzare ed aggiungere registrazioni sui valori vitali;
  5. chattare nell'area di segnalazioni;
  6. visualizzare la lista degli operatori che collaborano con lui su tale paziente.
- Visualizzare la lista dei farmaci ed aggiungerne di nuovi al sistema.
- Visualizzare il calendario delle operazioni programmate per i suoi pazienti e fissare un nuovo intervento per uno di loro.
- Inviare feedback sul sistema al Direttore Ospedaliero.

2.1.3 Infermiere

L'infermiere, analogamente al medico, può visualizzare e quindi gestire **solo i pazienti che gli vengono assegnati da un direttore ospedaliero**, e comunque finché quest'ultimo non lo rimuoverà dall'incarico. Durante questo periodo può:

- Visualizzare la lista dei propri pazienti in carico (attualmente ricoverati e dimessi).
- Per ogni paziente a lui assegnato potrà:
  1. solo visualizzare l'anagrafica;
  2. somministare le dosi giornaliere per i soli farmaci precedentemente prescritti da un medico;
  3. solo visualizzare dettagli delle operazioni;
  4. visualizzare ed aggiungere registrazioni sui valori vitali;

5. chattare nell'area di segnalazioni;
6. visualizzare la lista degli operatori che collaborano con lui su tale paziente.

- Visualizzare il calendario delle operazioni programmate per i suoi pazienti.
- Inviare feedback sul sistema al Direttore Ospedaliero.

2.2 Requisiti non funzionali

Si vuole ottenere un sistema in grado di:

- essere acessibile e facilmente utilizzabile anche da un utenza meno esperta, quindi non abbastanza ferrata all'uso delle tecnologie.
- garantire sicurezza, riservatezza ed integrità dei dati tramite l'accesso ad ogni funzioanlità attraverso un meccanismo di autenticazione.
- garantire disponibilità 24h in maniera reattiva e snella, per permettere uno svolgimento delle operazioni quotidiane molto rapido.

3 Design

3.1 Metodologie e scelte di design

Per la realizzazione del design ci si è concentrati sin da subito su una strategia di Human Computer Interaction, scegliendo una grafica ed una modalità semplice di interazione software-utente. La sottoposizione continua, durante le fasi di sviluppo, all'uso delle funzionalità ha permesso di trovare eventuali problematiche d'uso e risolverle tempestivamente. Tale uso in fase di sviluppo è stato seguito tramite la creazione di utenti "virtuali", ovvero una serie di Personas che rispecchiano l'utente medio che in seguito utilizzerà effettivamente l'applicativo. Il sistema garantisce un buon livello di Responsive Design, adattandosi graficamente in maniera automatica al tipo di dispositivo e di spazio utilizzabile sullo schermo. Si è seguito anche il principio KISS per permettere la realizzazione di pagine semplici, per nulla confusionarie e quindi facilmente accessibili dall'uente medio.

3.2 Mockup

L'applicativo ha massima friubilità in versione desktop. Questa scelta è dovuta al fatto che in ogni centro ospedaliero ogni operatore ha accesso ad un terminale messo a disposizione dalla struttura proprio per svolgere i compiti assegnati. L'operatore accede alle sue funzionalità tramite email e password, in seguito una dashboard è mostrata in base al tipo di utente che accede.

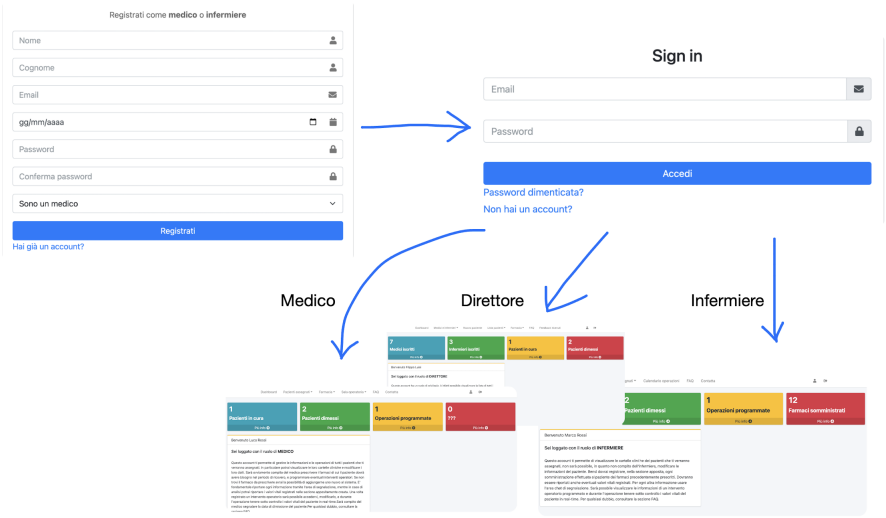


Fig.1 Design di accesso all'area utente

Una volta eseguito l'accesso ogni utente potrà svolgere i propri compiti in base al ruolo assegnatogli. In particolare, le principali attività si diramano a partire dal profilo di un paziente, accessibile dalla lista dei pazienti in carico all'utente loggato. Il profilo si divide a sinistra con il nominativo del paziente ed alcuni bottoni per l'accesso rapido a funzionalità come registrazioni delle analisi vitali o chat real-time e a destra con uno spazio di quattro schede (angrafica, farmaci, operazioni, equipe) ognuna con un diverso livello di autorizzazione all'accesso delle funzionalità in base all'utente loggato.

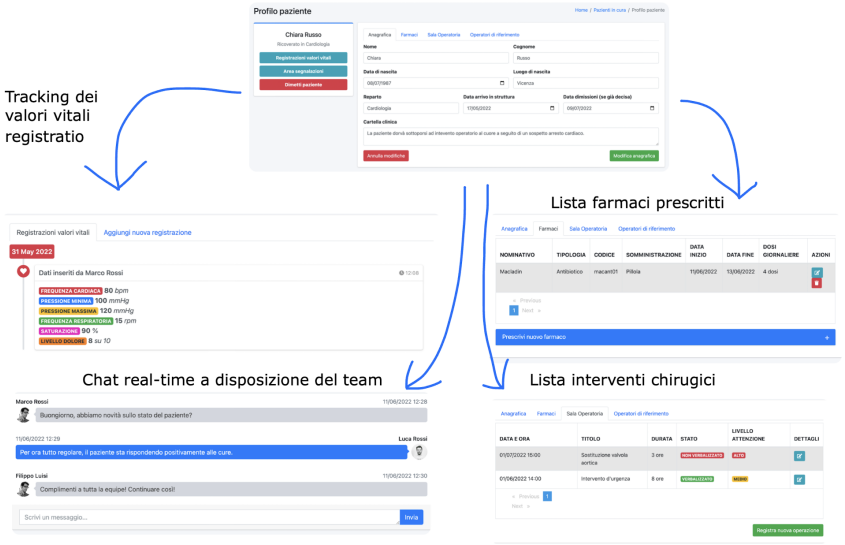


Fig.2 Principali diramazioni dal profilo del paziente

Per permettere l'accesso a tali funzionalità un direttore ospedaliero, oltre a potre visualizzare la lista dei membri dell'equipe, può anche inserire o eliiminare un membro al team. In questo modo tutti gli utenti del equipe potranno accedere alle funzionalità relative a tale paziente in base al loro ruolo.

