

MyClinic
*WebApp per la gestione delle
prenotazioni di visite mediche*

Sistemi Distribuiti - LM Ingegneria e Scienze Informatiche

Marco Chiavaroli - *marco.chiavaroli@studio.unibo.it*

Michele Torroni - *michele.torroni@studio.unibo.it*

01 Aprile 2023

Contents

1	Introduzione	1
2	Requisiti	2
2.1	Requisiti funzionali	2
2.1.1	Utente Medico	2
2.1.2	Utente Paziente	2
2.2	Requisiti non funzionali	3
3	Progettazione	4
4	Tecnologie Utilizzate	6
4.1	MongoDB e Mongoose	7
4.2	Express	7
4.3	Angular Js	7
4.4	Node.js	8
4.5	Socket.IO	8
4.6	Docker	9
5	Design in dettaglio	10
5.1	Backend	10
5.2	Base Dati	10
5.3	Frontend	11
5.3.1	Login	11
5.3.2	Sign Up	11
5.3.3	Application Layout	12
5.4	Interazione componenti sistema distribuito	15
6	Codice - aspetti salienti	15
6.1	Algoritmo Slots	15
6.2	Gestione Documenti	17
6.3	Visualizzazione calendario orari di lavoro di un reparto	18
6.4	Socket.io	20
7	Testing	21
8	Deployment	22
8.1	Avvio dell'applicazione	22
8.2	Fase di registrazione	22
9	Conclusioni	23

1 Introduzione

Lo scopo del sistema realizzato in questo progetto è gestire le visite mediche di una clinica privata, automatizzandone le operazioni di prenotazione e di refer-tazione.

In particolare, l'applicazione si pone lo scopo di facilitare il lavoro sia dei medici della clinica che dei pazienti. Infatti, ogni medico potrà accedere alla sua area personale per registrare gli orari settimanali in cui sarà presente in studio e potrà eseguire delle visite.

L'applicazione inoltre dividerà i medici in reparti, in modo che un paziente, una volta effettuata la registrazione al sistema, potrà visualizzare il calendario degli orari disponibili per ricevere una visita nel suo reparto di interesse. Ovviamente il paziente potrà scegliere l'orario che più gli aggrada visualizzando anche il nominativo del medico che effettuerà la visita.

Medico e paziente potranno gestire i dati della prenotazione fino al giorno dell'appuntamento ed inoltre il medico potrà aggiungere delle note per il paziente o caricare dei documenti (immagini o PDF utili ad esempio per inviare risultati o referti di eventuali analisi svolte durante la visita).

Infine, se sarà il paziente a volere entrare in contatto con il medico, sarà messa a disposizione una chat privata tra i due utenti diversa per ogni appuntamento prenotato.

2 Requisiti

Di seguito elenchiamo quali sono i requisiti che il sistema dovrà soddisfare per rendere efficiente il suo lavoro.

2.1 Requisiti funzionali

Di seguito sono descritte le funzionalità a disposizione degli utenti che accedono al sistema, in base al loro ruolo di registrazione. A queste funzionalità si aggiungono quelle generiche per ogni livello di utenza (login con credenziali, gestione area riservata, modifica password di accesso). **Ogni utente** dovrà avere accesso alle seguenti funzionalità:

- Area privata a cui accedere tramite la coppia di credenziali username/-password.
- Operazioni di gestione account (registrazione, recupero e modifica password).

2.1.1 Utente Medico

Ogni **medico** dovrà inoltre avere accesso alle seguenti funzionalità:

- Gestione degli orari di ricevimento visite, con possibilità di aggiungere o rimuovere una fascia oraria selezionata.
- Visualizzazione delle visite mediche effettuate o prenotate per una data futura.
- Cancellazione di una visita medica o invio di eventuali messaggi al paziente.
- Caricamento allegati riferiti ad una prenotazione (esempio: eventuali referti medici).
- Creazione di un token per permettere la registrazione tramite codice fiscale a nuovi medici della clinica, in accordo con quest'ultima.
- Creazione di nuovi reparti o modifica di quelli già registrati nel sistema.

2.1.2 Utente Paziente

Ogni **paziente** dovrà inoltre avere accesso alle seguenti funzionalità:

- Visualizzazione dei reparti della clinica disponibili a ricevere prenotazioni.
- Visualizzazione del calendario degli orari disponibili per un appuntamento in base al reparto selezionato. Per ogni fascia oraria il paziente dovrà avere anche accesso al nominativo del medico che lo visiterà.
- Visualizzazione delle visite mediche effettuate o prenotate per una data futura.

- Cancellazione di una visita medica o invio di eventuali messaggi al medico.
- Download degli allegati ricevuti dal medico che ha eseguito o eseguirà la visita prenotata.

