UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

DIPARTIMENTO DI INFORMATICA

CORSO DI LAUREA IN

Informatica e Tecnologie per la Produzione del Software

TESI DI LAUREA

IN

SISTEMI COOPERATIVI

PROGETTAZIONE E SVILUPPO DI UN SOFTWARE DESKTOP MULTIPIATTAFORMA BASATO SU TECNOLOGIE WEB PER IL CONTROLLO REMOTO E MONITORAGGIO DI PARAMETRI EMATOLOGICI

RELATORE

Prof. Giovanni Dimauro

LAUREANDO

Alessandro Zermo

ANNO ACCADEMICO 2016 / 2017

Possiamo vedere solo poco davanti a noi,
ma possiamo vedere tante cose che bisogna fare.

Alan Turing

ABSTRACT

Questo lavoro di tesi si inserisce nel contesto della progettazione e realizzazione di un sistema non invasivo per la stima del livello di emoglobina mediante l'analisi del colore della congiuntiva palpebrale.

Il software realizzato, di nome DeskEmo, permette al medico curante di ricevere ed archiviare le informazioni relative al colore della congiuntiva palpebrale dei propri pazienti semplificando e velocizzando le operazioni di diagnosi. Queste informazioni viaggiano attraverso canali sicuri che garantiscono autenticazione, integrità dei dati e cifratura operando sul protocollo standard per la trasmissione via internet di e-mail. Le informazioni sensibili, una volta giunte sul dispositivo del medico curante, rimangono archiviate in locale per il tempo necessario al medico o allo specialista di verificare lo stato e la salute del paziente. DeskEmo utilizza un’organizzazione efficiente dei dati raggruppando i report per paziente e consentendo una visualizzazione dello storico dei vari parametri ematologici così da semplificare, velocizzare e rendere più efficaci eventuali diagnosi. I dati storici sono archiviati senza riferimenti a dati personali, al fine della garanzia della privacy e della sicurezza.

Il funzionamento e l’interfaccia di DeskEmo si basano interamente sulle tecnologie tipiche del web abbinate ad un framework open-source che permette l’utilizzo di queste tecnologie in ambito desktop.

INDICE

1. EMOGLOBINA E ANEMIA

L'emoglobina è una proteina, indicata solitamente con il simbolo Hb, costituente principale dei globuli rossi, adibita principalmente al trasporto dell’ossigeno dai polmoni ai tessuti. Il suo valore è l’indicatore più affidabile di anemia.

Valori bassi di emoglobina (anemia) sono comuni a molte malattie, quello che le accomuna è di norma il fatto che il corpo produca meno globuli rossi oppure li distrugga più velocemente di quanto possano essere prodotti, o ancora se si verificano perdite di sangue. Le cause possono quindi essere varie:

- carenza di ferro
- carenza di vitamina B12
- carenza di folati
- sanguinamento
- tumori che colpiscono il midollo osseo, come la leucemia
- malattie renali
- malattie del fegato
- ipotiroidismo
- talassemia (una malattia genetica che provoca bassi valori di emoglobina e globuli rossi)

L’anemia è un problema con conseguenze sia sulla salute dell’uomo sia sullo sviluppo socio-economico in quanto colpisce più di 1,5 miliardi di persone, ovvero circa il 30% della popolazione mondiale.

I valori di soglia minima, che indicano la presenza di anemia variano con l’età, il sesso e lo stato psicologico del paziente.

Tabella 1 - Soglie di emoglobina per definire l'anemia

Età/Sesso	**Hb normale**
Bambini (0.5 – 5 anni)	11.0
Bambini (5-12 anni)	11.5
Adolescenti (12-15 anni)	12.0
Donne, non gravide (>15 anni)	12.0
Donne, in gravidanza	11.0
Uomo (>15 anni)	13.0

Essenzialmente una forte anemia compromette la disponibilità di ossigeno ai tessuti causando danni agli organi vitali ed una potenziale condizione pericolosa per la vita. I pazienti affetti da questo problema devono ricevere trasfusioni di sangue continue ed i medici hanno bisogno di conoscere il livello di emoglobina per decidere se effettuare la trasfusione oppure no. Pertanto, dato che il livello varia ogni giorno, è necessario fare test frequentemente con ingenti costi e disagio nel paziente.

La pratica medica corrente per diagnosticare l'anemia richiede la determinazione invasiva del livello di emoglobina nel sangue, ovvero un prelievo di sangue effettuato da un infermiere o medico con alti costi di manodopera e strumentazione, una procedura che richiede tempo e aumenta il rischio della trasmissione di infezioni trasmettibili per via ematica.

La pratica standard da utilizzare per individuare il livello di emoglobina è la flebotomia con la determinazione del livello di emoglobina in laboratorio. Attualmente però la tecnica più utilizzata è la citometria a flusso, ossia una tecnica laser che in biologia permette il conteggio, la separazione e il rilevamento delle cellule in modo di poter effettuare un'analisi simultanea dei parametri sia fisici che chimici su migliaia di cellule al secondo.

Queste tecniche, sebbene rapide, affidabili e facilmente accessibili nei paesi sviluppati, risultano difficili da praticare nelle grandi regioni del mondo in via di sviluppo in cui l'accesso alle tecnologie è limitato. Inoltre in queste regioni il livello di anemia è di gran lunga più alto rispetto ai paesi sviluppati. Per questi motivi la difficoltà nel diagnosticare l' anemia aumenta considerevolmente i rischi legati alla malattia, infatti ciò ha portato un aumento considerevole del tasso di mortalità dei bambini ricoverati in ospedale per anemia e si sono riscontrati alti rischi di salute per le donne incinte e i loro bambini se queste sono affette di anemia, infine l'anemia porta ad una diminuzione della forza fisica significativa per i lavoratori provocando così anche problemi dal punto di vista economico e sociale.

1.1. TECNICHE DI STIMA NON-INVASIVE

Un metodo non invasivo per individuare il livello di emoglobina nel sangue è l'analisi della congiuntiva. La presenza del pallore congiuntivale infatti, è stato usato come input per la determinazione del livello di emoglobina in

molti studi. In particolare Sheth ha individuato in uno suo studio un rapporto di verosimiglianza del 95% tra pazienti anemici e non, partendo da questo tipo di valutazione e considerando una misura di 90 g/L di concentrazione di emoglobina. Un altro studio di questo tipo ha evidenziato che la valutazione del livello di emoglobina tramite il pallore della congiuntiva è influenzata dall'esperienza dell'osservatore quindi è necessaria una controprova. Altri studi, invece hanno cercato di individuare la correlazione tra il pallore di altre parti del corpo, come quello del palmo, ed il livello di emoglobina dimostrando che la valutazione tramite il pallore congiuntivale è più accurata in quanto nel caso del palmo la sua pigmentazione è un fattore di ostacolo nell'osservazione. Tali risultati hanno richiesto la valutazione del pallore di diverse parti del corpo e anch'essi dimostrano che queste tecniche di valutazione del pallore non sono oggettive, in quanto fatte soggettivamente dai medici.

Negli ultimi anni gli studi che coinvolgono la valutazione di emoglobina per rilevare il livello di anemia sono incrementati e altri metodi alternativi per rilevare l'emoglobina nel sangue sono stati individuati, come quello per cui N. Tsumura analizzando l'immagine del volto riesce ad estrarre i componenti emoglobina e melanina con l'analisi delle componenti indipendenti.

Un primo metodo che correli il colore della palpebra e la concentrazione di emoglobina, usando un approccio non invasivo, è stato proposto da C. I. Sanchez-Carrillo. Il metodo analizzato in questo studio utilizza uno strumento colorimetrico non invasivo, progettato con tonalità simili a quelle osservate nella congiuntiva, pertanto confrontando i due colori si riesce a misurare il livello di emoglobina. I risultati ottenuti da Sanchez-Carrillo hanno portato buoni livelli di sensibilità e specificità, con la presenza di falsi negativi minimi per uno screening di valori di emoglobina inferiori o uguali a 13 g/dl. La semplicità dello strumento lo rende accessibile anche a

personale non specializzato per una pre-diagnosi che deve essere seguita da ulteriori diagnosi e cure. Importante è un approccio non invasivo per quei tipi di pazienti che non possono effettuare un prelievo di sangue; perciò grazie allo sviluppo della computer vision sono stati introdotti nuovi metodi.

In particolare Suner ha trovato un'alta correlazione tra la concentrazione di emoglobina calcolata e misurata, con l'uso di un'immagine digitale a colori della congiuntiva. Il suo lavoro è stato effettuato su una scheda di colore di grigio standard con un noto valore RGB, in modo da poter confrontare foto acquisite in diverse condizioni di luce. Utilizzando un software di valutazione automatica su un Personal Digital Assistant (PDA) e considerando il modello di colore RGB, Suner ha raggiunto una correlazione moderata fra il colore della congiuntiva e il livello di emoglobina.

Invece in un suo studio Kim utilizza la combinazione di un modello stocastico di propagazione di un quanto di luce nel tessuto palpebrale e la riflessione umana nel multistrato per studiare la concentrazione spettrale nella congiuntiva e la determinazione dell'emoglobina. I livelli di emoglobina ottenuti confrontati con quelli misurati in vitro hanno dimostrato che il metodo fornisce per l'86% stime di sensibilità per i casi di anemia clinicamente diagnosticati. Unico svantaggio dello studio è che richiede un'apparecchiatura molto complessa per cui non utilizzabile al di fuori di un contesto clinico.

Ancora Suner utilizza un metodo che correla in modo obiettivo il colore ottenuto dell'immagine della congiuntiva e l'emoglobina misurata, in cui si utilizza una carta fotografica standard con scala di grigi del 18% per il controllo delle differenze di luce nell'ambiente in modo da rendere il sistema distribuibile indipendentemente dalle condizioni. Da questo studio

sono stati esclusi pazienti con emorragia attiva, con iperbilirubinemia misurata o ittero visibile e con iposia o ipotensione in quanto per questi il colore della congiuntiva potrebbe essere alterato per diverse motivazioni, inoltre sono stati esclusi pazienti con traumi o infezioni all'occhio.

L'algoritmo utilizzato nel progetto di Suner dopo aver acquisito la foto dell'occhio con la palpebra inferiore abbassata richiede la selezione manuale dell'area della congiuntiva evitando le aree di riflessione della luce, per poi sottoporre le immagini ritagliate ad un'analisi in cui sono separati i componenti di rosso, verde e blu all'interno dell'immagine. Per cui ad ogni pixel nell'area selezionata sono assegnati tre valori uno per ogni strato di colore fra lo 0 e il 255. Questi valori insieme ai valori standard e alle costanti determinate in un processo iterativo fanno si che l'emoglobina predetta sia ottimizzata e si trovi la formula ottimale per poi applicarla la processo di analisi dell'immagine della congiuntiva.

Nel nostro lavoro abbiamo provato a valutare se è possibile prevedere il livello di emoglobina con l'utilizzo di un'immagine digitale della congiuntiva palpebrale, eliminando la necessità di una trasfusione di sangue. In particolare nel nostro lavoro ci siamo concentrati sull'acquisizione dell'immagine da analizzare. Partendo da un'immagine dell'occhio acquisita con una fotocamera, solitamente di un cellulare, abbiamo elaborato un algoritmo che riesca ad individuare automaticamente l'area della congiuntiva palpebrale che successivamente sarà analizzata in modo da poter individuare il livello di emoglobina nel sangue. Per riuscire a raggiungere il nostro obiettivo sono state utilizzate tecniche della Computer Vision. Ovvero un insieme di processi che mirano a creare un modello approssimato di mondo reale partendo da immagini bidimensionali o tridimensionali col fine di riprodurre la vista umana.

1.2. CONCETTI CHIAVE

Come proseguo dello studio “A novel approach to evaluate blood parameters using computer vision techniques”, anche questo lavoro si baserà sull’analisi della congiuntiva palpebrale.