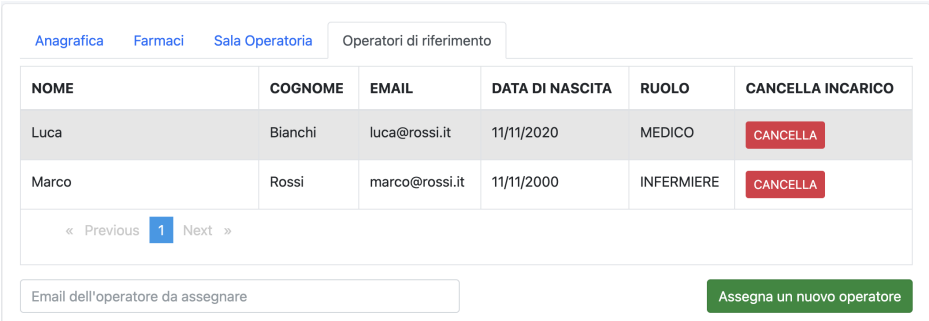


Fig.3 Assegnazione membri equipe

Tra le principali pagine si sottolinea anche quella di gestione della sala operatoria, infatti in questa pagina è possbile visualizzare i dettagli di un intervento

(modificabili solo dagli utenti medici) e se l'operazione è fissata per la giornata odierna saranno visibili i parametri vitali real-time del paziente, anche graficamente (dati simulati).

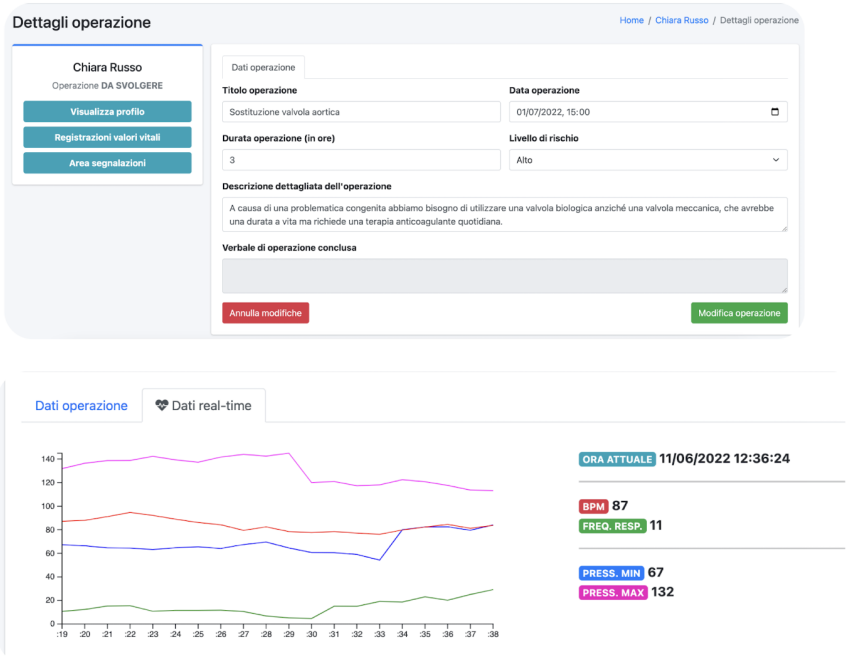


Fig.4 Dettagli operazione

Un'altra funzione principale riguarda gli infermieri, infatti questi hanno accesso ad una pagina riservata ad ogni farmaco prescritto (da un medico) ad un paziente e da qui possono registrare tutte le somministrazioni di tale farmaco che vengono effettuate. Il sistema si occuperà di verificare che tali somministrazioni siano svolte nell'intervallo di tempo e nel numero di dosi precedentemente prescritte dal medico.

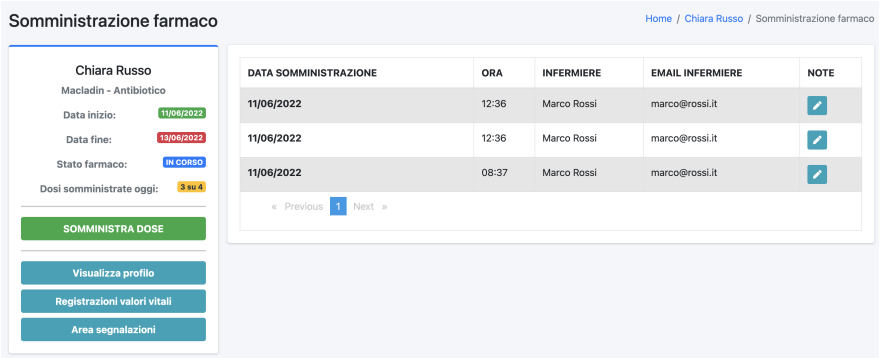


Fig.5 Somministrazione di un farmaco

Di seguito sono mostrate le schermate di altre pincipali funzionalità.