2.2 Requisiti non funzionali

Si vuole ottenere un sistema in grado di:

- Essere accessibile e facilmente utilizzabile anche da un utente meno esperto, quindi non abbastanza ferrato all'uso delle tecnologie.
- Garantire sicurezza, riservatezza ed integrità dei dati tramite l'accesso ad ogni funzionalità attraverso un meccanismo di autenticazione.
- Garantire disponibilità 24h, per permettere lo svolgimento delle operazioni da parte di un utente in qualsiasi orario del giorno.
- Rispondere correttamente alle operazioni eseguite dagli utenti senza subire eccessivi rallentamenti, anche in seguito ad un'alta presenza di dati o ad un carico elevato dell'uso contemporaneo del sistema.

3 Progettazione

Il sistema è stato realizzato nella forma di un'applicazione web facendo uso delle tecnologie presenti nello **Stack MEAN**.

Quindi si è fatto uso di un server **Node.js** per la gestione del backend e del framework **Angular** per il lato client. Infine, le informazioni vengono memorizzati in una base di dati non relazionale: **MongoDB**.

Durante lo svolgimento del progetto si è fatto fede ai principi *KISS* (“Keep It Stupid, Simple”) e *DRY* (“Don’t Repeat Yourself”).

L'architettura del sistema è ascrivibile al modello REST (“Representational State Transfer”), nella quale le comunicazioni avvengono mediante HTTP e il sistema è di tipo “stateless”. L'applicazione può essere descritta mediante sette entità:

- User (*Utente*) : utilizzatore del sistema, che può essere di tipo:
 1. paziente;
 2. medico;
- Departments (*Settore*): Ogni utente di tipo medico possiede un settore di riferimento (in questo modo sarà più facile per il paziente “filtrare” le disponibilità);
- TimeTable (*Orari*): lista delle disponibilità (espresse come orario-giorno della settimana) dei medici
- Reservation (*Prenotazione*): Visita medica prenotata dal paziente ed eseguita dal medico;
- Document (*Documento*): referto di tipo documentale fornito dal medico all'interno della pagina della visita/prenotazione;
- Messages (*Messaggi*): Oggetti rappresentanti i messaggi condivisi tra paziente e medico all'interno della chat privata della prenotazione;
- Token : ogni medico per potersi iscrivere all'interno del sistema deve inserire un token (codice generato randomicamente da un amministratore e collegato univocamente al codice fiscale di un medico)

Di seguito uno schema riassuntivo delle relazioni tra le entità.

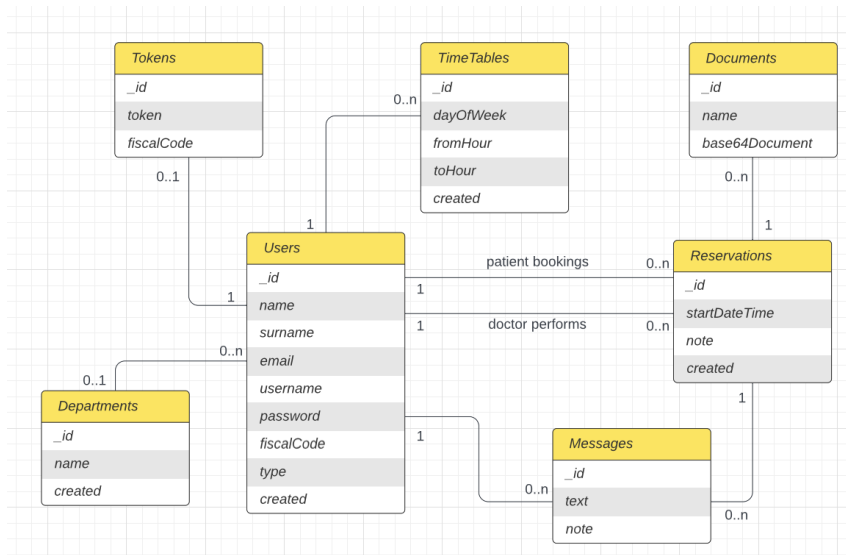


Figure 1: Schema Entità e relazioni

Per quanto riguarda la descrizione delle interazioni che gli utenti (Medici e Pazienti) possono eseguire all'interno del sistema, si é scelto di riassumere le azioni mediante due diagrammi di sequenza, il primo mostra l'operazione di selezione degli orari di lavoro da parte di un medico:

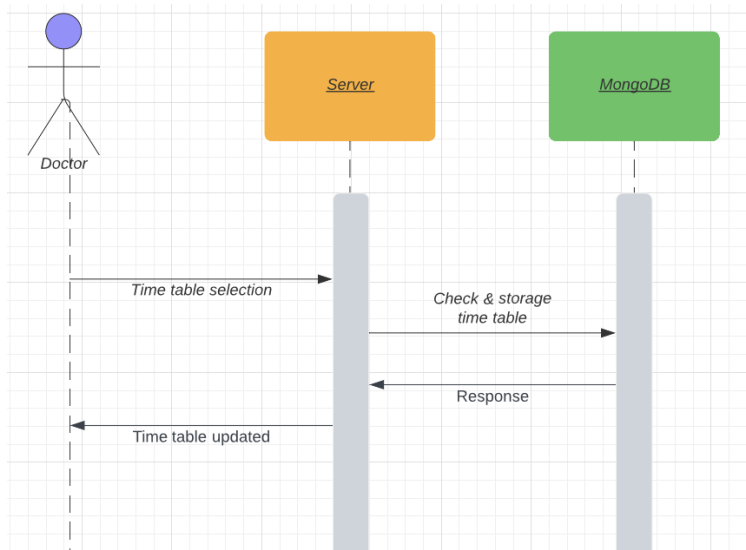


Figure 2: Sequence diagram scelta orari di lavoro

Il secondo diagramma invece mostra l'operazione di prenotazione di una visita medica da parte di un paziente in un orario disponibile di uno dei medici del reparto selezionato:

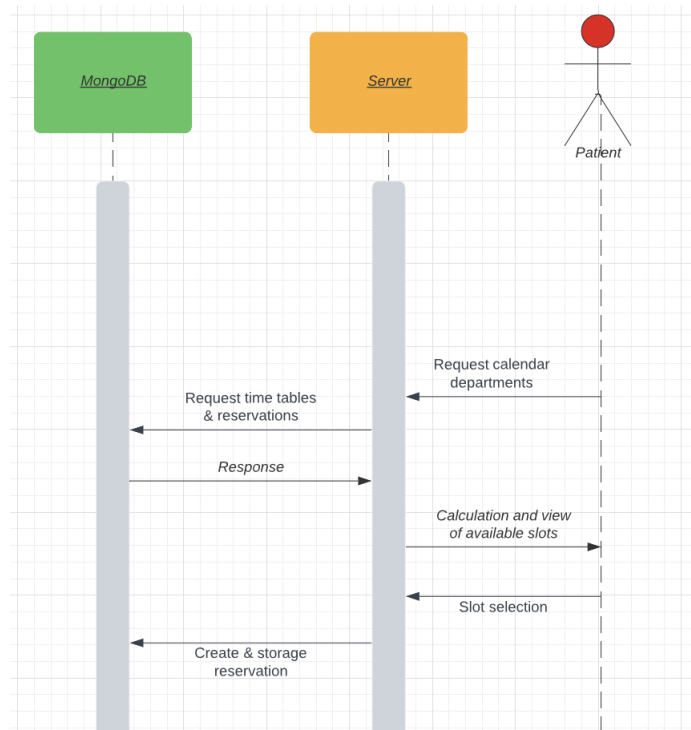


Figure 3: Sequence diagram scelta orario libero per prenotare una visita

4 Tecnologie Utilizzate

Il progetto, come sopra evidenziato, é stato sviluppato utilizzando il solution stack MEAN e le principali tecnologie utilizzate sono state:

- MongoDB e Mongoose
- Express
- Angular Js
- Node.js
- Socket.IO
- Docker

4.1 MongoDB e Mongoose

Lato backend, per gestire la persistenza dei dati, si é fatto uso di MongoDB come DBMS non relazionale e Mongoose.

MongoDB, anche grazie alla sua natura non relazionale, può essere classificato come database di tipo NoSQL, molto più permissivo in termine di struttura dei dati grazie all'utilizzo di documenti in stile JSON per la persistenza dei dati.

Mongoose é una libreria (scaricabile dal package manager di Node.js, ovvero “npm”) orientata agli oggetti e in grado di creare una connessione tra il framework Express e il database MongoDB.

Questa tecnologia mette a disposizione una funzione “Schema()” che produce oggetti su cui é possibile chiamare le classiche procedure di interfacciamento al db.

4.2 Express

Express é un server framework open source per applicazioni web sviluppate con Node.js, progettato per creare Web API e generare percorsi per le chiamate alle stesse.

4.3 Angular Js

Per la gestione del lato client si è scelto di usare il framework Angular in quanto molto flessibile agli usi dello sviluppatore ma allo stesso tempo semplice all'uso. Permette di gestire lo sviluppo ed il testing del client in maniera molto agile, secondo tecniche basate sull'architettura MVC: Model View Controller.

Per quanto riguarda la visualizzazione delle pagine web, il framework si occupa di leggere la pagina HTML che può presentare delle variabili al suo interno. Queste variabili saranno elaborate tramite il codice Typescript scritto nel component di riferimento della pagina HTML. Infine, per la comunicazione con il lato server, sarà possibile estrarre con facilità tutte le informazioni necessarie (e di conseguenza mostrarle con altrettanta semplicità all'utente che le richiede) tramite l'uso dei Service, tecnica che permette di incapsulare interazioni esterne (verso il server) per ottenere dei dati condividendo tale logica tra tutti i component dell'applicazione.

```
export class TimeTableService {  
  
  constructor(private _http: HttpClient) { }  
  
  fetchSlotsByDepartmentId(body:any, id: string | null): Observable<any> {  
    return this._http.post('http://localhost:4000/api/timeTables/departmentId/',  
      observe: 'body',  
      headers: new HttpHeaders().append('Content-type', 'application/json'))  
  });  
}
```

```

    }

    fetchSlotsByDoctorUserId(id: string | null): Observable<any> {
      return this._http.get('http://localhost:4000/api/timeTables/userId/' + id, {
        observe: 'body',
        headers: new HttpHeaders().append('Content-type', 'application/json')
      });
    }
    ... CONTINUA CON ALTRE FUNZIONI ...
  }
}