La congiuntiva è una membrana mucosa, che ricopre il bulbo oculare e la parte interna delle palpebre; ha la funzione di proteggere il bulbo oculare, soprattutto la cornea (benché la sua faccia anteriore sia sprovvista del rivestimento congiuntivale), nonché di facilitare sia il suo scorrimento che quello delle palpebre nelle fasi di ammiccamento, mediante la secrezione della componente mucinica del film lacrimale.

Figura 1: Foto di un occhio con congiuntiva palpebrale in evidenza

2. HB METER

Lo scopo del nostro lavoro è stato quindi quello di trovare un modo alternativo e più rapido per la selezione dell'area della congiuntiva, area formata dai pixel d'interesse per la valutazione del colore.

Dopo aver realizzato l'impossibilità (momentanea) di riconoscere e delimitare questa regione in modo del tutto automatico, date le possibilità delle tecnologie odierne, si è cercato un modo per chiedere l'iterazione dell'utente per fare questo ed allo stesso tempo di fare in modo che questa iterazione fosse più intuitiva possibile, quindi si è puntato a due fattori chiave: semplicità e velocità.

Il sistema "A novel approach to evaluate blood parameters using computer vision techniques" prevede l'iterazione, su un'applicazione software per desktop, dove l'utente deve selezionare l'area che desidera analizzare con uno strumento di selezione rettangolare, questo porta ad una non precisissima selezione con una perdita di pixel importanti per la valutazione finale.

Questo problema è stato risolto combinando l'utilizzo di un algoritmo di image analisys, lo SLIC SuperPixels con cui si è risolto il problema della selezione non più rettangolare ma di forma sempre diversa in base alla somiglianza dei pixels adiacenti tra loro in abbinamento alle capacità dei moderni smartphone che offrono una superficie touch con cui è immediato e semplice interagire, anche con una elevata precisione. All'utente quindi verrà chiesto di selezionare le aree SuperPixels semplicemente disegnando una curva sulla regione d'interesse, un secondo algoritmo si occuperà di

selezionare i SuperPixels che coincidono con i punti della curva citata poc’anzi.

2.1. PANORAMICA DELL’APP

L’applicazione è stata sviluppata sia per la piattaforma iOS di proprietà di Apple, più precismanete per dispositivi iPhone, sia per i dispositivi dotati del sistema operativo Android (il più diffuso attualmente in ambito smartphone).

La prima view ad essere presentata all’utente è una view che innanzitutto mostra un breve tutorial, composto da quattro step che illustrano il funzionamento dell’applicazione.

Il primo step suggerisce di montare la lente (attraverso l’utilizzo di un prototipo hardware per l’acquisizione delle foto), questo step non è obbligatorio, in quanto nello step successivo sarà possibile selezionare una foto scattata in precedenza (a patto che questa sia stata scattata utilizzato l’apposita lente di acquisizione);

il secondo step invita a premere il pulsante “scegli foto”, l’iterazione con questo pulsante aprirà una nuova view che ci permetterà di scegliere la foto dal rullino fotografico del dispositivo (quindi una foto precedentemente scattata) oppure di scattare una nuova foto;

il terzo step mostra il metodo di selezione dell’area interessata che avviene tramite uno slide del dito sui SuperPixels più rilevanti, a discrezione dell’utente;

il quarto ed ultimo step, invita a premere il pulsante “analizza” per avviare l’analisi del colore dei pixel interessati dalla selezione precedente, a tal proposito va sottolineato che l’analisi è possibile solo se è stato selezionato almeno un SuperPixel in caso contrario il pulsante “analizza” viene disabilitato.

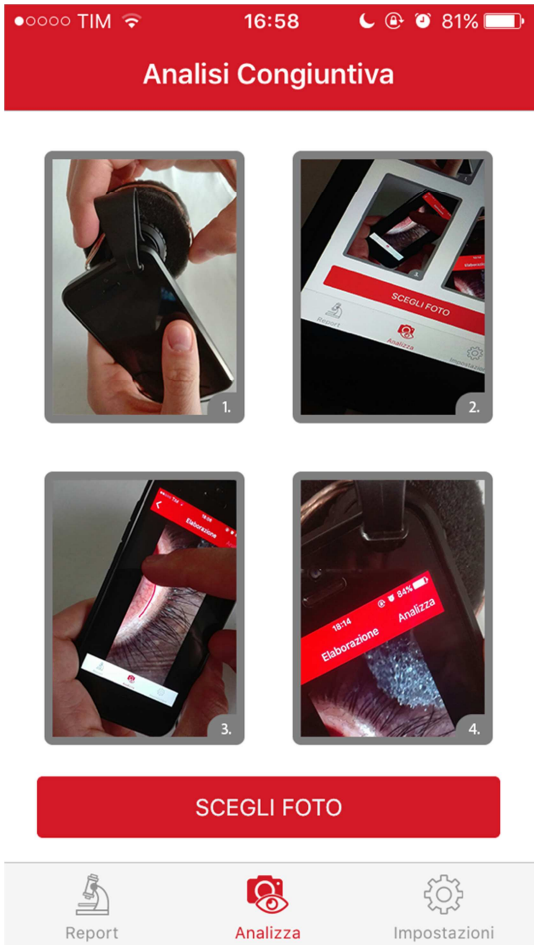

Figura 2: View iniziale dell’applicazione per iOS

Oltre alle view presentate sopra, il sistema in fase di analisi memorizza i dati d’interesse per mostrarli in un’apposita view composta da una TableView. Ogni cella di questa tabella è stata personalizzata implementando una sottoclasse della classe UITableCellView in modo da

visualizzare su ognuna di esse le informazioni di interesse che rappresentano il resoconto dell'analisi effettuata in passato, più precisamente queste informazioni sopra citate sono:

- Immagine della selezione effettuata
- Data e ora dell'analisi
- Valore della media del canale a*

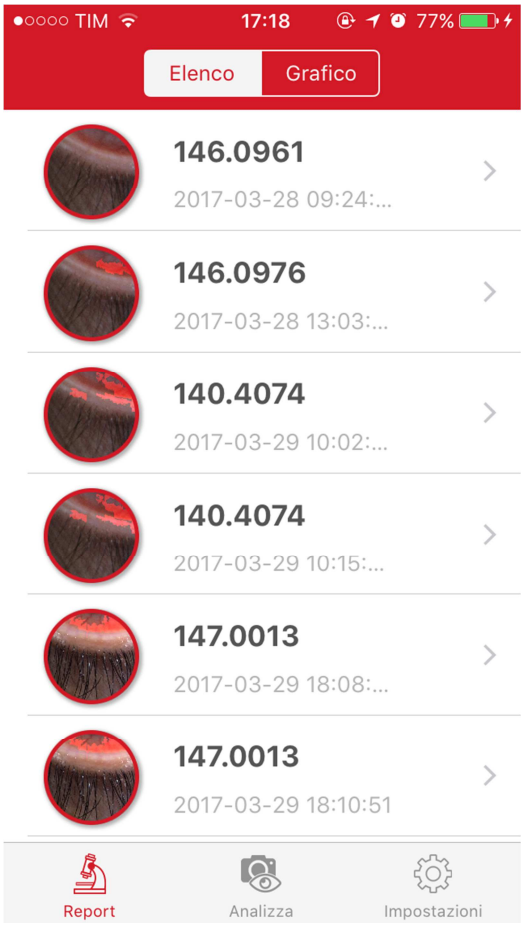

Figura 3: TableView per la gestione dello storico dei report

L'iterazione, tramite tap, di una di queste celle, da visibilità ad una nuova view, visivamente identica a quella di report dell'analisi effettuata dopo lo scatto della foto.

Il suo funzionamento però differisce da quest'ultima in quando invece di salvare i dati, li recupera dalla memoria interna al fine di presentarli all'utente.

Questa funzione permette, quindi, di tenere uno storico delle analisi effettuate, di prenderne visione ogni volta che si vuole ed infine, grazie a **DeskEmo**, progetto di questa tesi di laurea, di cui parleremo nei capitoli successivi, di permettere al medico curante di ricevere e consultare questi report.

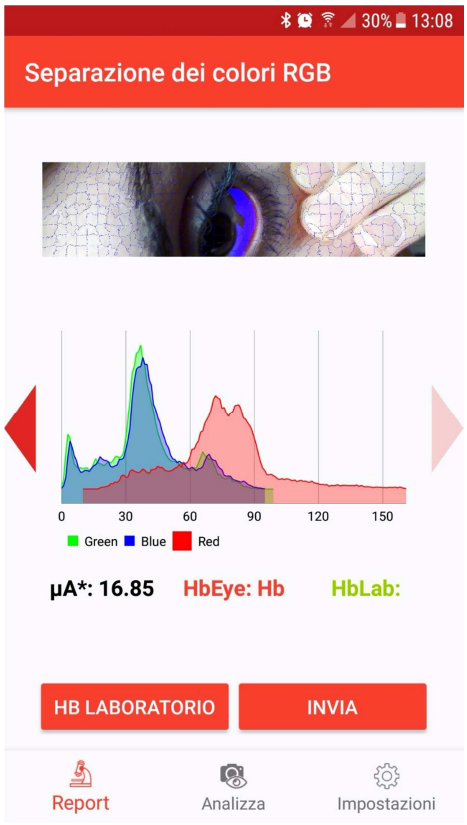

Figura 4: View di riepilogo dell'analisi (versione per Android), richiamata dallo storico

2.2. USO DEL DISPOSITIVO DI ACQUISIZIONE

Al paziente è chiesto semplicemente di abbassare la palpebra inferiore e di appoggiare il viso, più precisamente l'occhio al dispositivo, a questo punto

un pulsante a pressione chiuderà il circuito a cui sono collegati i diodi LEDs che si accenderanno.

Giunti in questa fase, il paziente da solo (con l'ausilio di uno specchio) o il medico dovrà scattare la foto (tramite la normale applicazione per le foto o utilizzato l'app per smartphone).

La foto acquisita è considerata valida se l'area della congiuntiva palpebrale è chiaramente visibile.

Figura 5: Dispositivo hardware per l'acquisizione delle foto

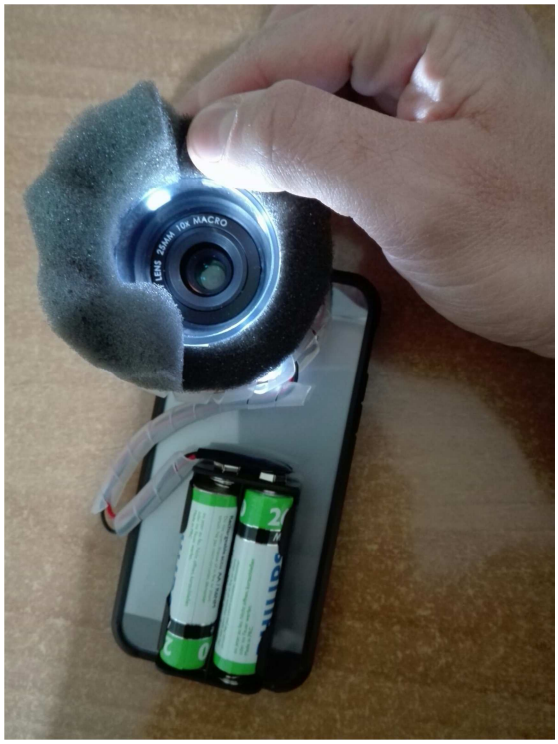

Figura 6: Dispositivo hardware per l’acquisizione delle foto con LEDs accesi

3. DESKEMO

DeskEmo permette al medico curante di ricevere ed archiviare le informazioni relative al colore della congiuntiva palpebrale dei propri pazienti semplificando e velocizzando le operazioni di diagnosi. Queste informazioni viaggiano attraverso canali sicuri che garantiscono autenticazione, integrità dei dati e cifratura operando sul protocollo standard per la trasmissione via internet di e-mail. Le informazioni sensibili, una volta giunte sul dispositivo del medico curante, rimangono archiviate in locale per il tempo necessario al medico o allo specialista di verificare lo stato e la salute del paziente. DeskEmo utilizza un'organizzazione efficiente dei dati raggruppando i report per paziente e consentendo una visualizzazione dello storico dei vari parametri ematologici così da semplificare, velocizzare e rendere più efficaci eventuali diagnosi. I dati storici sono archiviati senza riferimenti a dati personali, al fine della garanzia della privacy e della sicurezza.