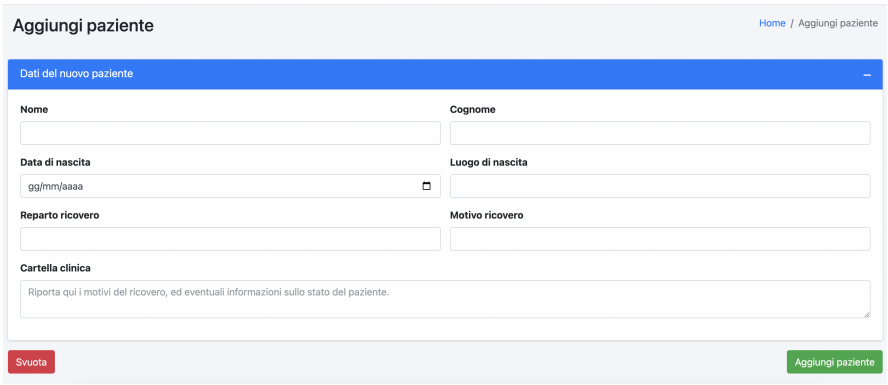


Fig.6 Aggiunta nuovo paziente

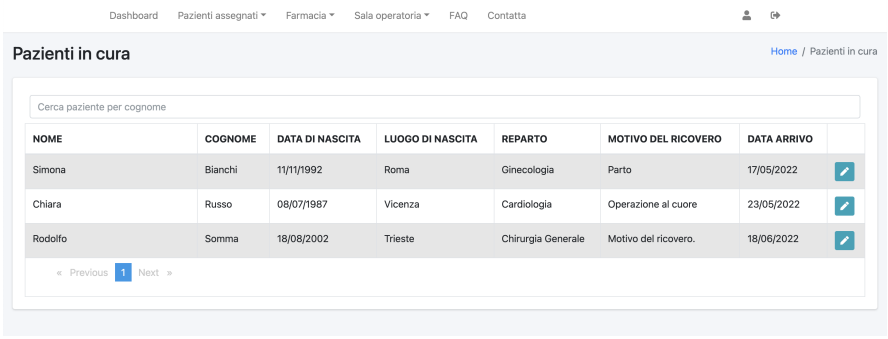


Fig.7 Lista pazienti ricoverati

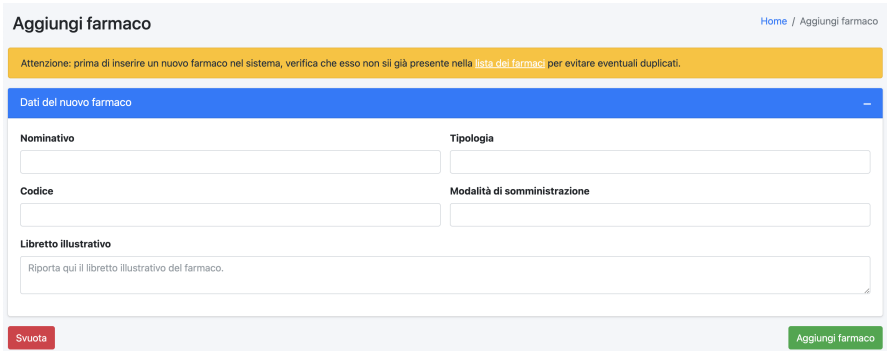


Fig.8 Aggiunta nuovo farmaco

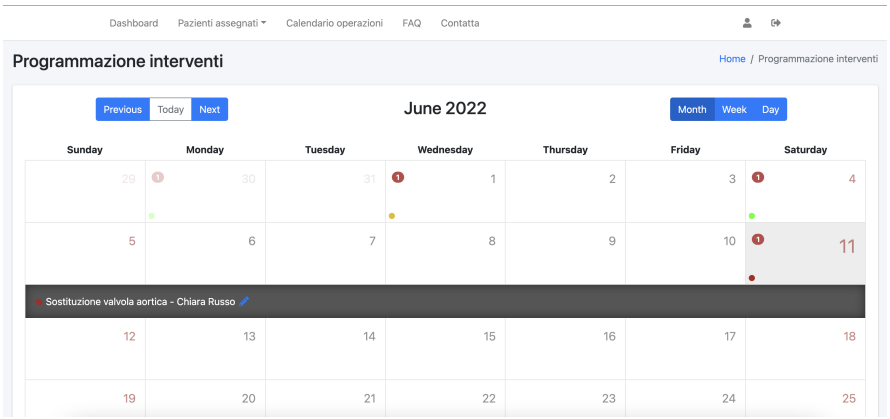


Fig.9 Calendario operazioni programate

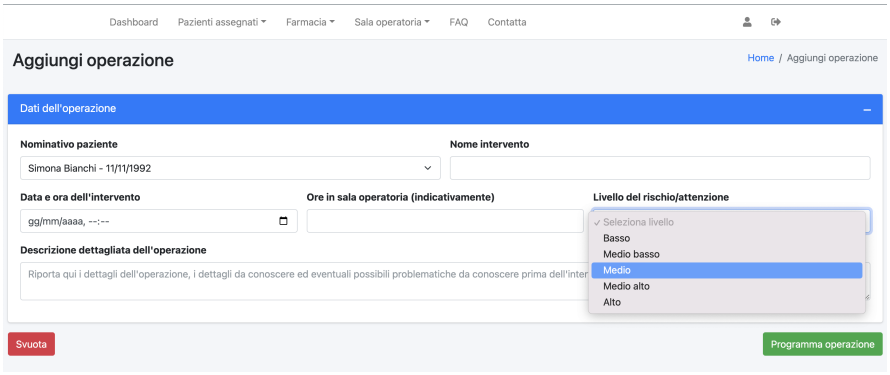


Fig.10 Programmazione nuova operazione

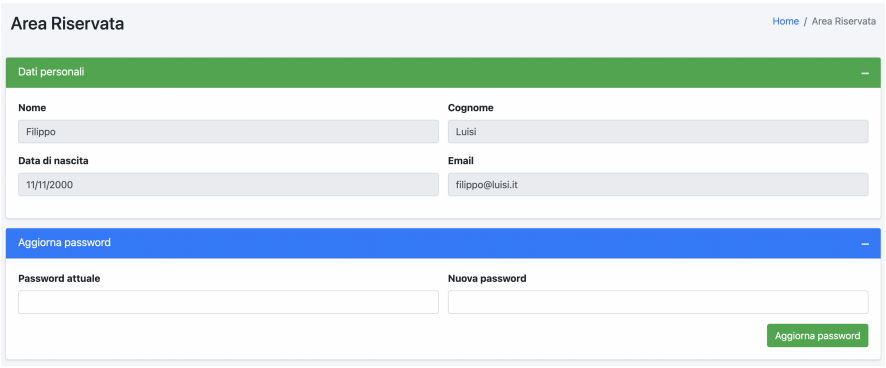


Fig.11 Area privata

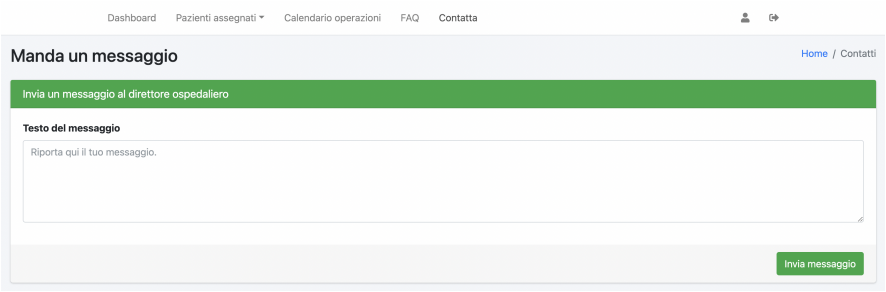


Fig.12 Area contatti

3.3 Analisi Targer User

Questa sottosezione vuole inquadrare meglio l'applicativo nel suo uso nel quotidiano. Per far ciò sono state realizzate delle Personas non particolarmente avvezze all'uso delle tecnologie.

3.3.1 Persons: Claudia (Direttore Ospedaliero)

Quotidianamente Claudia si reca in ospedale, accede al suo terminale e si occupa di inserire i nuovi pazienti arrivati in struttura. Grazie alla semplicità ed alla chiarezza delle schede dell'applicativo, Claudia riesce con facilità a ricoprire questo compito. In seguito si occupa di assegnare o modificare le equipe dei pazienti di suo interesse, quindi accede al profilo di quest'ultimi e tramite le email degli operatori (che può ritrovare dall'elenco degli utenti iscritti a lui visibile) inserisce i medici e gli infermieri di suo interesse. Infine Claudia ha l'abitudine di accedere nelle chat dei pazienti più in osservazione per leggere eventuali messaggi inviati dagli operatori sanitari sullo stato di salute del paziente.

3.3.2 Persons: Luca (Medico)

Luca è un chirurgo che lavora presso la struttura ospedaliera che fa uso di iHospital. Durante la giornata Luca visita i pazienti che gli sono stati assegnati dal direttore e si occupa di inserire nell'applicativo i risultati delle sue analisi. Quando lo ritiene opportuno si occupa di prescrivere eventuali farmaci segnalando eventuali novità sul paziente. Luca riesce ad avere un quadro chiaro degli interventi chiurgici che dovrà sostenere grazie al calendario presente nel sistema e, durante tali operazioni iHospital lo accompagna in sala operatoria per monitorare i parametri del paiente.

3.3.3 Persons: Marco (Infermiere)

Marco è un infermiere presso il reparto di chirurgia e, quando è di turno, registra man mano ogni sua operazione svolta in reparto in modo da tenere traccia di ogni movimento. Non registra solo i risultati di eventuali visite, ma si occupa anche di verificare se ci sono dosi di farmaci da somministrare ad uno dei suoi pazienti ed in tal caso si occupa di farlo, reguistrando l'operazione su iHospital. Questo permette di tracciare in ogni momento chi ha fatto cosa ed allo stesso tempo di tenere sotto traccia i pazienti a cui vanno portati dei farmaci e quali invece non ne hanno attualmente bisogno.