```

4.4 Node.js

Node.js è un runtime system, ovvero un software atto a fornire i servizi necessari all'esecuzione di un programma e non facente parte del sistema operativo, distribuito dalla Node.js Foundation per l'esecuzione di codice JavaScript lato server, costruito sul motore JavaScript V8 del web browser Google Chrome. Si tratta di un software open source, quindi ogni detentore è abilitato alla modifica, studio, utilizzo e redistribuzione del suo codice sorgente ed è multiplatforma. Node possiede un'architettura orientata agli eventi, che rende possibile l'ottimizzazione del throughput e della scalabilità in tutte le applicazioni web con molte operazioni di input/output (come programmi di comunicazione real time o browser game, ma non solo) grazie ad un design che ne permette l'asincronicità. Come già detto, JavaScript è tipicamente interpretato dalle componenti del browser ed utilizzato prettamente lato client, ma grazie a Node.js è possibile creare codice eseguibile lato server, al fine di produrre pagine web dinamiche prima che queste vengano inviate al browser dell'utente. Questo permette un'unificazione dello sviluppo di applicazioni web intorno a JavaScript (paradigma "JavaScript everywhere").

4.5 Socket.IO

Socket.IO è una libreria Javascript (scaricabile sempre da npm) orientata agli eventi, che permette la comunicazione in tempo reale di applicazioni web, mettendo a disposizione un "canale" bidirezionale tra client e server, dando quindi la possibilità a quest'ultimo di inviare dati anche in assenza di una chiamata lato frontend.

Meccanismo interessante (e utilizzato all'interno del progetto) sono le "rooms", ovvero gruppi di client all'interno dei quali è possibile inviare comunicazioni di tipo broadcast solo ai partecipanti; nello specifico si è fatto uso di questa tecnologia per implementare una chat "real time" tra paziente e medico.

4.6 Docker

Docker é una piattaforma software che fornisce la possibilità di creare, scaricare ed eseguire unità di codice sotto forma di "immagini". Le immagini possono essere eseguite molteplici volte (anche in parallelo) e ogni istanza é chiamata "container".

All'interno del progetto é stata utilizzata la funzionalità "docker compose" che permette di creare "bundle" di container che vengono eseguiti all'unisono pur proveniendo da immagini distinte (si tratta infatti della modalità di deploy scelta per questa WebApp); nello specifico vengono eseguiti in maniera contemporanea ed automatica 3 containers:

- Frontend - Angular Js
- Backend - Node Js
- Database - MongoDB

5 Design in dettaglio

5.1 Backend

Il backend é stato sviluppato attraverso l'uso delle tecnologie Express, Mongoose e Socket.IO.

I controllers sono rappresentati da classi javascript che espongono API per interagire con il backend, nello specifico sono presenti operazioni di tipo CRUD (Create, Read, Update, Delete) e sfruttano I Models, ovvero classi javascript che esportano Mongoose Schemas (rappresentazioni della struttura delle entità). Nel progetto si é mantenuta ove possibile una struttura "1 a 1" per ogni entità, quindi sono presenti sette file di controller, sette di model e sette di routes.

I file di routes utilizzano Express per fornire path e metodi specifici per richiamare le web Api esposte dai controller.

L'entry point del backend ("./server" nei file di progetto) é il file "app.js", che fa uso di Express, Mongoose, Socket.IO e dei file di route sopracitati. Socket.IO é utilizzato all'interno del progetto per rendere possibile la sincronizzazione in tempo reale della chat di una prenotazione.

5.2 Base Dati

Essendo MongoDB un database di tipo non relazionale, le informazioni sono memorizzate sotto la forma di documenti; facili sia da leggere che da scrivere. Per salvare tutto il necessario al corretto funzionamento del sistema sono stati realizzati **sette** tipi di documenti.

- **Users:** memorizza le informazioni di ogni utente i propri dati anagrafici, il ruolo con cui è stato registrato, l'ID del reparto di riferimento (nel caso di un utente di tipo Medico) ed infine username e password per effettuare l'accesso.
- **Departments:** memorizza i nominativi di tutti i reparti della clinica.
- **TimeTables:** memorizza le fasce orarie settimanali (scelte dal medico associato al record) in cui si è stata data disponibilità a ricevere pazienti.
- **Reservations:** memorizza le informazioni sulla prenotazione effettuata da un paziente come medico assegnato, note, data e orario dell'appuntamento.
- **Documents:** memorizza gli allegati caricati dai medici, ognuno associato ad una singola prenotazione.
- **Messages:** memorizza i messaggi scambiati tra medico e paziente di una determinata prenotazione.
- **Tokens:** memorizza il codice fiscale e un token (codice di dieci caratteri) di un medico non ancora presente nel sistema a cui è stato dato il permesso da parte della clinica di procedere alla registrazione.