Il funzionamento e l'interfaccia di DeskEmo si basano interamente sulle tecnologie tipiche del web abbinate ad un framework open-source che permette l'utilizzo di queste tecnologie in ambito desktop.

L'utilizzo delle tecnologie web ha come vantaggio principale quello di garantire la portabilità del software in modo che possa essere eseguito sui sistemi operativi desktop più diffusi. DeskEmo è stato pensato e progettato per essere un software out-of-the-box, non sarà necessario effettuare alcuna installazione o configurazione che possa richiedere l'aiuto di un esperto e pertanto il medico sarà in grado di operare in tutta autonomia. L'interfaccia richiama lo stesso stile grafico dell'app per smartphone in modo da garantire una certa familiarità ed è di tipo responsive, in grado di sfruttare

ed adattarsi appieno a qualsiasi dimensione dello schermo. Tutte queste caratteristiche rendono DeskEmo un software malleabile e capace in futuro di integrarsi in maniera del tutto naturale, data la sua progettazione basata sul web, con il RESP (Registro Elettronico Sanitario Personale).

3.1. MISSION

DeskEmo nasce dall'esigenza di poter garantire il monitoraggio dei parametri ematologici salvati sull'applicazione mobile da parte del medico curante oppure di un medico di laboratorio.

È innegabile però, che la maggior parte dei software nati con lo scopo di poter semplificare e velocizzare il lavoro di un medico finiscono esattamente nel produrre l'effetto contrario per via della loro complessità di configurazione, interfaccia poco intuitiva ed incompatibilità con hardware e software in dotazione presso lo studio medico o laboratorio di analisi.

DeskEmo pertanto fa della semplicità la sua forza, non richiede alcuna procedura particolare d'installazione, ha un'interfaccia grafica semplice e pulita che richiama lo stile utilizzato per l'app sui dispositivi mobili e cosa più importante, grazie ad un framework open-source, di cui daremo maggiori dettagli in seguito, è in grado di essere eseguita su qualsiasi sistema operativo per desktop; tra cui i più diffusi come Windows, macOS e Linux; secondo le statistiche stilate da NetMarketShare questi tre sistemi operativi raggiungono il 96,4% di diffusione. Pertanto possiamo certamente affermare che DeskEmo è grado di essere eseguito sul 96,4% dei sistemi operativi attualmente in circolazione.

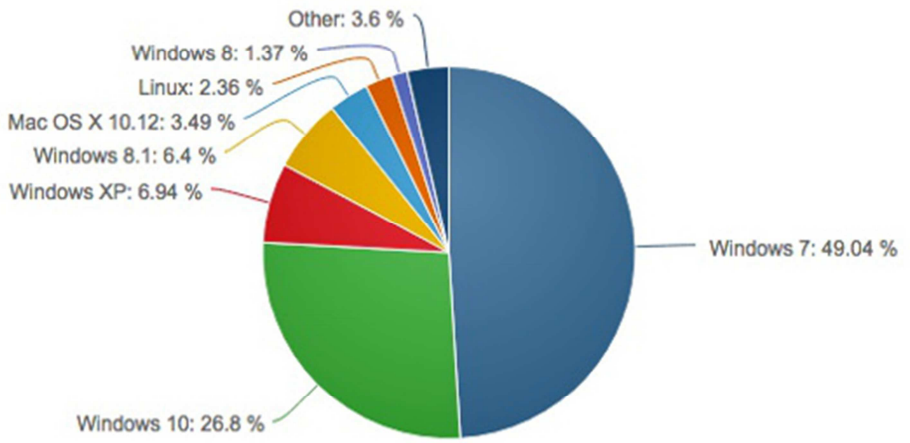

Figura 7: La “torta” dei sistemi operativi desktop di giugno 2017 secondo NetMarketShare,

3.2. TECNOLOGIE WEB ALLA BASE

La progettazione e l’intero sviluppo di DeskEmo si basano completamente sulle tecnologie tipiche del web nonostante DeskEmo sia stato pensato per essere un software puramente desktop. I vantaggi dell’utilizzo di queste tecnologie sono molteplici, a partire dalla maturità e dalla diffusione che ormai queste hanno raggiunto (la maggior parte dei servizi al giorno d’oggi viene offerta via web).

In particolar modo, le tecnologie principali adottate per lo sviluppo di DeskEmo sono le seguenti:

- Tecnologie alla base dell’interfaccia grafica:
  - HTML5
  - CSS
- Tecnologie alla base del funzionamento:
  - Javascript

3.2.1. HTML5

L'**HTML** (HyperText Markup Language) è il linguaggio utilizzato per descrivere e definire il contenuto di una pagina web in un formato ben strutturato. HTML è uno standard internazionale le cui specifiche sono manutenute dal W3C (World Wide Web Consortium). Sebbene le specifiche di HTML5 (l'ultima evoluzione degli standard che definiscono l'HTML) siano ancora tecnicamente in fase di sviluppo, è considerato uno "standard vivo", ed è essenzialmente una versione sempre attuale delle specifiche HTML.

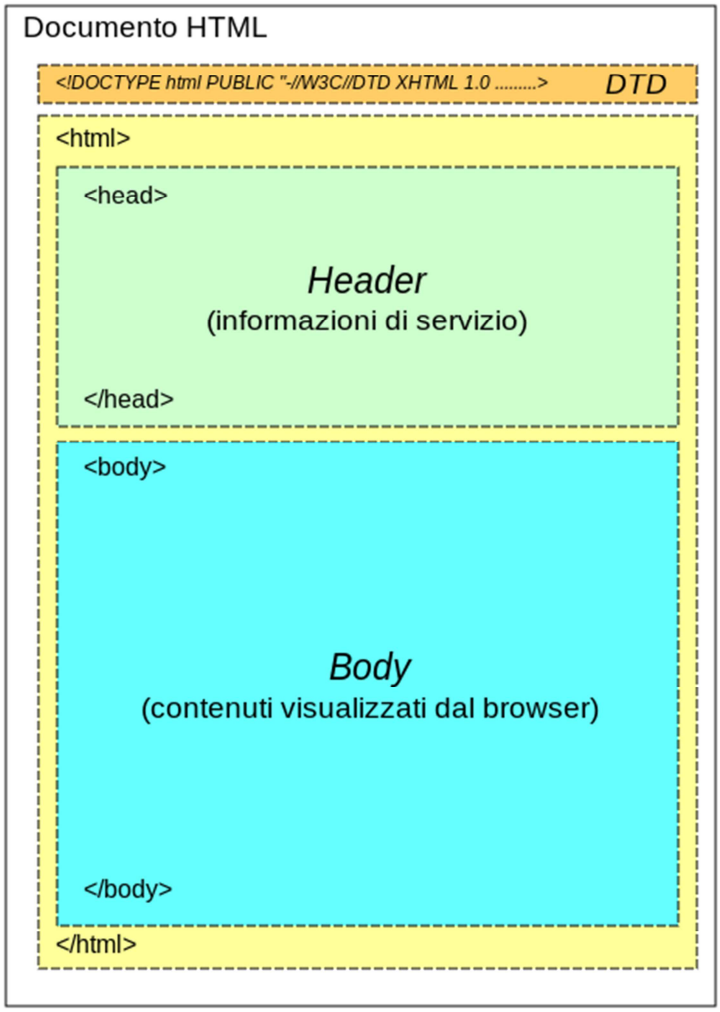

Figura 8: Struttura di un generico documento HTML

Il motivo principale che ha spinto il W3C e i suoi membri a sviluppare HTML5 è stata la necessità di fornire direttamente le funzionalità che in precedenza erano fruibili tramite estensioni proprietarie all'esterno dei browser, come Adobe Flash e simili. Un secondo obiettivo che gli sviluppatori si erano prefissati era quello di garantire una maggiore compatibilità tra i diversi browser, indipendentemente dalla piattaforma software utilizzata, e principalmente mirata all'espansione dei dispositivi mobili.

Le novità introdotte dall'HTML5 sono finalizzate soprattutto a migliorare il disaccoppiamento fra struttura, definita dal markup, caratteristiche di resa (tipo di carattere, colori, eccetera), definite dalle direttive di stile, e contenuti di una pagina web, definiti dal testo vero e proprio. Inoltre l'HTML5 prevede il supporto per la memorizzazione locale di grandi quantità di dati scaricati dal web browser, per consentire l'utilizzo di applicazioni basate su web (come per esempio le caselle di posta di Google o altri servizi analoghi) anche in assenza di collegamento a Internet.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Example<title>
5          <link rel="stylesheet" href="styl
6      </head>
7      <body>
8          <h1>
9              <a href="/">Header</a>
10         </h1>
11         <nav>
12             <a href="one/">One</a>
13             <a href="two/">Two</a>
14             <a href="three/">Three</a>
15         </nav>
```

Figura 9: Un esempio di codice HTML con sintassi evidenziata

3.2.2. CSS

Il **CSS** (Cascading Style Sheets, in italiano fogli di stile a cascata) è un linguaggio usato per definire la formattazione di documenti HTML, XHTML e XML. Le regole per comporre il CSS sono contenute in un insieme di direttive (Recommendations) emanate a partire dal 1996 dal W3C.

L'introduzione del CSS si è resa necessaria per separare i contenuti delle pagine HTML dalla loro formattazione e permettere una programmazione più chiara e facile da utilizzare, sia per gli autori delle pagine stesse sia per gli utenti, garantendo contemporaneamente anche il riutilizzo di codice ed una sua più facile manutenzione.

Un foglio di stile CSS è sintatticamente strutturato come una sequenza di regole, che sono coppie costituite da un selettore e un blocco di dichiarazioni, racchiuso tra parentesi graffe. Un selettore è un predicato che individua certi elementi del documento HTML; una dichiarazione, separata con un punto e virgola dalle altre, è a sua volta costituita da una proprietà, ovvero un tratto di stile (come il colore del testo) e un valore da assegnare a quest'ultimo (per esempio blu) separati dai due punti.

Esempio:

```
selettore1 {
  proprietà1: valore1;
  proprietà2: valore2;
}
selettore2 {
  proprietà3: valore3
}
```

3.2.3. JAVASCRIPT

JavaScript (spesso abbreviato in JS) è il linguaggio di programmazione utilizzato nei browser, il quale viene utilizzato per realizzare siti Web interattivi e applicazioni eseguibili in modo sicuro dal browser.

JavaScript è un linguaggio leggero formalizzato con una sintassi vicina a quella del linguaggio Java di Sun Microsystems (che nel 2010 è stata acquistata da Oracle), interpretato, funzionale ed orientato agli oggetti, conosciuto per lo più come linguaggio di script per pagine web, ma utilizzato in molti ambienti non-browser così come node.js (un runtime javascript utilizzato anche in DeskEmo sulla quale daremo maggiori dettagli in seguito).

Tali script, utilizzati dunque nella logica di presentazione, possono essere opportunamente inseriti in file HTML, in pagine JSP o in appositi file separati con estensione .js per poi essere richiamati nella logica di business. Ultimamente il suo campo di utilizzo è stato esteso alle cosiddette Hybrid App (app ibride, di cui DeskEmo ne fa anche parte), con le quali è possibile creare app per più sistemi operativi utilizzando un unico codice sorgente bastato appunto su JavaScript, HTML e CSS.

JavaScript è stato standardizzato per la prima volta il 1997 dalla ECMA con il nome ECMAScript. L'ultimo standard, di giugno 2016, è ECMA-262 Edition 7. È anche uno standard ISO.

Le caratteristiche principali di JavaScript sono:

- l'essere un linguaggio interpretato: il codice non viene compilato, ma interpretato (in JavaScript lato client, l'interprete è incluso nel browser che si sta utilizzando).

- la sintassi è relativamente simile a quella del C, del C++ e Java.
- definisce le funzionalità tipiche dei linguaggi di programmazione ad alto livello (strutture di controllo, cicli, ecc.) e consente l'utilizzo del paradigma object oriented.
- è un linguaggio debolmente tipizzato.
- è un linguaggio debolmente orientato agli oggetti. Ad esempio, il meccanismo dell'ereditarietà è più simile a quello del Self e del NewtonScript che a quello del linguaggio Java (che è un linguaggio fortemente orientato agli oggetti). Gli oggetti stessi ricordano più gli array associativi del Perl che gli oggetti di Java o del C++.

Altri aspetti di interesse: in JavaScript lato client, il codice viene eseguito direttamente sul client e non sul server. Il vantaggio di questo approccio è che, anche con la presenza di script particolarmente complessi, il web server non viene sovraccaricato a causa delle richieste dei client. Di contro, nel caso di script che presentino un codice sorgente particolarmente grande, il tempo per lo scaricamento può diventare abbastanza lungo. Un altro svantaggio è il seguente: ogni informazione che presuppone un accesso a dati memorizzati in un database remoto deve essere rimandata ad un linguaggio che effettui esplicitamente la transazione, per poi restituire i risultati ad una o più variabili JavaScript; operazioni del genere richiedono il caricamento della pagina stessa (con l'avvento di AJAX tutti questi limiti sono stati superati).

A differenza di altri linguaggi, quali il C o il C++, che permettono la scrittura di programmi completamente stand-alone, JavaScript viene utilizzato soprattutto in quanto linguaggio di scripting, integrato, quindi, all'interno di un altro programma.

L'idea di base è che il programma ospite (quello che ospita ed esegue lo script) fornisca allo script un'API ben definita, che consente l'accesso ad operazioni specifiche, la cui implementazione è a carico del programma ospite stesso. Lo script, quando eseguito, utilizza riferimenti a questa API per richiedere (al programma ospite) l'esecuzione di operazioni specifiche, non previste dai costrutti del linguaggio JavaScript in sé. In effetti, questo è esattamente lo stesso meccanismo che viene adottato anche in un linguaggio quale il C o il Java, nel quale il programma si affida a delle librerie, non previste dal linguaggio in sé, che permettono di effettuare operazioni quali l'I/O o l'esecuzione di chiamate a funzioni di sistema.

L'esempio tipico (e, forse, il più noto) di programma ospite per uno script JavaScript è quello del browser. Un browser tipicamente incorpora un interprete JavaScript; quando viene visitata una pagina web che contiene il codice di uno script JavaScript, quest'ultimo viene portato in memoria primaria ed eseguito dall'interprete contenuto nel browser.

Le varie implementazioni di JavaScript, come già accaduto per l'HTML, spesso non sono conformi agli standard, ma piuttosto sono costruite per funzionare con uno specifico browser web. L'attuale standard ECMAScript dovrebbe essere teoricamente la base di tutte le implementazioni JavaScript, ma in pratica i browser Mozilla (e Netscape) usano JavaScript, Microsoft Internet Explorer usa JScript, e altri browser come Opera e Safari usano altre implementazioni ECMAScript, spesso con ulteriori caratteristiche non standard per permettere la compatibilità con JavaScript e JScript.

JavaScript e JScript contengono molte caratteristiche che non sono parte dello standard ufficiale ECMAScript, e possono anche essere privi di diverse caratteristiche. In tal modo, sono in parte incompatibili, il che porta gli autori di script a sopperire a tali problemi. Tra i due, JavaScript è più

conforme allo standard: ciò significa che uno script redatto secondo gli standard ECMA funzionerà con la maggior parte dei browser.

Un altro effetto è che ciascun browser potrebbe trattare lo stesso script in modo diverso, e ciò che funziona in un browser potrebbe non funzionare in un altro browser, o persino in una diversa versione dello stesso browser.

Esempio di utilizzo di JavaScript all'interno di una pagina HTML:

```
<script type="text/javascript">
// <![CDATA[
Codice JavaScript...
// ]]>
</script>
```

Per ampliare le funzionalità offerte dal linguaggio Javascript sono disponibili diverse librerie. Le librerie contengono sempre dei sottoprogrammi che hanno delle funzioni per facilitare il lavoro di programmazione. Al contrario di un framework, una libreria viene sviluppata per un uso specifico e possiede per questo delle funzioni adatte al software di riferimento. Ad esempio si utilizza la libreria JavaScript D3.js (presente in DeskEmo) per la resa visiva dei dati, così da poter realizzare sia piccole tabelle, diagrammi e statistiche che rappresentazioni grafiche complesse (comprensive di animazioni e altre possibilità di interazione). Le librerie sono sempre legate ad un software che ricorre alle funzioni di una libreria di programmazione, nel momento in cui viene richiesta una precisa funzione della collezione, per questo funzionano solo all'interno di un programma e non possono essere eseguite in maniera autonoma.

Le librerie JavaScript sono dei codici riutilizzabili attraverso i quali vengono assegnati proprietà e funzioni specifiche per un sito. La libreria

JavaScript più famosa è **jQuery**, che offre molte funzioni utili, ma ve ne sono anche altre.

3.2.4. JQUERY

La libreria completa jQuery è quella più utilizzata e conosciuta, dal momento che l'omonimo codice può essere utilizzato su tutti i browser e dispone di molti plug-in. La libreria open source è una componente essenziale di molti CMS come WordPress, Drupal o Joomla!. jQuery serve a semplificare la gestione degli elementi DOM e presenta diverse funzioni per questo scopo:

- Scelta facile di elementi del sito (analogamente ai selettori CSS3);
- Modifica degli elementi scelti del sito (come posizione, colore, ecc.);
- Gestione degli eventi: gli elementi del sito reagiscono ai comandi degli utenti (click con il mouse, battitura tramite tastiera, ecc.);
- Facile realizzazione di effetti e animazioni;
- Le richieste Ajax facilitano l'interazione tra i comandi dati dall'utente e i dati del server (come la funzione di autocompletamento).

3.2.5. JQUERY UI

jQuery UI è un'estensione libera per jQuery e serve alla realizzazione e gestione di un'interfaccia utente (in inglese "User Interface" o abbreviato "UI"), per esempio per le pagine web o web app. È particolarmente adatta per realizzare effetti semplici ed elementi interattivi; infatti jQuery dispone di diverse funzioni per creare elementi interattivi (come drag&drop, ingrandimento e ridimensionamento degli elementi del sito, ecc.), animazioni, effetti e widget. Grazie all'editor grafico ThemeRoller è

possibile creare dei propri temi, ma anche utilizzare quelli già disponibili e adattarli; la struttura modulare permette di implementare solo le componenti necessarie.

Se si usano linguaggi di programmazione lato server come C o PHP, il codice di programmazione di regola viene portato a termine in maniera sequenziale. Questo significa che un server inizia con la configurazione di una nuova indicazione nel codice solo quando queste sono state eseguite e si è ottenuto un risultato. In questo caso si parla di un'elaborazione sincrona. L'esecuzione di ulteriori codici viene bloccata fino a quando il processo attuale non viene terminato. Questo può portare a consistenti ritardi nelle azioni che richiedono più tempo come accessi al file system, a database o a servizi web.

Molti linguaggi di programmazione, i *runtime environment* e le implementazioni basate su questi, supportano quindi la possibilità di eseguire parallelamente processi nei cosiddetti thread. Si tratta di singoli thread di esecuzione nell'ambito di un processo, con i quali si portano a termine le azioni, mentre viene eseguito il resto del codice. Lo svantaggio di questo metodo consiste nel fatto che più thread vengono iniziati, più tempo di CPU e memoria RAM sono necessari. In altre parole: il multithreading richiede molte risorse. Inoltre, ulteriori thread sono associati a sforzi di programmazione decisamente maggiori. Con un'implementazione lato server di JavaScript è possibile aggirare questi problemi. La base per questo è messa a disposizione da **Node.js**.

3.2.6. NODE.JS

Node.js è una piattaforma software con un'architettura basata su eventi, che permette di usare lato server il linguaggio di scripting JavaScript, originariamente pensato per l'utilizzo lato client. Questo significa quindi che si possono usare i linguaggi Java, .NET, Ruby o Pyton allo stesso modo

di PHP per scrivere il codice per il server. Node.js si usa nello sviluppo di applicazioni JavaScript lato server, che devono elaborare in tempo reale una grande quantità di dati. Il runtime enviroment è molto popolare per realizzare un web server leggero.

Il software multipiattaforma fu creato nel 2009 da Ryan Dahl ed era basato su V8, il motore JavaScript di Google (progettato per avere performance elevate, esegue e compila JavaScript, ha un garbage collector altamente efficiente e gestisce in modo efficace l'allocazione di memoria per gli oggetti), utilizzato anche nel browser Chrome. Il progetto nella forma di Node.js, iniziato dall'azienda Joyent, è passato dal 2005 alla Linux Foundation. Versioni aggiornate sono disponibili per Windows, Mac OS e Linux.

Node.js ha una libreria di diversi moduli JavaScript, che possono essere caricati attraverso una funzione semplice e messi a disposizione come componenti già pronti per lo sviluppo delle applicazioni web. Un esempio è il modulo http, che permette di generare con una singola funzione un web server obsoleto. Inoltre è possibile installare moduli aggiuntivi con il sistema di gestione dei pacchetti NPM (Node Package Manager).

Un grande vantaggio di Node.js è l'architettura basata su eventi, con la quale è possibile eseguire il codice di programmazione in maniera asincrona. Pertanto Node.js si affida su thread singoli ed un sistema di input-output (I/O) esterno, che permette un'elaborazione parallela di più operazioni di scrittura e lettura.

- **I/O asincrono**: tra i compiti classici di un server rientrano il rispondere alle richieste, il salvataggio dei dati in un database, la lettura dei file da un disco rigido e l'instaurazione di collegamenti tra le altre componenti di rete. Queste attività vengono raggruppate sotto l'acronimo I/O (input/output). Nei linguaggi di programmazione,

come C o Java, le operazioni I/O vengono eseguite in modo sincrono, portando a termine un compito dopo l'altro. Pertanto il sistema di input-output si blocca fino a che il compito attuale non viene portato a termine. Node.js usa invece un I/O asincrono, nel quale vengono delegate le operazioni di scrittura e di lettura al sistema operativo o al database. Questo permette di eseguire un grande numero di operazioni I/O parallelamente, senza che si arrivi al blocking, cosa che procura alle applicazioni basate su Node.js e JavaScript un grande vantaggio in termini di velocità.

- **Thread singoli**: per compensare i tempi di attesa nell'I/O sincrono, le applicazioni server sulla base del linguaggio di programmazione classico lato server usano ulteriori thread, con gli svantaggi sopra accennati di un approccio multithreading. Così ad esempio un Apache HTTP Server inizia un nuovo thread per ogni richiesta in entrata. Il numero dei possibili thread viene limitato dalla memoria di lavoro a disposizione e con questo anche il numero delle richieste, a cui si può rispondere parallelamente in un sistema multithreading sincrono. Node.js invece utilizza un solo thread sulla base del sistema di input-output esterno, e in questo modo sia la complessità che l'utilizzo delle risorse vengono chiaramente ridotte.
- **Architettura basata su eventi**: l'elaborazione asincrona delle operazioni I/O si realizza attraverso la struttura basata su eventi di Node.js. Questa si basa sostanzialmente su un singolo thread, che si trova in un event loop infinito. Questo event loop ha il compito di aspettare gli eventi e di gestirli. Intanto gli eventi possono essere presentati sia come compiti che come risultati. Se l'event loop registra un compito, ad esempio una richiesta del database, la si salva nel background attraverso la funzione callback di un processo. L'elaborazione del compito non avviene quindi nello stesso thread,

nel quale si esegue l'event loop, in modo che questo possa passare immediatamente all'evento successivo. Se si è eseguito un compito esterno, i risultati del processo esterno vengono restituiti attraverso la funzione callback come nuovi risultati dell'event loop. Di conseguenza questo può influire sulla consegna dei risultati.

Il modello basato su eventi e senza blocking ha il vantaggio che un'applicazione basata su Node.js non aspetti mai inerte i risultati. Così è ad esempio possibile eseguire allo stesso tempo diverse richieste del database, senza che l'esecuzione del programma venga arrestata. Una struttura di un sito web, che necessita di diverse richieste esterne, si può quindi effettuare con Node.js in maniera molto più veloce rispetto ad un processo di elaborazione sincrono.

Sulla base dell'ambiente di runtime V8, Node.js vi permette di realizzare server efficienti e altre applicazioni di rete scritti in JavaScript, linguaggio di scripting preferito sul web, e gli mette a disposizione una libreria molto ampia con moduli dei programmi. Questi moduli base integrati nell'ambiente di runtime comprendono funzionalità base come l'interazione con il sistema operativo, la comunicazione di rete o l'uso di meccanismi di criptazione. Inoltre Node.js, grazie al sistema di gestione dei pacchetti **NPM** integrato, offre la possibilità di installare moduli di programmi Node.js in maniera individuale.

3.2.7. NODE PACKAGE MANAGER

NPM è il principale software utilizzato per maneggiare i moduli di Node.js e consente di condividere il codice per problemi tipici tra gli sviluppatori JavaScript. La filosofia alla base di NPM è quella che se un problema è stato già risolto da altri programmatori non ha senso doverlo sviluppare per conto proprio, è possibile utilizzare la soluzione condivisa messa a disposizione di tutti. NPM oltre a consentire il riuso del codice, consente di

tenerlo costantemente sotto controllo in modo da aggiornarlo se dovesse essere migliorato.

NPM suddivide il codice in package o moduli. Un package è una directory che contiene uno o più file insieme ad un file chiamato *package.json* che contiene dei dati relativi al pacchetto. In genere una tipica applicazione è costituita da decine o migliaia di package.

NPM è interamente scritto in JavaScript ed è stato sviluppato da Isaac Z. Schlueter stanco di utilizzare CommonJS. NPM è l'acronimo di Node Package Manager, anche se nel 2014 Collin Winter ha dichiarato che NPM non è un acronimo ma si tratta di un acronimo inverso "NPM is not an acronym".

3.3. ELECTRON FRAMEWORK

Il limite di tutte le tecnologie web elencate in precedenza è sempre stato il fatto che il loro utilizzo fosse limitato in ambito web. Infatti quando si vuole sviluppare un'applicazione desktop solitamente la prima cosa a cui si pensa è per quale sistema operativo svilupparla, quale linguaggio usare ed opzionalmente a quale base dati si ha bisogno di connettersi, non prendendo neanche in considerazione la possibilità di utilizzare le tecnologie su cui è basato il web.

Non a caso il mondo dell'informatica intanto si sta evolvendo verso l'utilizzo di applicazioni web per la loro semplicità di distribuzione e aggiornamento, la compatibilità con ogni sistema operativo e la superfluità di un'installazione. Da queste esigenze nasce Electron, che sta dando il via ad una nuova era per le applicazioni desktop.

Dal sito ufficiale electronjs.org, il suo slogan è “Build cross platform desktop apps with JavaScript, HTML, and CSS” (“Crea applicazioni desktop multipiattaforma con Javascript, HTML e CSS”).

Electron è un framework open source sviluppato dal team di GitHub che permette di dare vita ad un’applicazione desktop utilizzando le stesse tecnologie di un sito web unendo la potenza del browser Chromium, la flessibilità di Node.js ed il più grande ecosistema di librerie open source al mondo: npm. In aggiunta a tutto questo la possibilità di pacchettizzare l’applicazione per Mac, Windows, e Linux con lo stesso risultato in termini di grafica e funzionalità.

L’installazione di Electron avviene tramite npm, utilizzando il comando:

```
npm install electron
```

Fondamentalmente un’applicazione sviluppata con Electron si compone di tre file: un manifest, un entrypoint HTML e un entrypoint javascript. Il manifest sarà il nostro file *package.json* dove possiamo dichiarare le varie proprietà come il nome dell'applicazione, la sua versione ed il nome dell'entrypoint javascript:

```
{
  "name"    : "DeskEmo",
  "version" : "1.0.0",
  "main"    : "main.js"
}
```

In questo modo Electron saprà quale file javascript potrà interpretare come main della nostra applicazione. Nel main.js con la funzione *require* possiamo iniettare una serie di oggetti javascript che ci saranno utili per

scrivere la nostra applicazione ad esempio per controllare il ciclo di vita della nostra applicazione, oppure per controllare il browser. Un semplice main di una nostra applicazione potrebbe essere:

```
var app = require('app');  /* Modulo che controlla il ciclo di vita dell'applicazione. */
var BrowserWindow = require('browser-window'); /* Modulo che crea la finestra dove verr
à visualizzata l'applicazione. */


// Report di eventuali crash.
require('crash-reporter').start();


/* Si usa dichiarare questa variabile per mantenere una referenza     globale della finestra
principale */
/* Se ciò non venisse fatto, la finestra verrebbe chiusa
   all'avvio del Garbage Collector di Javascript */
var mainWindow = null;

/* Chiude l'applicazione quando tutte le finestre sono chiuse */
app.on('window-all-closed', function() {
  /* Su macOS è usanza comunque lasciare attive le applicazioni
   finquando l'utente non esprime la volontà di chiuderle
   esplicitamente utilizzando Cmd + Q */
  if (process.platform != 'darwin') {
    app.quit();
  }
});
/* Questo metodo viene chiamato quando Electron ha finito la fase
   di inizializzazione ed è pronto per creare una nuova
   finestra */
app.on('ready', function() {
  // Crea la finestra.
  mainWindow = new BrowserWindow({width: 800, height: 600});
  // e visualizza il file index.html dell'applicazione
  mainWindow.loadUrl('file://' + __dirname + '/index.html');
  // Emitted when the window is closed.
  mainWindow.on('closed', function() {
    /* Rimuovi qualsiasi referenza all'oggetto 'Finestra'
```