4 Tecnologie

4.1 Stack MEAN

Per la realizzazione dell'applicativo si è fatto uso dello **Stack MEAN**. Quindi si compone di un lato di backend basato su server **Node.js** con il framework **Express.js** a supporto, una base di dati non relazione, orientata ai documenti, quale **MongoDB** ed infine del framework **Angular** per la gestione del frontend.

4.2 Frontend

Per quanto riguarda il frontend si fa uso del framework Angular che permette di interagire in modo semplice e veloce con l'interfaccia utente. Contiene un header ed un footer che sono presenti in quasi tutte le pagine dell'applicativo. Mentre il footer è completamente statico, l'header conserva un po' di dinamicità poiché gli item del menu da mostare **variano in base al tipo di utente** che accede al sistema. Successivamente la parte centrale di ogni pagina viene costruita da uno o più component realizzati. Una route si occupa quindi di direzionare il component adatto in base alla pagina richiesta dall'utente. Sono state realizzate anche delle interfacce CanActive che permettono di verificare se l'utente ha i permessi per accedere o meno alla pagina richiesta, in caso negativo questo verrà reindirizzato in un'altra pagina. Per quanto riguarda l'estrapolazione dei dati, sono stati realizzati dei Service che si occupano di interagire con il server che in

seguito interrogerà la base di dati e spedirà i risultati al richiedente. E' presente un service per ogni modello rappresentato:

- AdministrationService: si occupa delle richieste legate alle somministrazionei dei farmaci.
- ChartService: si occupa delle richieste legate ai parametri live durante un intervento.
- ContactService: si occupa delle richieste legati ai messaggi di contatto inviati.
- DrugService: si occupa delle richieste legate alla gestione dei farmaci.
- MedicAssignmentService: si occupa delle richieste legate alla gestione delle equipe.
- MessageService: si occupa delle richieste legate ai messaggi inviati nelle chat real-time.
- OperationService: si occupa delle richieste legate alla gestione degli interventi chirurgici.
- PatientService: si occupa delle richieste legate alla gestione dei pazienti.
- PrescriprionService: si occupa delle richieste legate alla gestione delle prescrizione mediche dei farmaci.
- UserService: si occupa delle richieste legate alla gestione degli utenti.
- VitalValueService: si occupa delle richieste legate alla gestione dei parametri vitali dei pazienti registrati.

4.3 Backend

4.3.1 Node.js

All'avvio del server viene avviato il file **app.js** che si occuperà di:

- creare un'istanza di express.js;
- connettersi al database in MongoDB;
- impostare i parametri per PassportJS (il modulo che si occuperà dell'autenticazione);
- istanziare socket.io per la gestione delle funzionalità real-time;
- istanziare le rotte per le interrogazioni al database.

Il principale lavoro lato server è stato diviso in due directory: "**model**" e "**routes**". La prima contiene la **struttura degli schemi** presenti nel database, la seconda invece contiene le **interrogazioni** effettuate a quest'ultimo. Le interrogazioni sviluppate sono indicate di seguito:

- administrations.js
    1. "/new/:id/:inf" (POST): creazione nuova somministrazione.
    2. "/find-by-interval/:id/:start" (GET): ricerca somministrazione in un intervallo di tempo.
    3. "/find-by-prescription/:id" (GET): ricerca di una somministrazione.
    4. "/find-by-nurse/:id" (GET): ricerca somministrazioni di un infermiere.
    5. "/update/:id/:note" (PUT): modifica dati somministrazione.
- contacts.js
    1. "/new" (POST): creazione nuovo contatto.
    2. "/:id" (GET): ricerca di un contatto.
    3. "/" (GET): estrazione di tutti i contatti.
- drugs.js
    1. "/new-drug" (POST): creazione nuovo farmaco.
    2. "/all": (GET) estrazione di tutti i farmaci.
    3. "/find-by-id/:id" (GET): ricerca di un farmaco.
    4. "/update-desc/:id/:desc" (PUT): modifica di un farmaco.
- medic-assignment.js
    1. "/new" (POST): creazione nuovo membro equipe.
    2. "/patient/:id" (GET): ricerca equipe di un paziente.
    3. "/medic/:id" (GET): ricerca equipes di un medico.
    4. "/:id-pat/:id-med" (GET): verifica che un medico faccia parte dell'equipe di un paziente.
    5. "/delete/:id-pat/:id-med" (DELETE): eliminazione associazione
- messages.js
    1. /new" (POST): creazione nuovo messaggio.
    2. "/:id" (GET): tutti i messaggi di una chat.
- operations.js
    1. "/new-operation" (POST): creazione operazione.
    2. "/all" (GET): tutte le operazioni.
    3. "/find-by-patient/:id" (GET): tutte le operazioni di un paziente.
    4. "/:id" (GET): ricerca di una operazione.
    5. "/update/:id" (PUT): modifica di una operazione.

6. "/delete/:id" (DELETE): eliminazione operazione.

- patients.js
  1. "/new-patient" (POST): creazione paziente.
  2. "/all" (GET): tutti i pazienti.
  3. "/current" (GET): tutti i pazienti attualmente ricoverati.
  4. "/dismiss" (GET): tutti i pazienti dimessi.
  5. /:id" (GET): ricerca di una paziente.
  6. "/update/:id" (PUT): modifica dati di un paziente.
- prescriptions.js
  1. "/new" (POST): creazione nuova prescrizione.
  2. "/find-by-id/:id" (GET): ricerca prescrizione.
  3. "/find-by-patient/:id" (GET): tutte le prescrizioni di un paziente.
  4. "/delete/:id-pat/:id-drug" (DELETE): eliminazione prescrizione.
  5. "/update/:id/:note" (PUT): modifica dati di una prescrizione.
- users.js
  1. "/login" (POST): tentativo di login con verifica credenziali.
  2. "/registrati" (POST): registrazione nuovo utente.
  3. "/user.logged" (GET): verifica che l'utente è loggato.
  4. "/find-by-email:email" (GET): ricerca utente tramite email.
  5. "/find-by-id:id" (GET): ricerca utente tramite id.
  6. "/find-by-role:role" (GET): ricerca utenti con un determinato ruolo.
  7. "/update-pwd/:id" (PUT): aggiornamento password.
- vital-values.js
  1. "/new/:my-id/:id-pat" (POST): creazione nuova analisi.
  2. "/all" (GET): tutte le analisi.
  3. "/find-by-patient/:id" (GET): tutte le analisi di un paziente.
  4. "/delete/:id" (DELETE): eliminazione analisi.

4.3.2 MongoDB

I dati generati dall'applicativo sono salvati in un database MongoDB, e suddivisi in collezioni che rispecchiano perfettamente i models descritti nel paragrafo precedente. L'elasticità di un database non relazione permette di raccogliere dati in maniera molto flessibile, lasciando così anche più libertà all'utente finale nella gestione delle informazioni inserite.

4.4 Passport.js

Il login degli utenti è gestito tramite Passport.js, un middleware di autenticazione per Node.js. E' stato scelto poiché estremamente flessibile e modulare da utilizzare insieme ad Express. Permette di realizzare un sistema di autenticazione tramite la coppia di valori email e password.

4.5 Socket.io

La libreria Socket.io è stata utilizzata per gestire le funzionalità real-time dell'applicativo. E' usata per la realizzazione di due funzioni.

4.5.1 Parametri real-time sala operatoria

Durante un'operazione, gli utenti hanno accesso ad una scheda che riceve, tramite Socket.io i dati dei parametri vitali del paziente e di conseguenza aggiorna costantemente le etichette dei valori ed il grafico.