5.3 Frontend

Per quanto riguarda il frontend, Angular ha permesso di realizzare una interfaccia semplice e intuitiva in maniera molto rapida in fase di progettazione. Il design di presenta con due parti statiche: l'header che diventa dinamico solo in fase di login, personalizzandosi in base al tipo di utente loggato; ed il footer completamente statico.

Il corpo centrale della pagina è invece dinamico e mostra le informazioni in base alla url in cui si trova l'utente. Per gestire tali informazioni è stata costruita una **tabella di routing** che reindirizza automaticamente il browser "svegliando" il component di riferimento.

Ogni component interroga il server in base alle informazioni di cui ha bisogno usando dei **Service**, che avranno quindi il compito di estrarre i dati predefiniti nel database usando le interrogazioni messe a disposizione dalle API del server. Possiamo quindi dividere l'applicazione in tre macro-strutture, descritte di seguito.

5.3.1 Login

Appena l'utente apre l'applicazione web si trova davanti alla schermata di login, in cui deve inserire username e password con cui ha eseguito l'iscrizione al sistema. Se le credenziali saranno corrette il sistema riconoscerà l'utente e procederà all'autenticazione come Medico o Paziente, altrimenti mostrerà un messaggio di errore.

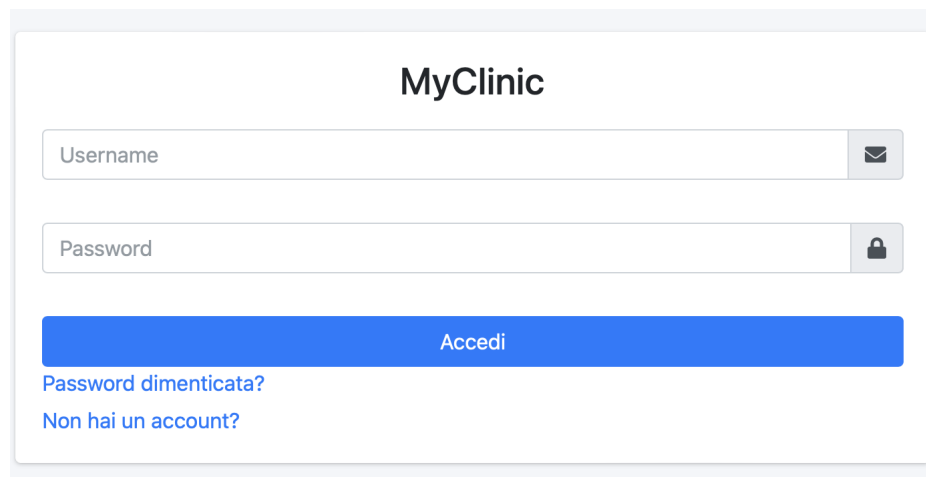
The image shows a login form for 'MyClinic'. At the top, the text 'MyClinic' is centered. Below it, there are two input fields: 'Username' and 'Password'. The 'Username' field has an envelope icon on the right, and the 'Password' field has a lock icon. Below these fields is a blue button labeled 'Accedi'. Under the button, there are two links: 'Password dimenticata?' and 'Non hai un account?'. The entire form is enclosed in a light gray border.

Figure 4: Login

5.3.2 Sign Up

Se l'utente non ha ancora effettuato la registrazione all'applicazione, potrà effettuarla mediante l'apposita schermata, accessibile da un link posizionato in alto a

sinistra della schermata di Login. La schermata richiederà all'utente tutti i dati anagrafici necessari, inoltre, se l'utente dichiara di essere un medico, un pop-up richiederà l'inserimento del token identificativo (collegato in maniera univoca al proprio codice fiscale) creato precedentemente da un amministratore della clinica ed il reparto di riferimento del medico. L'interfaccia presenta eventuali errori all'utente mediante un banner posizionato sopra ai campi compilabili.



The image shows a web form titled "Inserisci un token valido" (Insert a valid token) with a close button (X) in the top right corner. Below the title, there is a section labeled "Token" containing a text input field with the value "biH9PrOH9y". Below this, there is a section labeled "Seleziona il tuo reparto" (Select your department) with a dropdown menu showing "Cardiologia". At the bottom right of the form, there are two buttons: "Annulla" (Cancel) and "Salva" (Save).

Figure 5: Richiesta del token per la registrazione di un medico

5.3.3 Application Layout

Una volta autenticato, l'utente potrà interagire con l'applicazione, che presenta una struttura composta principalmente da tre componenti:

1. Una **toolbar**, contenente un menu che può variare in base al tipo di utente loggato.
2. Il **corpo centrale**, contenente le informazioni contestuali alla schermata selezionata.
3. Un **footer**.

Area Riservata Cliccando sul proprio username (posto nella parte destra della toolbar superiore) è possibile accedere alla schermata "Area Riservata", dove l'utente può visionare i propri dati (quali Nome, Cognome, Username, Email e Codice Fiscale), oltre a poter modificare la password associata al proprio account.