```
  mainWindow = null;
  });
});
```

3.4. INTERFACCIA

3.4.1. DESIGN RESPONSIVO

L’interfaccia di DeskEmo si basa su un design di tipo responsivo e richiama lo stile grafico dell’applicazione per i dispositivi mobili (Hb Meter). Il design responsivo, o responsive web design (RWD), indica una tecnica di web design per la realizzazione di interfacce in grado di adattarsi graficamente in modo automatico al dispositivo coi quali vengono visualizzati (computer con diverse risoluzioni, tablet, smartphone, cellulari, web tv), riducendo al minimo la necessità dell'utente di ridimensionare e scorrere i contenuti.

La necessità di adattare l'impaginazione alle diverse dimensioni e risoluzioni degli schermi, ha introdotto il concetto di "Resolution breakpoint" ("punti di interruzione della risoluzione"), in modo da stabilire delle soglie alle quali modificare la presentazione grafica in funzione della larghezza del dispositivo. Tali soglie sono generalmente espresse in pixel, anche se l'aumento della densità dei pixel nelle nuove generazioni di dispositivi comporta che non si possa considerare l'area di visualizzazione solo in termini di pixel, senza considerarne l'effettiva dimensione.

Il design responsivo è un importante elemento dell'accessibilità, la quale tiene conto inoltre di numerosi altri fattori, incentrati non solo sui dispositivi ma anche sulle caratteristiche dell'utente (quali: capacità cognitive, vista, difficoltà fisiche, e così via).

Il framework Bootstrap (utilizzato in DeskEmo, di cui daremo maggiori dettagli in seguito) identifica (in riferimento al "max-device-width") quattro tipi di device e corrispondenti resolution breakpoint:

- extra small device con risoluzione inferiore a 768 pixel
- small device con risoluzione fino a 992 pixel
- desktop con risoluzione inferiore a 1200 pixel
- large device con risoluzione superiore a 1200 pixel

Le strategie per riorganizzare i contenuti in funzione dei dispositivi, hanno portato alla classificazione di diverse tipologie di impaginazioni grafiche:

- Reflowing
- Equal Width
- Off Canvas
- Source-Order Shift
- List
- Grid Block

L'impaginazione di tipo Reflowing contiene diverse sottocategorie: Mostly Fluid (multi colonna con margini più larghi su grandi schermi, e su schermi narrow le aree vengono allineate su un'unica colonna), Column Drop, Layout Shifter, Tiny Tweaks.

L'impaginazione di tipo Equal Width divide lo schermo in colonne delle stesse dimensioni, e il numero di colonne può aumentare o diminuire proporzionalmente alla larghezza dello schermo.

L'impaginazione di tipo Off Canvas divide lo schermo in aree, principalmente verticali, che al diminuire della risoluzione non vengono mostrate in funzione della loro importanza fino a mostrare una sola colonna con il contenuto principale.

L'impaginazione di tipo Source-Order Shift sfrutta le proprietà flexbox e box-ordinal-group dei css per cambiare l'ordine con i quali i blocchi di contenuti vengono visualizzati nella pagina.

L'impaginazione di tipo List organizza la pagina in semplici liste di elementi che, analogamente a quanto succede sulle impaginazioni di tipo Equal Width, sono visualizzate su un numero di colonne proporzionali alla larghezza dello schermo così come le impaginazioni di tipo Grid Block che suddividono il layout in una griglia di rettangoli o quadrati.

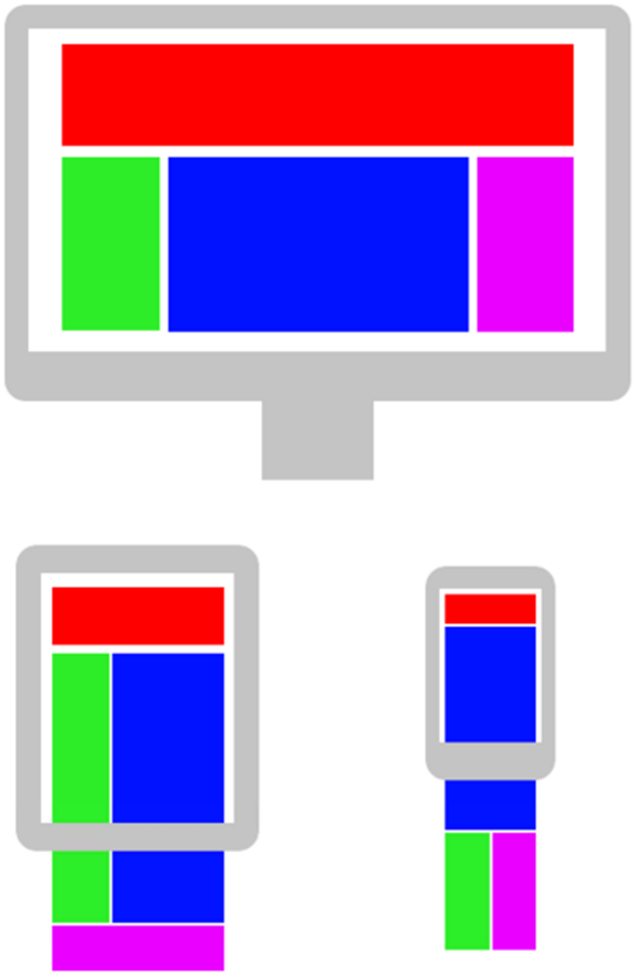

Figura 10: Un esempio di riorganizzazione con layout reflowing dei contenuti su device desktop, tablet e smartphone

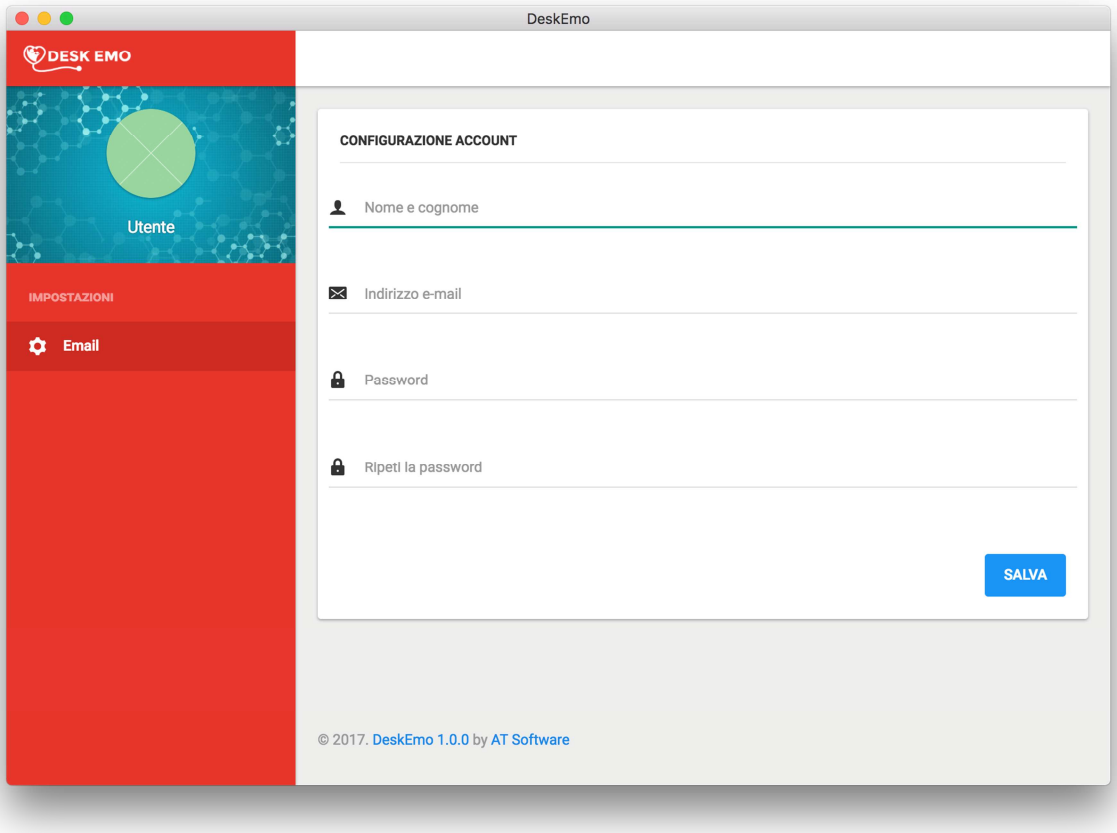

Figura 11: Interfaccia di DeskEmo in modalità desktop

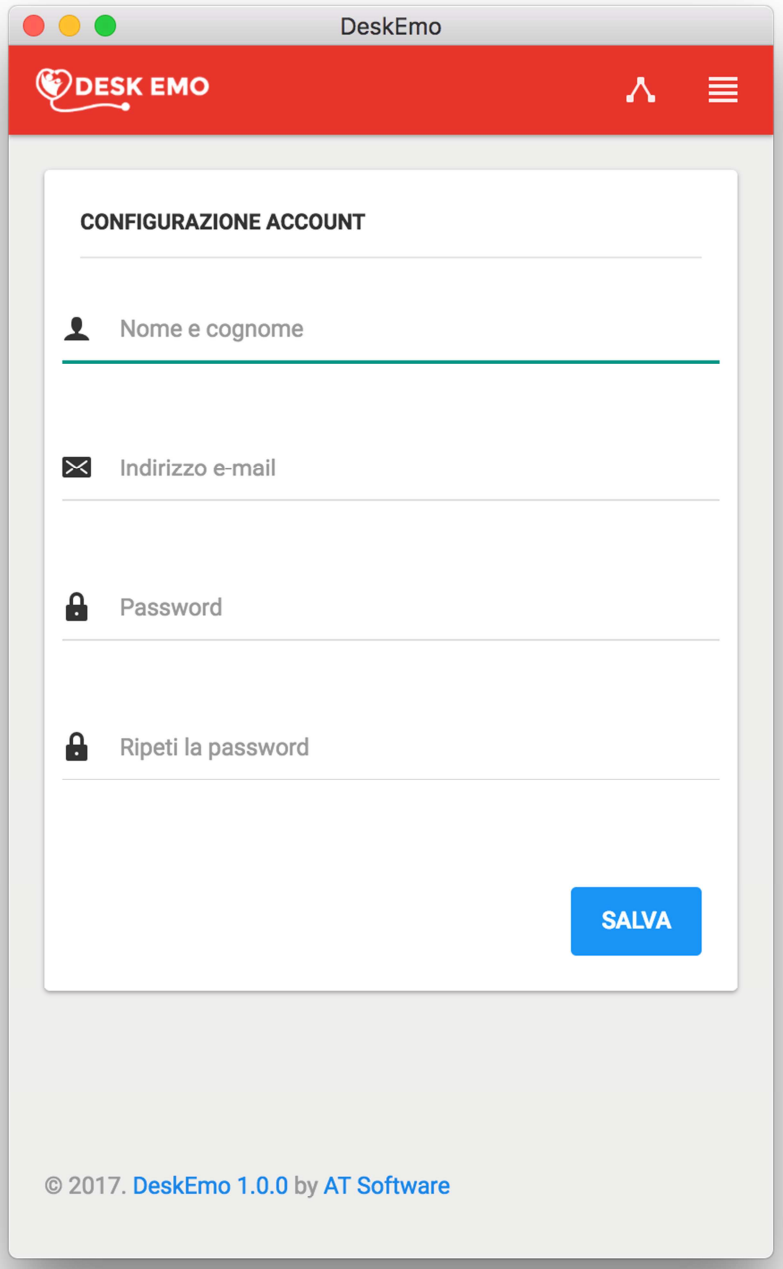

Figura 12: Interfaccia di DeskEmo in modalità extra small device

3.4.2. BOOTSTRAP

Alla base della struttura dell'interfaccia grafica di DeskEmo c'è Bootstrap, un progetto che nasce nell'anno 2010 per opera degli sviluppatori Mark Otto e Jacob Thornton. Inizialmente si presentava come un progetto interno a Twitter, ma successivamente è diventato indipendente ed è, perciò, utilizzabile dagli sviluppatori di tutto il mondo come base per la realizzazione di interfacce web. Al momento è considerato il più efficiente ed il più utilizzato framework per rendere i siti web responsive.

La sua definizione esatta è "HTML, CSS, and JS toolkit from Twitter", ovvero una raccolta di strumenti grafici, stilistici ed impaginazione che permettono di avere a disposizione una gran quantità di funzionalità e di stili modificabili e adattabili a seconda delle nostre esigenze. Bootstrap è compatibile con tutte le ultime versioni dei principali browser e la sua principale funzione è facilitare la progettazione di un design responsivo. Esso si pone, perciò, come una libreria multidispositivo e multipiattaforma.

I componenti principali di Bootstrap sono:

- Grid System, non è altro che un insieme di fogli che considerano il contenitore generale disposto su una griglia con una larghezza base di 960 px, nella quale distribuire i contenuti in varie righe e colonne, che definiranno il layout, ovvero la struttura base del nostro template.
- CSS Base, contiene una serie di stili predefiniti riguardanti tutta la parte tipografica, ovvero la parte riguardante i titoli (H1, H2, ...) e la gestione di tabelle, paragrafi, form e anche pulsanti per richiamare stili e icone pronti all'uso.
- Componenti e javascript, ovvero elementi quali menù dropdown, interfacce a tab, tooltip, alert, menù adaccordion, slider, banner di navigazione, PopOver che ci aiutano nell'implementazione degli

elementi dinamici della pagina senza la necessità di scrivere codice Javascript grazie alla presenza dei data-attributes, che Bootstrap interpreta e gestisce senza nessun intervento da parte nostra.

Esempio di una riga suddivisa in tre colonne utilizzando Bootstrap:

```
<div class="container">
  <div class="row">
    <div class="col-sm">
      Una di tre colonne
    </div>
    <div class="col-sm">
      Una di tre colonne
    </div>
    <div class="col-sm">
      Una di tre colonne
    </div>
  </div>
</div>
```

3.5. ORGANIZZAZIONE DEI DATI

3.5.1. ARCHIVIAZIONE

DeskEmo non fa alcun uso di database strutturato. La motivazione di questa scelta di progetto è da attribuire ad uno dei principali obiettivi con cui ci si era prefissati fin dall'inizio; rendere DeskEmo un'applicazione leggera, portatile e di semplice installazione e configurazione. Infatti l'installazione di un database locale (per esempio MySQL) richiederebbe l'intervento di un esperto. Utilizzare un database online, oltre che richiedere l'intervento di un esperto in fase di installazione, porterebbe i servizi di DeskEmo a dover dipendere da un server provider andando incontro ad eventuali problematiche dovute alla mancata manutenzione, assenza di connessione, sovraccarico del server e così via. Inoltre affidarsi ad un server provider richiederebbe un esborso economico per via dei servizi offerti.

Per tutte queste motivazioni, durante la progettazione di DeskEmo si è deciso di organizzare i dati utilizzando i file che vengono salvati ed organizzati in directory speciali dal sistema operativo, rispettivamente:

- *%APPDATA%* su Windows
- *$XDG_CONFIG_HOME* oppure *~/.config* su Linux
- *~/Library/Application Support* su macOS

Queste directory raramente vengono consultate dagli utenti del sistema operativo (di default vengono nascoste dal sistema operativo stesso). L'organizzazione in directory speciali, nascoste all'utente, garantisce l'integrità dei file ed evita qualsiasi tipo modifiche accidentali da parte dell'utente.

L'accesso a queste directory avviene per mezzo di una particolare funzione messa a disposizione dal framework Electron:

app.getPath(name)

dove *name* è una stringa e, in base al suo valore, la funzione ritorna un'altra stringa contenente la directory richiesta. Nel caso di DeskEmo, come parametro alla funzione verrà passata la stringa *appData*.

Tutti i dati sensibili elaborati da DeskEmo sono crittografati con una chiave AES (Advanced Encryption Standard) a 256 bit e la loro trasmissione da parte di Hb Meter (l'applicazione mobile) avviene tramite canali sicuri che garantiscono l'integrità e la provenienza dei dati.

3.5.2. TRASMISSIONE E RICEZIONE

Le informazioni ricevute da DeskEmo da parte di Hb Meter viaggiano attraverso canali sicuri che garantiscono autenticazione, integrità dei dati e cifratura operando sul protocollo standard per la trasmissione via internet di e-mail.

I due protocolli più popolari per la comunicazione via email sono POP e IMAP. Entrambi i protocolli permettono ad un client (programma di posta elettronica oppure servizio di webmail) di accedere, leggere e cancellare le e-mail da un server, ma con alcune differenze. Con entrambi i protocolli, il client scarica la posta direttamente sul PC, eventualmente cancellandola dal server, ma è altresì possibile conservare copia delle proprie e-mail sul server, e scaricarle in un secondo momento da altri computer. IMAP, a differenza di POP, permette procedure complesse di sincronizzazione. Ecco un elenco delle caratteristiche hanno spinto ad adottare il protocollo IMAP in DeskEmo a discapito del protocollo di tipo POP:

- Accesso alla posta sia online che off-line

Quando si utilizza il POP3, il client si connette per scaricare i nuovi messaggi e poi si disconnette. Con l'IMAP il client rimane connesso e risponde alle richieste che l'utente fa attraverso l'interfaccia; questo permette di risparmiare tempo se ci sono messaggi di grandi dimensioni. POP3 salva in locale i messaggi, IMAP li lascia sul server e li memorizza temporaneamente nella cache (sebbene si possa poi configurare il client perché crei un file dati archiviato in locale, ad esempio nel formato pst).

- Più utenti possono utilizzare la stessa casella di posta

Il protocollo POP assume che un solo client (utente) sia connesso ad una determinata mailbox (casella di posta), quella che gli è stata assegnata. Al contrario l'IMAP (dalla versione 4) permette connessioni simultanee alla

stessa mailbox, fornendo meccanismi per controllare i cambiamenti apportati da ogni dispositivo/client di posta con il quale si utilizza l'account.

- Supporto all'accesso a singole parti MIME di un messaggio

La maggior parte delle e-mail sono trasmesse nel formato MIME, che permette una struttura ad albero del messaggio, dove ogni ramo è un contenuto diverso (intestazioni, allegati o parti di esso, messaggio in un dato formato, eccetera). Il protocollo IMAP4 permette di scaricare una singola parte MIME o addirittura sezioni delle parti, per avere un'anteprima del messaggio o per scaricare una mail senza i file allegati.

- Sottoscrizione delle cartelle IMAP

Il protocollo è sviluppato secondo un framework che permette ad un client di posta di sottoscrivere le cartelle prescelte. L'operazione è eseguita tramite query e successiva scelta di quale singola cartelle sottoscrivere. Solitamente, il client poi permette di visualizzare nella gerarchia di cartelle solo quelle sottoscritte o tutte. Tra le altre cose la sottoscrizione permette di condividere una cartella tra dispositivi/client/utenti diversi. La sottoscrizione è un'operazione di configurazione con il server di posta necessaria per poter visualizzare le cartelle IMAP.

- Supporto per attributi dei messaggi tenuti dal server.

Attraverso l'uso di attributi, tenuti sul server, definiti nel protocollo IMAP4, ogni singolo client può tenere traccia di ogni messaggio, per esempio per sapere se è già stato letto o se ha avuto una risposta.

- Accesso a molteplici caselle di posta sul server

Alcuni utenti, con il protocollo IMAP4, possono creare, modificare o cancellare mailbox (di solito associate a cartelle) sul server. Inoltre, questa

gestione delle mailbox, permette di avere cartelle condivise tra utenti diversi.

- Possibilità di fare ricerche sul server

L'IMAP4 permette al client di chiedere al server quali messaggi soddisfano un certo criterio, per fare, per esempio, delle ricerche sui messaggi senza doverli scaricare tutti.

- Supporto di un meccanismo per la definizione di estensioni

Nelle specifiche dell'IMAP è descritto come un server può far sapere agli utenti se ha delle funzionalità extra. Molte estensioni dell'IMAP sono molto diffuse, ad esempio l'IMAP Idle, ovvero la funzione del protocollo IMAP descritto nella RFC 2177 che consente a un client di indicare al server che è pronto ad accettare notifiche in tempo reale. Questa caratteristica permette quindi all'utente di ricevere dal server ogni modifica che avviene nella casella e-mail, senza il bisogno di premere ogni volta "Invia/Ricevi".

L'IMAP è principalmente utilizzato in ambito professionale ovvero presso enti ed imprese, dove un utente cambia postazione spesso: con il POP3, sarebbe necessario scaricare i messaggi ogni volta che si cambia dispositivo, mentre con l'IMAP si possono scaricare solo i nuovi messaggi o accedere ad un messaggio specifico senza dover scaricare gli altri.