4.5.2 Chat real-time

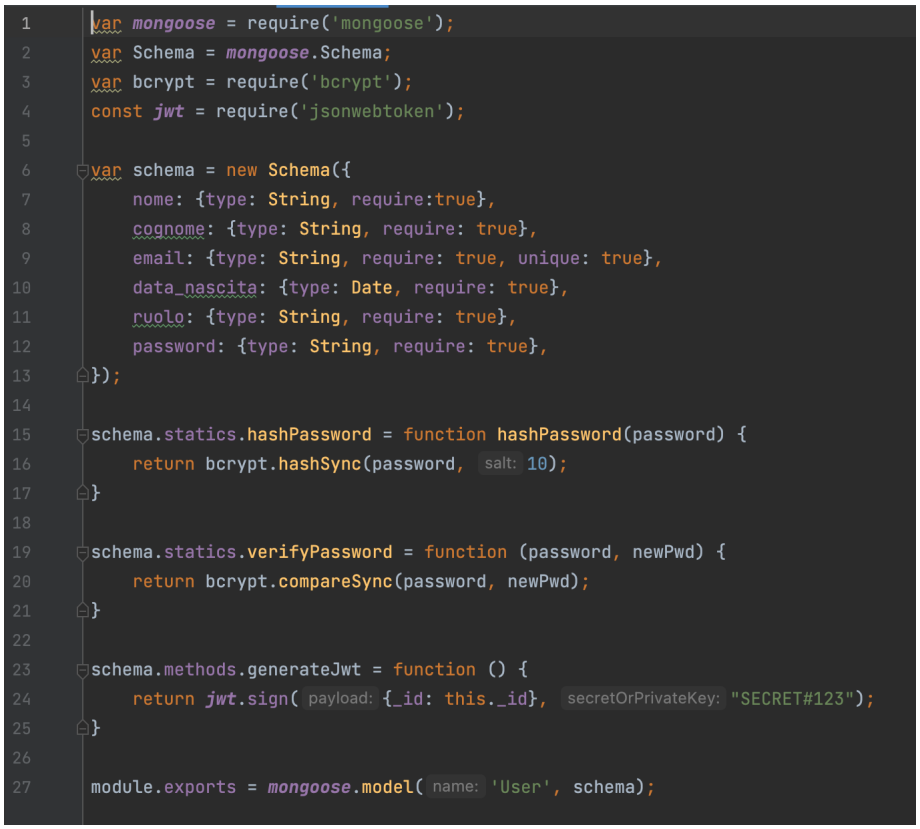
La libreria permette, tramite una comunicazione bidirezionale client-server di gestire una chat di messagistica istantanea notificando l'arrivo di un nuovo messaggio ad ogni utente connesso alla chat, mostrando quindi il testo ed il mittente.

5 Codice

Di seguito vengono mostrati alcuni frammenti di codice, cercando di mettere in evidenza le parti più centrali dell'applicativo. Si sottolinea come molte strategie di scrittura del codice possono essere generalizzate per tutte le altre funzionalità similari del progetto.

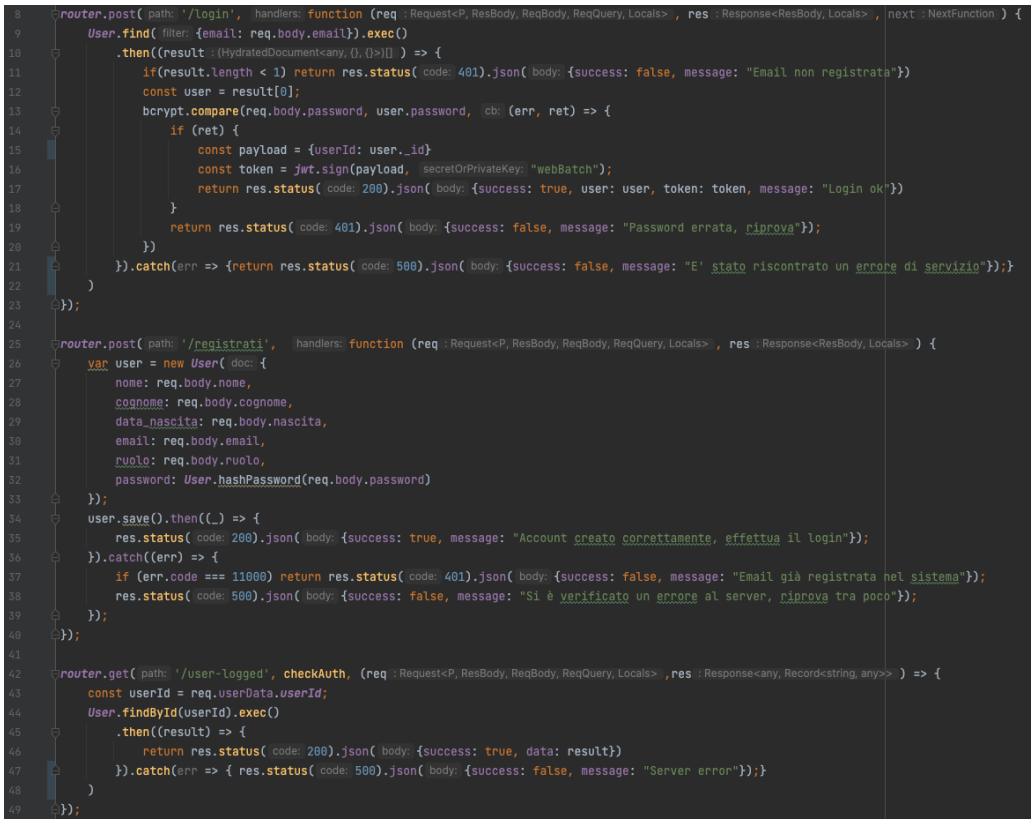
5.1 Operazioni CRUD sugli utenti

Il frammento di codice che segue mostra come sono state realizzate le interrogazione al database, in particolare sulla collezione "users" si evidenzia il funzionamento di tre operazioni (tentativo di login, registrazione nuovo utente e verifica che l'utente sia loggato).



```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');

var schema = new Schema({
    nome: {type: String, require:true},
    cognome: {type: String, require: true},
    email: {type: String, require: true, unique: true},
    data_nascita: {type: Date, require: true},
    ruolo: {type: String, require: true},
    password: {type: String, require: true},
});

schema.statics.hashPassword = function hashPassword(password) {
    return bcrypt.hashSync(password, salt: 10);
}

schema.statics.verifyPassword = function (password, newPwd) {
    return bcrypt.compareSync(password, newPwd);
}

schema.methods.generateJwt = function () {
    return jwt.sign( payload: {_id: this._id}, secretOrPrivateKey: "SECRET#123");
}

module.exports = mongoose.model( name: 'User', schema);
```