Area Riservata Home / Area Riservata

Dati personali

Nome: Cognome:

Username: Email: Codice fiscale:

Aggiorna password

Password attuale: Nuova password:

[Aggiorna password](#)

Figure 6: Area riservata

Orari Medici Ogni medico, per poter dare la possibilità ai pazienti di prenotarsi alle visite, deve esporre i propri orari di disponibilità; per farlo é necessario accedere alla schermata "Orario Di Lavoro" cliccando la voce "Modifica le tue disponibilità" posta nella toolbar superiore. Gli orari sono espressi come "giorno", "da ora" e "a ora". (il sistema userà questi dati per generare gli "slots" di prenotazione).

Orari di lavoro Home / Orari di lavoro

Giorno della settimana	Orario di inizio	Orario di fine	
Lunedì	15:00	19:00	
Martedì	16:00	19:00	

« Previous 1 Next »

[Aggiungi disponibilità](#)

Figure 7: Orari di lavoro scelti dal medico

Prenotare Una Visita Per prenotarsi ad una visita, un utente (di tipo "paziente") deve accedere alla lista delle disponibilità; in primis é necessario selezionare la voce di menu "Prenota una visita medica", per poi scegliere il reparto desiderato. A questo punto il sistema mostra all'utente una schermata calendario, dove, per ogni giorno, sono presenti pallini verdi (se lo slot é disponibile) o rossi (se lo slot é già occupato da un'altra prenotazione).

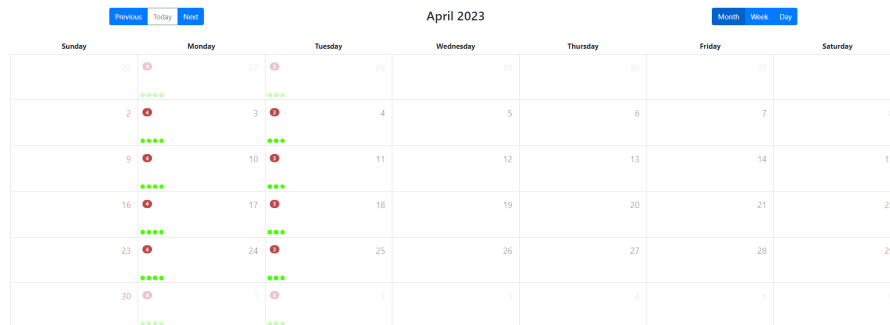


Figure 8: Calendario orari disponibili di un reparto

Dettaglio Visita Cliccando sulla voce "Gestisci le tue visite", gli utenti potranno accedere alla lista delle prenotazioni effettuate, e poter decidere di:

- Eliminare la prenotazione;
- Visionare/Caricare i documenti (o referti medici, modificabili solo da un medico);
- Visionare/Modificare gli aggiornamenti;
- Accedere ad una chat, in tempo reale, tra medico e paziente.

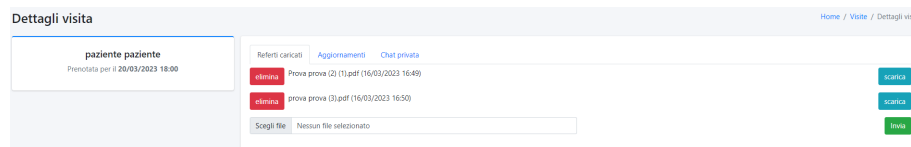


Figure 9: Dettaglio visita medica

Schermata Admin Ogni medico può accedere alla schermata "Area Amministratore" (cliccando sempre l'apposita voce di menù posta nella parte destra della toolbar superiore), dove può:

- Aggiungere e rimuovere un reparto (quest'ultima azione è possibile solo se nessun medico vi appartiene)
- Aggiungere Tickets (fondamentali per permettere ad un collega medico di iscriversi alla piattaforma)

Figure 10: Area amministratore

5.4 Interazione componenti sistema distribuito

Il frontend, mediante docker, viene esposto alla porta 4200; quando si effettua un'operazione che richiede l'intervento del server (e quindi della base di dati), viene fatta una chiamata http (di tipo GET, POST, PATCH o DELETE) attraverso la porta 4000 dal componente "&.service.ts" (dove al posto di "&" s'intende il nome dell'entità in gioco, ad esempio "user.service.ts" oppure "timeTable.service.ts").

Il server, che espone appunto la porta 4000, riceve la chiamata dall'"entry point" (app.js) che, grazie ai metodi esposti nelle componenti "&Routes.js", richiama la funzione desiderata del componente "&Api.js" (che utilizza la struttura delle entità esposte in "&Model.js").

Infine, la funzione sopracitata esegue una chiamata al database (sempre grazie a docker) alla porta 27017 e restituisce la risposta al chiamante.

Si specifica che tutte le interazioni appena descritte avvengono mediante meccanismi di "await" (subscribe nel caso di AngularJS) per permettere di eseguire operazioni solo previa risposta da parte del database.

6 Codice - aspetti salienti

In questa sezione verranno proposte e spiegate alcune parti di codice che si ritengono significative all'interno del progetto.