Fig.13 ./models/user.js



```
router.post( path: '/login',  handlers: function (req : Request<P, ResBody, ReqBody, ReqQuery, Locals> , res : Response<ResBody, Locals> , next : NextFunction ) {
    User.find( filter: {email: req.body.email}).exec()
        .then((result : (HydratedDocument<any, {}, {}>)[] ) => {
            if(result.length < 1) return res.status( code: 401).json( body: {success: false, message: "Email non registrata"})
            const user = result[0];
            bcrypt.compare(req.body.password, user.password,  cb: (err, ret) => {
                if (ret) {
                    const payload = {userId: user._id}
                    const token = jwt.sign(payload,  secretOrPrivateKey: "webBatch");
                    return res.status( code: 200).json( body: {success: true, user: user, token: token, message: "Login ok"})
                }
                return res.status( code: 401).json( body: {success: false, message: "Password errata, riprova"});
            })
        }).catch(err => {return res.status( code: 500).json( body: {success: false, message: "E' stato riscontrato un errore di servizio"});}
    )
});

router.post( path: '/registrati',  handlers: function (req : Request<P, ResBody, ReqBody, ReqQuery, Locals> , res : Response<ResBody, Locals> ) {
    var user = new User( doc: {
        nome: req.body.nome,
        cognome: req.body.cognome,
        data_nascita: req.body.nascita,
        email: req.body.email,
        ruolo: req.body.ruolo,
        password: User.hashPassword(req.body.password)
    });
    user.save().then((_) => {
        res.status( code: 200).json( body: {success: true, message: "Account creato correttamente, effettua il login"});
    }).catch((err) => {
        if (err.code === 11000) return res.status( code: 401).json( body: {success: false, message: "Email già registrata nel sistema"});
        res.status( code: 500).json( body: {success: false, message: "Si è verificato un errore al server, riprova tra poco"});
    });
});

router.get( path: '/user-logged', checkAuth, (req : Request<P, ResBody, ReqBody, ReqQuery, Locals> ,res : Response<any, Record<string, any>> ) => {
    const userId = req.userData.userId;
    User.findById(userId).exec()
        .then((result) => {
            return res.status( code: 200).json( body: {success: true, data: result})
        }).catch(err => { res.status( code: 500).json( body: {success: false, message: "Server error"});}
    )
});
```

Fig.14 ./routes/users.js

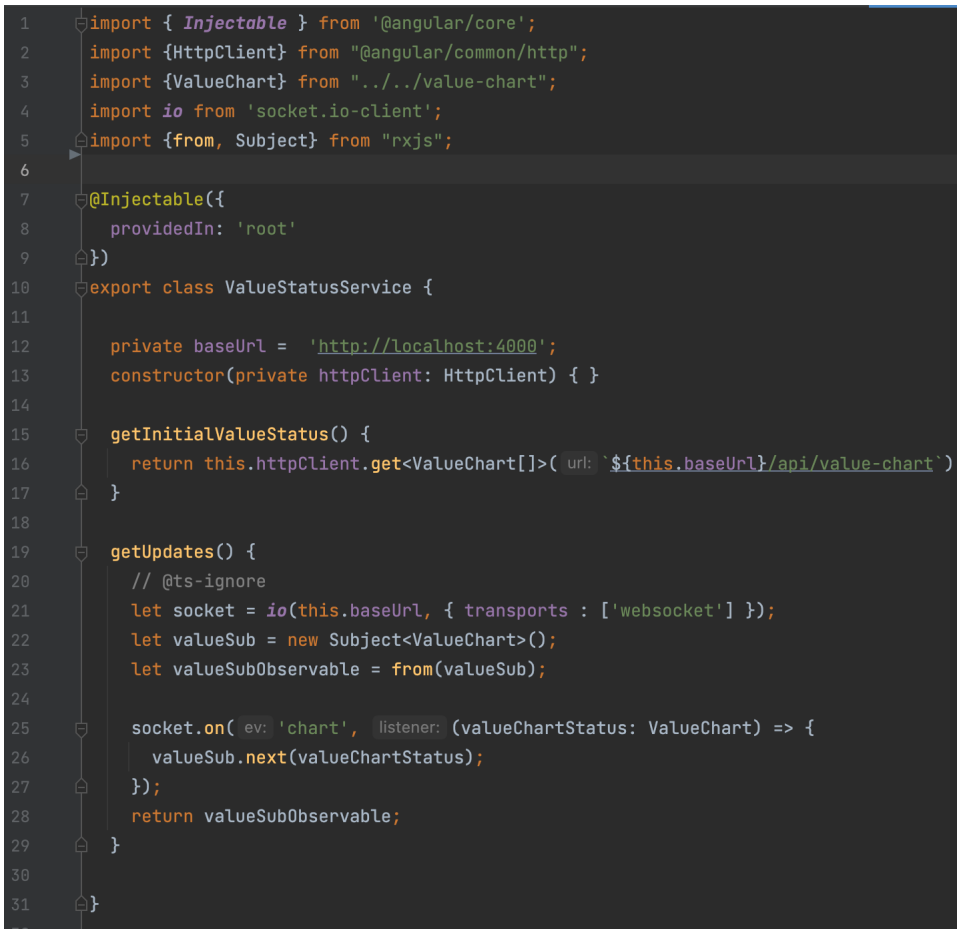
5.2 Gestione parametri vitali real-time con Socket.io

I parametri vitali real-time durante le operazioni sono ovviamente simulati in questo prototipo. Per fare ciò il file JS che segue nella prossima immagine si occupa di **randomizzare i valori** di frequenza cardiaca, frequenza respiratioria, pressione minima e pressione massima. Si sottolinea che si tratta di una randomizzazione "controllata", infatti in ogni secondo i valori possono variare di più o meno cinque punti rispetto il valore prececedente e comunque, qualora questi valori superassero dei minimi o massimi prestabiliti, verrebbero reimpostati ad un valore iniziale di base.

```
const moment = require('moment');
const valChart = [
    { "hr": getRandom(55,110), "press_min": getRandom(70,90), "press_max": getRandom(110,135),
        "freq_resp": getRandom(3,18), date: (new Date()) },
]
let counter = 0;

function updateValueChart() {

    const lastDay = moment(valChart[0].date, 'hh:mm:ss').add( amount: 1,  unit: 'seconds');
    let hr, press_min, press_max, freq_resp, date;

    hr = valChart[0].hr + getRandom(-5,5);
    press_min = valChart[0].press_min + getRandom(-5,5);
    press_max = valChart[0].press_max + getRandom(-5,5);
    freq_resp = valChart[0].freq_resp + getRandom(-5,5);

    // range data
    if (hr < 40 || hr > 160) hr = 100;
    if (press_min < 50 || press_min > 110) press_min = 80;
    if (press_max < 90 || press_max > 145) press_max = 120;
    if (freq_resp < 1 || freq_resp > 30) freq_resp = 15;

    valChart.unshift( items: {
        hr, press_min, press_max, freq_resp, date: lastDay.format( format: 'hh:mm:ss'),
    });
    counter++;

}

module.exports = { valChart, updateValueChart }

function getRandom(min, max) {
    return Math.random() * (max - min) + min;
```

Fig.15 value-chart.js

Nel frammento che segue si mostra come i dati vengono raccolti lato frontend da un Service apposito che è quindi utilizzato dal component che si occupa di realizzare il grafico con i dati ricevuti ed aggiornarlo ogni secondo.



```
import { Injectable } from '@angular/core';
import {HttpClient} from "@angular/common/http";
import {ValueChart} from "../../value-chart";
import io from 'socket.io-client';
import {from, Subject} from "rxjs";

@Injectable({
  providedIn: 'root'
})
export class ValueStatusService {

  private baseUrl =  'http://localhost:4000';
  constructor(private httpClient: HttpClient) { }

  getInitialValueStatus() {
    return this.httpClient.get<ValueChart[]>( url: `${this.baseUrl}/api/value-chart`)
  }

  getUpdates() {
    // @ts-ignore
    let socket = io(this.baseUrl, { transports : ['websocket'] });
    let valueSub = new Subject<ValueChart>();
    let valueSubObservable = from(valueSub);

    socket.on( ev: 'chart',  listener: (valueChartStatus: ValueChart) => {
      valueSub.next(valueChartStatus);
    });
    return valueSubObservable;
  }

}
```

Fig.16 value-status.service.ts

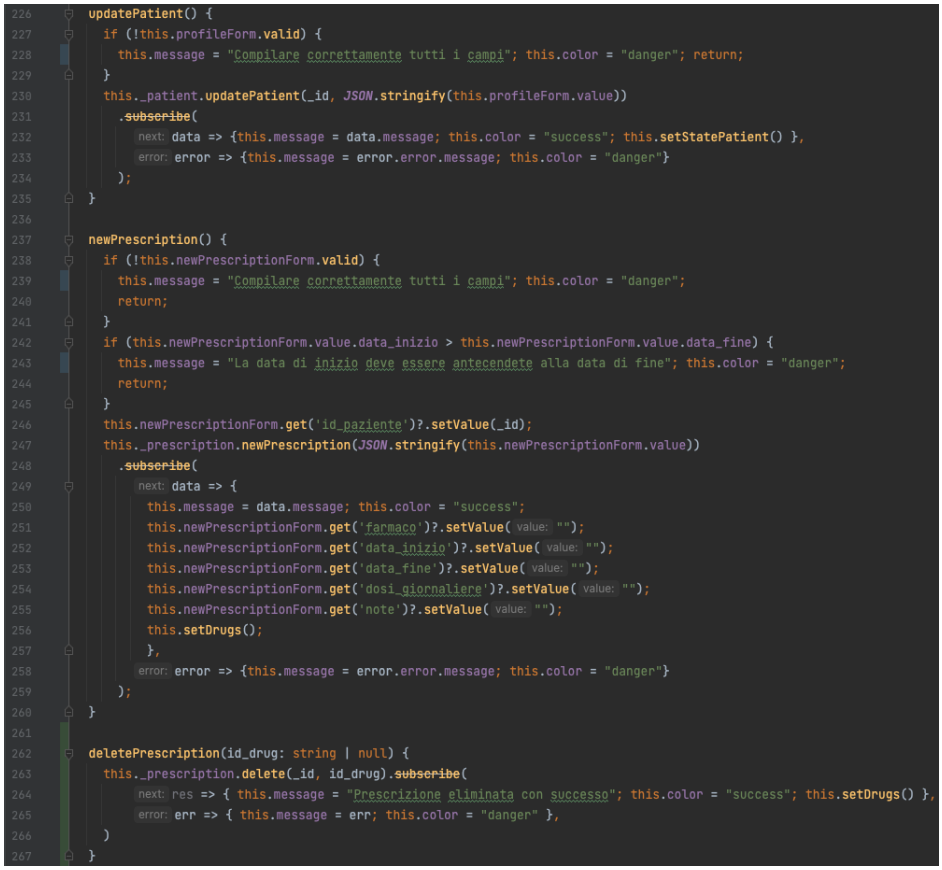
```
    valueChartStatus: ValueChart[] | undefined;
    valueChartStatusToPlot: ValueChart[] | undefined;

    set ValueStatus(status: ValueChart[]) {
        this.valueChartStatus = status;
        this.valueChartStatusToPlot = this.valueChartStatus.slice(0, 20);
    }

    constructor(private route: ActivatedRoute, private _patient: PatientService,
                private _operation: OperationService, private _valueStatus: ValueStatusService) {
        this.idOperation = this.route.snapshot.paramMap.get('id');
        this._operation.findById(this.idOperation)
            .subscribe(
                next: res => {
                    this.operation = res.data;
                    this._patient.findById(this.operation.id_paziente)
                        .subscribe(
                            next: res => {this.patient = res.data},
                        );
                    this.setValueForm();
                },
                error: err => {
                }
            );
        this._valueStatus.getInitialValueStatus()
            .subscribe( next: prices => {
                this.ValueStatus = prices;
                let valueUpdateObservable =  this._valueStatus.getUpdates();
                valueUpdateObservable.subscribe( next: (latestStatus: ValueChart) => {
                    // @ts-ignore
                    this.ValueStatus = [latestStatus].concat(this.valueChartStatus);
                });
            });
    }
```

Fig.17 dettagli-operazione.component.ts

5.3 Controller azioni sul profilo del paziente

Gran parte delle funzionalità si diramano a partire dalla pagina del profilo di un paziente, di conseguenza il codice sorgente più corposo ed articolato di sviluppa nel file profilo-paziente.components.ts. In questo file sono attivate tutte le adeguate richieste verso il server in base alle azioni eseguite dall'utente. Il frammento di codice che segue riporta i metodi che si attivano a seguito della richiesta (tramite i bottoni appositi) dell'utente di:

1. aggiornare i dati di un paziente;
2. prescrivere un nuovo farmaco al paziente;
3. rimuovere la prescrizione di un farmaco dal paziente.



```
updatePatient() {
  if (!this.profileForm.valid) {
    this.message = "Compilare correttamente tutti i campi"; this.color = "danger"; return;
  }
  this._patient.updatePatient(_id, JSON.stringify(this.profileForm.value))
    .subscribe(
      next: data => {this.message = data.message; this.color = "success"; this.setStatePatient() },
      error: error => {this.message = error.error.message; this.color = "danger"}
    );
}

newPrescription() {
  if (!this.newPrescriptionForm.valid) {
    this.message = "Compilare correttamente tutti i campi"; this.color = "danger";
    return;
  }
  if (this.newPrescriptionForm.value.data_inizio > this.newPrescriptionForm.value.data_fine) {
    this.message = "La data di inizio deve essere antecendete alla data di fine"; this.color = "danger";
    return;
  }
  this.newPrescriptionForm.get('id_paziente')?.setValue(_id);
  this._prescription.newPrescription(JSON.stringify(this.newPrescriptionForm.value))
    .subscribe(
      next: data => {
        this.message = data.message; this.color = "success";
        this.newPrescriptionForm.get('farmaco')?.setValue( value: "");
        this.newPrescriptionForm.get('data_inizio')?.setValue( value: "");
        this.newPrescriptionForm.get('data_fine')?.setValue( value: "");
        this.newPrescriptionForm.get('dosi_giornaliere')?.setValue( value: "");
        this.newPrescriptionForm.get('note')?.setValue( value: "");
        this.setDrugs();
        },
      error: error => {this.message = error.error.message; this.color = "danger"}
    );
}

deletePrescription(id_drug: string | null) {
  this._prescription.delete(_id, id_drug).subscribe(
      next: res => { this.message = "Prescrizione eliminata con successo"; this.color = "success"; this.setDrugs() },
      error: err => { this.message = err; this.color = "danger" },
  )
}
```

Fig.18 profilo-paziente.component.ts

5.4 Route Guard

Per l'accesso a determinate pagine, sono richiesti determinati privilegi, ad esempio il medico può accedere solamente al profilo dei pazienti che esso tiene in cura. Per svolgere tali controlli sono state realizzate delle **guardie**, che si occupano di autorizzare o meno l'accesso a tali pagine in base al tipo di utente loggato. Se l'accesso è autorizzato, l'utente viene "lasciato passare", altrimenti sarà reinderizzato verso un'altra pagina a cui invece gli è permesso accedere. Di seguito le guardie implementate:

1. DirectorGuard: autorizzato ad accedere alle pagine del direttore ospedaliero.
2. MedicGuard: autorizzato ad accedere alle pagine del medico.
3. NurseGuard: autorizzato ad accedere alle pagine dell'infermiere

4. DirectorOrMedicGuard: autorizzato ad accedere alle pagine del direttore ospedaliere o del medico (per le pagine accessibili da entrambri i tipi di utenti).