6.1 Algoritmo Slots

Si ritiene che l' "algoritmo degli slots", risiedente all'interno del backend, sia di notevole importanza all'interno del sistema, in quanto si tratta del codice che fornisce al paziente tutti i possibili orari (gli slots) per cui prenotare una visita. In fase di analisi si è scelto di non creare queste entità all'interno del database, dato il loro numero cospicuo e al fatto che presentano una grande variabilità

nel tempo; essi dipendono infatti sia dalle visite già prenotate che dagli orari dei medici (per reparto), che possono essere cambiate a piacimento da questi ultimi.

```
static async fetchSlotsByDepartmentId(req, res) {
  const body = req.body;
  const departmentId = req.params.departmentId
  let slots = new Array()
  try {
    const doctors = await User.find()
      .where("departmentId", departmentId)
    const timeTables = await TimeTable.find()
    const reservations = await Reservation.find()

    for (const doctor of doctors) {
      let cycleDate = new Date(body.fromDate)
      cycleDate.setHours(0, 0, 0);
      let toDateFilter = new Date(body.toDate)
      toDateFilter.setHours(0, 0, 0);

      while (cycleDate <= toDateFilter) {

        for (const timeTable of timeTables) {

          if (cycleDate.getDay() === timeTable.dayOfWeek
            && timeTable.doctorId === doctor._id) {

            const fromHour = parseInt(timeTable.fromHour
              .substring(0, 2))
            const toHour = parseInt(timeTable.toHour
              .substring(0, 2))
            cycleDate.setHours(fromHour)

            while (cycleDate.getHours() < toHour) {

              const slot = {}
              slot.dateTime = new Date(cycleDate)
              slot.doctorId = doctor._id
              slot.doctorName =
                doctor.name + " " + doctor.surname
              slot.reserved = false
              for (const reservation of reservations) {
                if (new Date(reservation.startDateTime)
                  .getTime() === cycleDate.getTime()
                  && reservation.doctorId === doctor._id){
                  slot.reserved = true
                }
              }
            }
          }
        }
      }
    }
  }
}
```

```

                                break
                            }
                        }
                    slots.push(slot)
                    cycleDate.setHours(cycleDate.getHours() + 1)
                }
            }
        }
        cycleDate.setDate(cycleDate.getDate() + 1)
    }
    res.status(200).json(slots)
} catch (error) {
    res.status(404).json({ message: error.message })
}
}

```

6.2 Gestione Documenti

I documenti (referti medici relativi ad una visita) vengono salvati all'interno del database sotto forma di attributi (formato Stringa) del relativo oggetto; per poter effettuare questa operazione viene quindi convertito il documento in formato "Base64" per poterlo inviare al server (l'operazione inversa viene eseguita per poter effettuare il download).

In fase di analisi si é scelto di creare, all'interno del database, una "tabella" solo per gli oggetti "documento", per permettere di massimizzare le performance: in questo modo, quando viene richiesta una prenotazione, essa presenterá solo un campo (di tipo array) contenente i riferimenti e i nomi dei file ad essa associati. Solo al momento del download questi documenti vengono effettivamente richiesti al server/database. Questo approccio, seppur non il migliore dal punto di vista prestazionale (a causa della mole di risorse necessarie per convertire il file e alla limitata possibilità di dimensione dello stesso) permette agli utenti di caricare qualunque tipo di file (siano essi immagini, pdf o file di testo), poiché la conversione/deconversione dei file avviene con un meccanismo che prescinde dalla natura del file stesso.

```

uploadDocument() {
    if (!this.docForm.valid) return;
    const reader = new FileReader();
    reader.readAsDataURL(this.document);
    reader.onload = () => {
        var b64String = reader.result as string

        const document = {
            'base64Document': b64String,
            'reservationId': idReservation,

```

```

        'documentName': this.document.name
    }

    this._document.createDocument(document).subscribe(res => {
        location.reload()
    });
};
}

downloadDocument(documentId: any) {
    this._document.fetchDocumentById(documentId).subscribe(res => {
        let link = document.createElement('a');
        link.href = res.base64Document;
        link.download = res.documentName;
        link.click();
    })
}

```

6.3 Visualizzazione calendario orari di lavoro di un reparto

Un esempio rilevante lato client è l'uso dei Service, in questo caso per ottenere l'orario delle disponibilità dei medici di un determinato reparto.

Il codice richiede tutti gli slot di un reparto (elaborati dall'algoritmo lato server mostrato nel paragrafo 6.1) partendo dalla prima data utile fino ai successivi 18 mesi. Una volta ottenute le informazioni queste vengono incapsulate sotto forma di JSON per essere successivamente usate dalla pagina HTML. Tra i campi della variabile **events** sono presenti data e ora dello slot, nome del medico, ed il colore di visualizzazione sul calendario (verde se lo slot è classificato dall'algoritmo come libero, rosso se invece è già prenotato). Analogamente, anche gli altri component sfruttano le funzionalità messe a disposizione dai vari service per estrarre i dati necessari alla corretta visualizzazione delle informazioni.

```

this._timeTable.fetchSlotsByDepartmentId({
    "fromDate": new Date(new Date().getFullYear(),
        new Date().getMonth(), new Date().getDate()+2),
    "toDate": new Date(new Date().getFullYear(),
        new Date().getMonth() + 18, 0)
}, this.route.snapshot.paramMap.get('id')).subscribe(
    res => {
        for (let i = 0; i < res.length; i++) {
            let dateTime = this.formatDateTime(res[i].dateTime), color;
            if (dateTime !== null) {
                if (res[i].reserved)
                    color = colors.red;
                else
                    color = colors.green;
            }
        }
    }
)

```

```

        this.events.push({
            start: parse(dateTime, "dd/MM/yyyy HH:mm",
                new Date()),
            title: "Doc. " + res[i].doctorName + " ore " +
                this.formatTime(dateTime),
            color: color,
            actions: this.actions,
            id: res[i].doctorId
        });
    }
}
);

```

6.4 Socket.io

Ulteriore frammento significativo del codice risiede nella gestione delle comunicazioni di Socket.IO, che permettono di sincronizzare in tempo reale i dati visualizzati dagli utenti su più istanze dell'applicazione: Alla selezione di una prenotazione, l'utente viene inserito in una room identificata dall'id della prenotazione stessa. All'invio di un messaggio nella chat della prenotazione, il frontend emette un messaggio verso backend; il backend invia quindi un messaggio broadcast a tutte le istanze dell'applicazione aperte ed iscritte alla room sopracitata (tranne quella in cui è avvenuta l'operazione). Questo messaggio viene quindi intercettato dal frontend, che scatena l'aggiornamento dei messaggi. In questo caso si omette la trascrizione del codice poiché le operazioni sopracitate si scatenano in più punti dell'applicazione, e il codice sarebbe di difficile comprensione.

7 Testing

Il corretto funzionamento dell'applicazione é stata testata sui principali Browser (Chrome, Firefox, Edge e Safari), soprattutto per quanto riguarda la reattività del sistema.

Per testare le API e le funzionalità di Socket.IO é stata utilizzata l'applicazione desktop “Postman”, che permette di simulare chiamate http, fornendo un'interfaccia grafica semplice e intuitiva; a tal proposito, all'interno della root di progetto (scaricabile al link fornito nella sezione ”deployment”) é presente una Postman-Collection per poter simulare e testare tutte le chiamate che é possibile effettuare al backend.

8 Deployment

8.1 Avvio dell'applicazione

La messa in funzione dell'applicativo è possibile grazie alle funzionalità messe a disposizione da Docker, nello specifico si farà uso di Docker compose:

- Clonare il repository al link: MyClinic
- Aprire un terminale nella root del progetto ed eseguire il comando: `' docker-compose up --build '`
- Accedere ad un browser web all'indirizzo: Localhost:4200

8.2 Fase di registrazione

Una volta avviata l'applicazione sarà possibile registrarsi come paziente compilando normalmente i campi richiesti dal form di registrazione.

Per la registrazione come medico bisognerà prima associare un token al codice fiscale dell'utente che si vuole registrare. Per fare ciò è possibile accedere all'area amministratori tramite la coppia di credenziali seguente:

- username: admin
- password: admin

E' fondamentale creare, dalla stessa pagina dell'admin, almeno un reparto nel quale il medico potrà iscriversi nel momento della sua registrazione.

Una volta effettuato l'accesso e creato un token si consiglia di copiarlo per poi incollarlo in fase di registrazione del medico con il codice fiscale associato a tale token.

Si sottolinea che quando il primo medico avrà effettuato la registrazione usando il suo token associato non sarà più possibile accedere al sistema con le credenziali admin. L'accesso all'area amministratore resterà disponibile all'interno del profilo del medico appena registrato ed anche a tutti gli eventuali successivi medici.

Questa scelta implementativa ha lo scopo di evitare accessi indesiderati una volta che l'area amministrativa di una clinica ha proceduto a fornire i token ai suoi medici ed almeno uno di essi ha proceduto alla registrazione.

9 Conclusioni

Il software è completamente funzionante in tutte le sue funzioni, rispettando così i requisiti fissati inizialmente. Il progetto, dato anche il limitato tempo di progettazione e sviluppo, si presenta sì completo rispetto all'idea iniziale, ma aperto dal punto di vista di implementazioni e aggiunte future, possibili anche grazie ad un'ingegnerizzazione modulare delle sue componenti. Un esempio di possibile (e plausibile) evoluzione potrebbe essere quella di dare la possibilità al medico di scegliere la dimensione degli slots di prenotazione (al momento fissa ad 1 ora), oltre a poter selezionare orari "non tondi" (che quindi presenta un minutaggio, oltre al dato dell'ora). Ci si ritiene soddisfatti del lavoro svolto dal sistema, sia termini di funzioni che in termini di efficienza.

References

- [1] [Node] sito ufficiale Node : <https://nodejs.org/en/>
- [2] [Vue] sito ufficiale AngularJs : <https://angularjs.org/>
- [3] [Mongoose] sito ufficiale Mongoose : <https://mongoosejs.com/>
- [4] [Socket.IO] sito ufficiale Socket.IO : <https://socket.io/>
- [5] [Docker] sito ufficiale Docker : <https://www.docker.com/>