5. MedicOrNurseGuard: autorizzato ad accedere alle pagine del medico o dell'infermiere (per le pagine accessibili da entrambri i tipi di utenti).

6. MyPatientGuard: autorizzato ad accedere alle pagine del paziente se quest'ultimo è assegnato tra gli incarichi dell'utente loggato.

7. MyOperationGuard: autorizzato ad accedere alle pagine delle operazioni del paziente se quest'ultimo è assegnato tra gli incarichi dell'utente loggato.

8. MyPrescriptionGuard: autorizzato ad accedere alle pagine delle prescrizioni del paziente se quest'ultimo è assegnato tra gli incarichi dell'utente loggato.

```typescript
@Injectable({
  providedIn: 'root'
})
export class MyPatientGuard implements CanActivate {

  constructor(private _auth: UserService, private route: ActivatedRoute,
              private _router: Router, private _assignment: MedicAssignmentService) { }

  // @ts-ignore
  canActivate(par: ActivatedRouteSnapshot) {
    let role = localStorage.getItem( key: "role");
    if (role === "DIRETTORE") {
      return true;
    }else{
      let id = localStorage.getItem( key: 'id');
      let patient =  par.paramMap.get('id');
        this._assignment.findById(patient, id)
          .subscribe(
              next: res => {
              if (res.data !== null) {
              }else{
                this._router.navigate( commands: ["/pazienti-in-cura"]);
              }
            },
            error: err => {
              this._router.navigate( commands: ["/pazienti-in-cura"]);
            }
          )
        return true;
    }
  }

}
```

Fig.19 MyPatient.ts

6 Test

6.1 Test Postman

Dato l'elevato numero di rotte lato server presenti per interrogare la base di dati, ognuna di essa è stata testata tramite il software Postman. Sono stati analizzati i risultati usando dei dati di test e si è verificato che ogni rotta restituisca il risultato atteso.

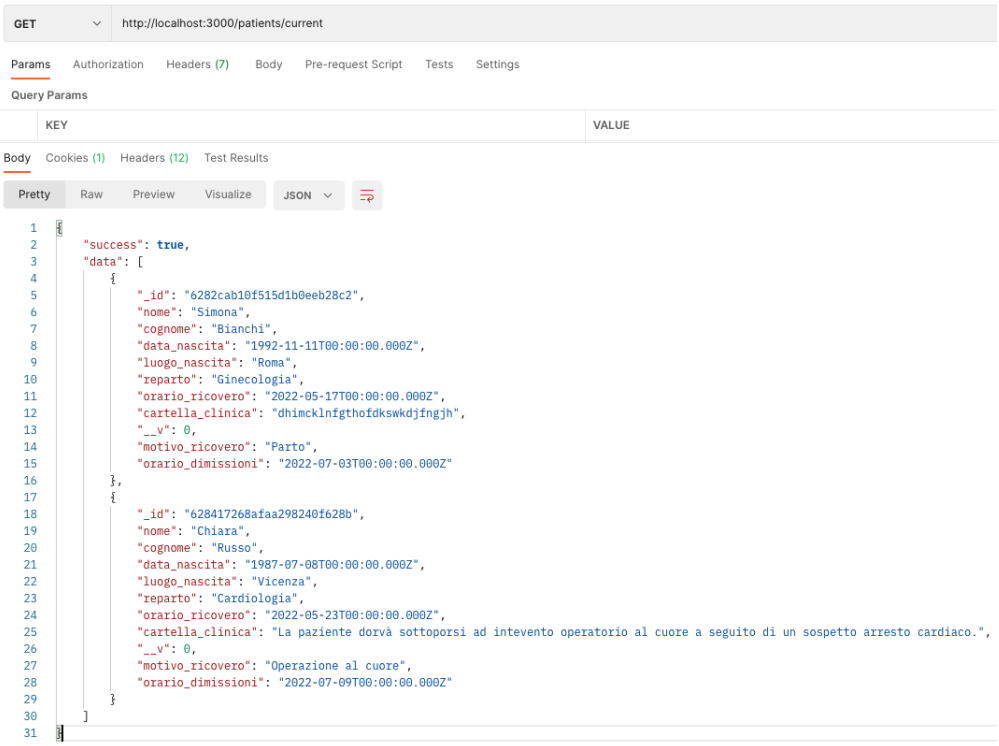


Fig.20 Esempio di test con Postman

6.2 Euristiche di Nielsen

Per migliorare al massimo i livelli di User Experience, durante lo svolgimento del software si è cercato di rispettare al meglio possibile i principali punti delle Euristiche di Nielsen:

- **Visibilità dello stato del sistema**: il sistema in qualsiasi momento riesce a fornire chiaramente all'utente cosa quest'ultimo sta svolgendo.
- **Corrispondenza tra sistema e mondo reale**: si utilizza un linguaggio molto comune ed informale, che possa arrivare in maniera diretta e chiara all'utente medio dell'applicativo.
- **Controllo e libertà**: l'utente è libero di muoversi tra le informazioni a lui accessibili, dovendo rispettare pochissimi vincoli.
- **Consistenza e standard**: in special modo nell'aspetto grafico si è cercato di rispettare il più possibile gli standard a cui l'utente potrebbe essere già abituato.

- **Prevenzione ed errore**: difficilmente l'utente può trovarsi davanti un errore e non poter tornare indietro.
- **Design ed estetica minimalista**: si è scelto un design minimalista che possa così semplificare l'uso dell'utente.

6.3 Test Users

Tutte le funzionalità sono state testate nei panni dei vari tipi di utenti che possono accedere all'applicativo. Inoltre il software è stato testato su più browser e su più tipologie di dispositivi, in modo da coprire il maggior numero di situazioni in cui un utente potrà trovarsi usando l'applicativo.

7 Deployment

7.1 Installazione

L'installazione dell'applicativo prevede l'esecuzione cronologica dei passaggi che seguono:

1. Clonare la repository al link https://github.com/marcochiavaroliunibo/iHospital.git.
2. Accedere tramite terminale alla cartella "iHospital" appena scaricata, oppure aprirla come progetto tramite un IDE (esempio: WebStorm).
3. Accedere alla directory di frontend tramite "cd frontend".
4. Installare npm tramite "npm install".
5. Tornare indietro tramite "cd .."
6. Accedere alla directory del server tramite "cd server".
7. Installare npm tramite "npm install".

7.2 Esecuzione

Per eseguire l'applicativo bisogna avviare il lato frontend ed il lato server, quindi:

- Accedere alla cartella iHospital/frontend ed eseguire il comando "ng serve" oppure il comando "npm start".
- Accedere alla cartella iHospital/server ed eseguire il comando "npm start".

Una volta avviati correttamente, accedere al link http://localhost:4200/.

8 Conclusioni

Il sottoscritto si ritiene soddisfatto del lavoro, che come già concordato è stato svolto in maniera completamente individuale per limitazioni legate ad impegni personali. Inoltre è stata rispettata la tempistica fissata ad inizio lavori per la consegna ed il prodotto finale svolge tutte le funzionalità stabilite in fase di stesura della relazione iniziale. Si tratta di un software agile da utilizzare che cerca di fare tale semplicità il suo punto di forza, dato il tipo di utilizzo che dovrebbe ricoprire. L'intero codice sorgente sarà sempre disponibile al link https://github.com/marcochiavaroliunibo/iHospital.git.