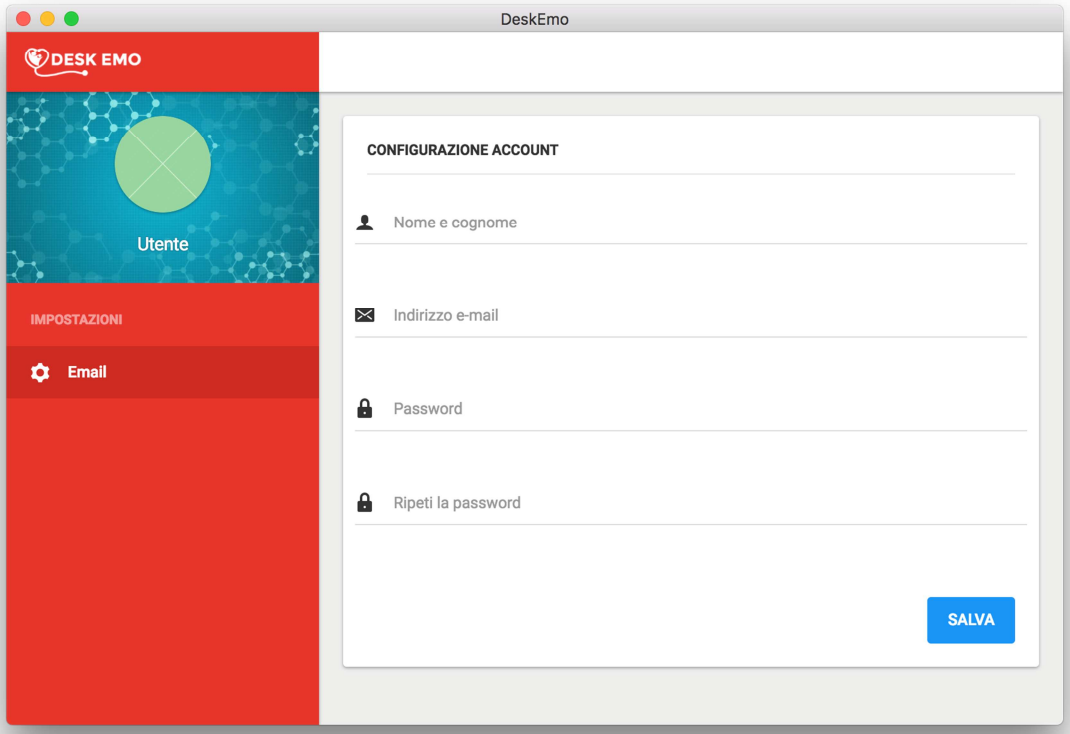

Figura 13: Primo avvio di DeskEmo in cui viene chiesto al medico curante o di laboratorio di inserire i propri dati così da poter iniziare a ricevere i report tramite email

Per garantire un canale sicuro durante lo scambio dei dati è stato adottato il protocollo TLS che consente alle applicazioni client/server di comunicare attraverso una rete in modo tale da prevenire il 'tampering' (manomissione) dei dati, la falsificazione e l'intercettazione. È un protocollo standard IETF che, nella sua ultima versione, è definito nella RFC 5246, sviluppata sulla base del precedente protocollo SSL da Netscape Communications.

Il protocollo TLS permette un'autenticazione bilaterale in cui entrambe le parti si autenticano in modo sicuro scambiandosi i relativi certificati. Questa autenticazione (definita Mutual authentication) richiede che anche il client possieda un proprio certificato digitale.

Tutte le informazioni sensibili scambiate via email, una volta giunte sul dispositivo del medico curante, rimangono archiviate in locale per il tempo necessario al medico o allo specialista di verificare lo stato e la salute del paziente per poi essere eliminate definitivamente. Questo aggiunge un ulteriore livello di sicurezza nel garantire la privacy di ogni singolo paziente.

I dati storici sono infatti archiviati senza riferimenti a dati personali, al fine della garanzia della privacy e della sicurezza. Il link con il paziente originale è mantenuto attraverso l'uso di una chiave cifrata che viene di volta in volta inviata dall'applicazione lato paziente (Hb Meter).

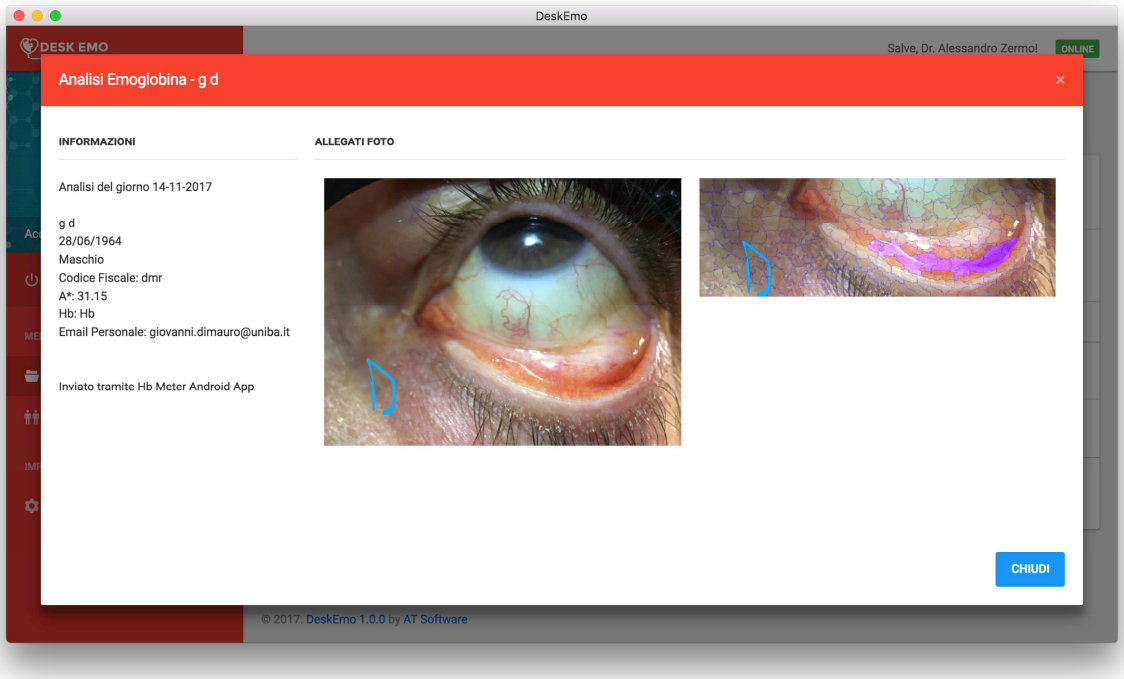

Figura 14: Visualizzazione di un report in DeskEmo ricevuto dall'applicazione lato paziente

3.5.3. DATA ANALYSIS

Inizialmente DeskEmo è stata pensata come un'applicazione semplice e leggera capace esclusivamente di far visualizzare al medico curante o di laboratorio i report ricevuti dall'applicazione lato paziente, ma spinti dalle enormi potenzialità delle tecnologie utilizzate e dalla voglia di rendere DeskEmo un software più completo si è deciso di integrare una nuova sezione in cui, grazie alla presenza dei dati storici, è possibile visualizzare l'andamento dei valori dei parametri ematologici dei pazienti così da poter avere un quadro generale delle loro condizioni di salute e permettere diagnosi più veloci ed efficaci.

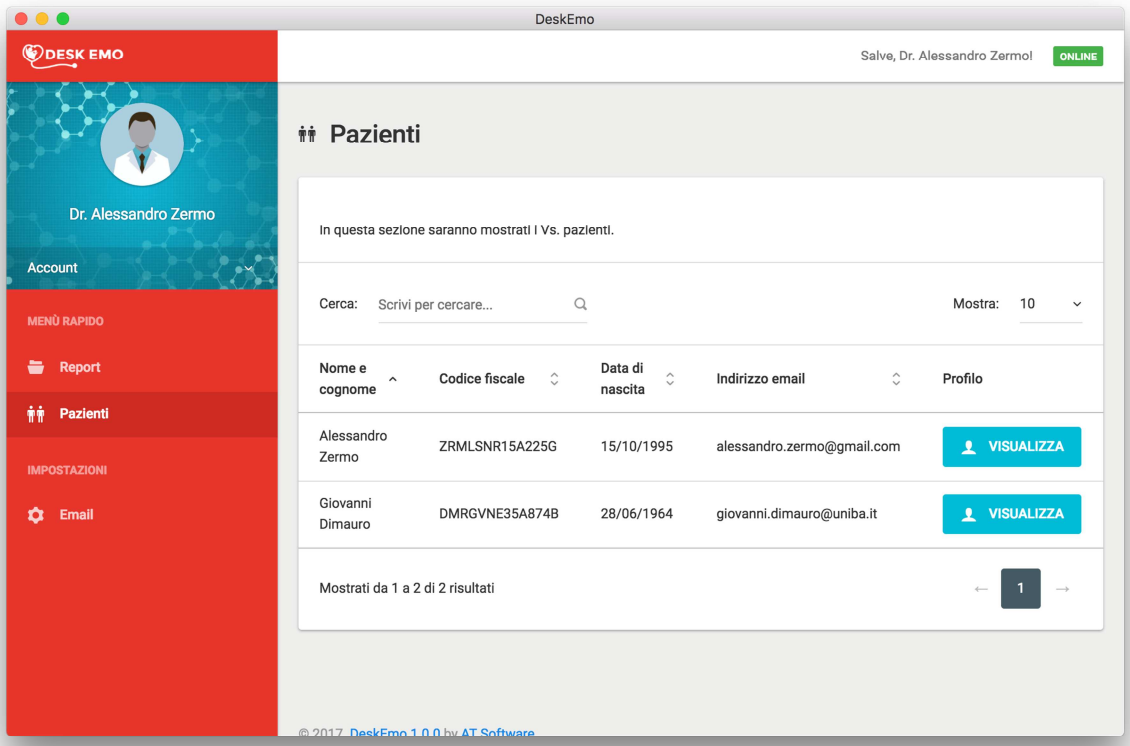

Figura 15: Panoramica della sezione Pazienti in cui è possibile consultare lo storico dei rispettivi parametri ematologici

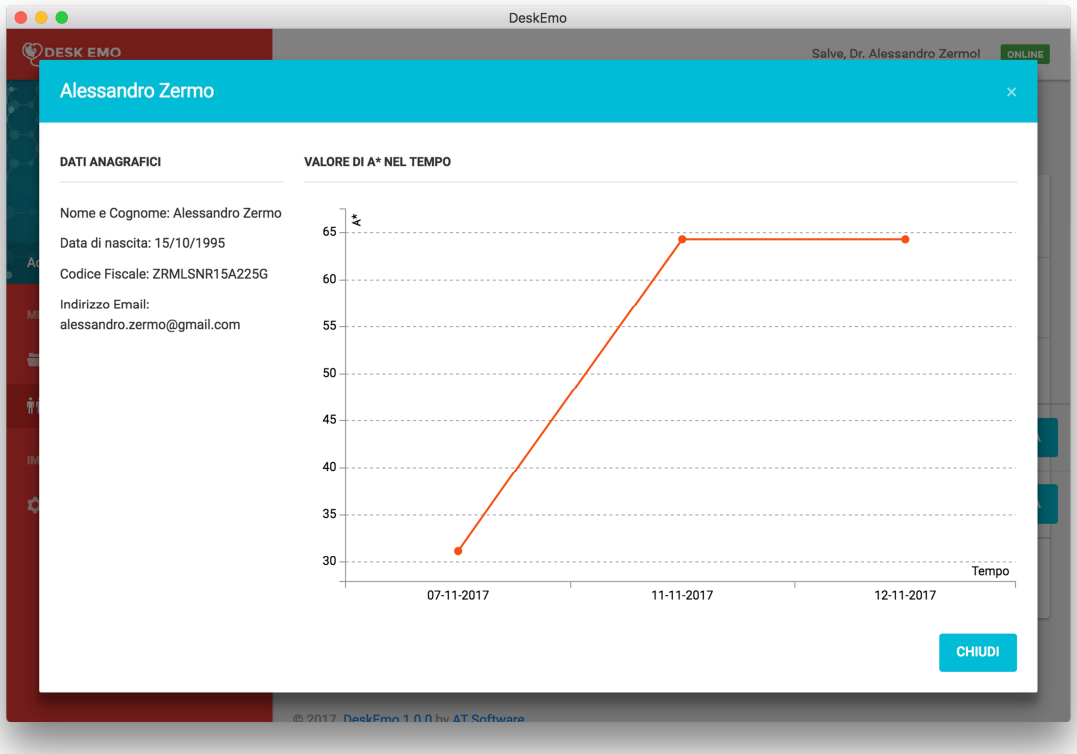

Figura 16: Visualizzazione dello storico dei valori dei parametri ematologici di un paziente

Per la visualizzazione dei dati dello storico dei valori dei parametri ematologici è stato realizzato un apposito parser in grado di leggere il formato dei report ricevuti dall'applicazione lato paziente ed estrapolare le informazioni contenute al loro interno tra cui:

- Anagrafica del paziente
  - Nome
  - Cognome
  - Data di nascita
  - Codice Fiscale
  - Indirizzo email
- Chiave univoca rappresentativa del paziente in DeskEmo

- Parametri ematologici
    - Valore di A*, indicatore principale del livello di emoglobina
    - Foto della congiuntiva
        - Foto senza indicazioni della selezione effettuata dal paziente; in questo modo il medico curante può effettuare una valutazione personale del colore della congiuntiva.
        - Foto con le indicazioni della selezione effettuata dal paziente;

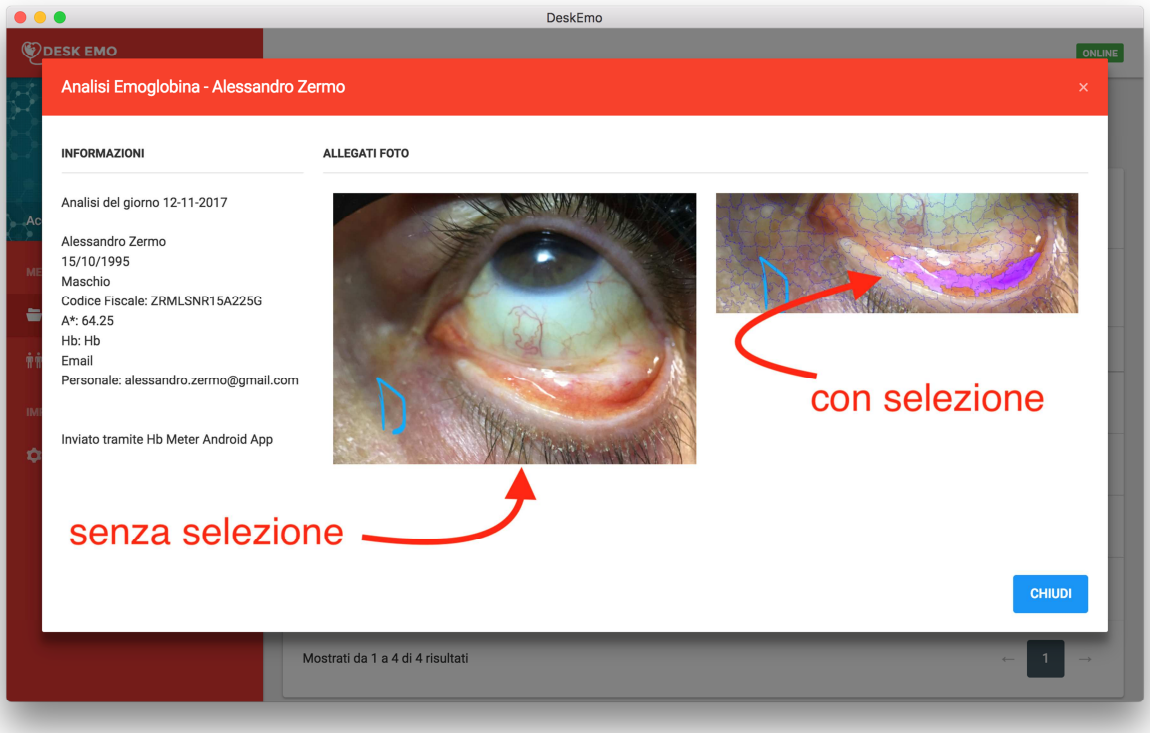

Figura 17: Visualizzazione di un report con i rispettivi allegati foto

4. TEST E CONCLUSIONI

4.1. TEST EFFETTUATI

Per Hb Meter (l’applicazione mobile lato paziente) sono stati effettuati test preliminari basati sulla correlazione del solo valore dell'emoglobina e la media dei valori del canale A* menzionato nei capitoli precedenti.

I test sono stati condotti presso l'Istituto Tumori "Giovanni Paolo II" e presso il centro trasfusionale dell'Ospedale della Murgia "Fabio Perinei", su un campione di 50 pazienti che si sono sottoposti al prelievo di una provetta di sangue al fine della stima dell'emoglobina, successivamente si è proceduto all’acquisizione della foto della congiuntiva palpebrale per la valutazione di quest'ultima tramite Hb Meter.

I test hanno avuto un riscontro positivo dimostrando che il lavoro effettuato sull’applicazione per dispositivi mobili ha raggiunto il suo scopo, ovvero ha prodotto un nuovo metodo di selezione pratico, veloce ed intuitivo, con l’ausilio di un dispositivo smartphone e un dispositivo hardware di supporto realizzato ad-hoc.

Constatato il funzionamento corretto di Hb Meter, durante la fase di sviluppo di DeskEmo sono stati eseguiti test sia di tipo White Box (test strutturale) che Black Box (test funzionale).

4.1.1. TEST STRUTTURALE

Il test strutturale, detto anche white box o verifica strutturale, è un particolare tipo di test che viene effettuato per rilevare errori in uno o più

componenti (parte di codice, metodo, funzione, classe, programmi, ecc.) di un sistema software.

Il suo funzionamento si basa su alcuni criteri che hanno lo scopo di trovare dati di test che consentano di percorrere tutto il programma.

Per trovare un errore nel codice, infatti, bisogna usare dei dati che “percorrono” la parte erronea del programma. Per testare una parte di programma si introduce il concetto di cammino: una sequenza di istruzioni attraversata durante un'esecuzione. Naturalmente non esiste un criterio in grado di testare ogni singolo cammino dato l'elevato numero di questi ultimi (soprattutto in presenza di cicli), tuttavia, è possibile trovare un numero finito di cammini indipendenti che combinati tra loro forniscano tutti (o per lo meno la maggior parte) i restanti cammini. Se eseguito correttamente e senza particolari eccezioni, il test può coprire fino al 90% delle istruzioni.

A differenza della metodologia black box, per cui non si ha a disposizione il codice, alla base della metodologia white box sta proprio l'analisi del codice, che permette appunto di identificare i cammini da percorrere e viene effettuata anche in modo manuale, con l'uso di diagrammi di flusso che mostrano i diversi cammini del programma e delle tabelle di traccia, che simulano l'esecuzione del codice.

Durante la fase di coding di DeskEmo è stata quindi adottata una procedura di testing di tipo incrementale. Ad ogni nuova funzione aggiunta nel codice è stato corrisposto uno specifico test al fine di verificare il corretto funzionamento di essa.

Per il corretto funzionamento del client mail sono stati utilizzati diversi provider di posta elettronica (in particolar modo i più diffusi come gmail, outlook e libero) così da poter assicurare il corretto funzionamento con essi

e quindi soddisfare il requisito fondamentale di DeskEmo: quello di essere un software flessibile e capace di adattarsi alla maggior parte dei contesti di business.

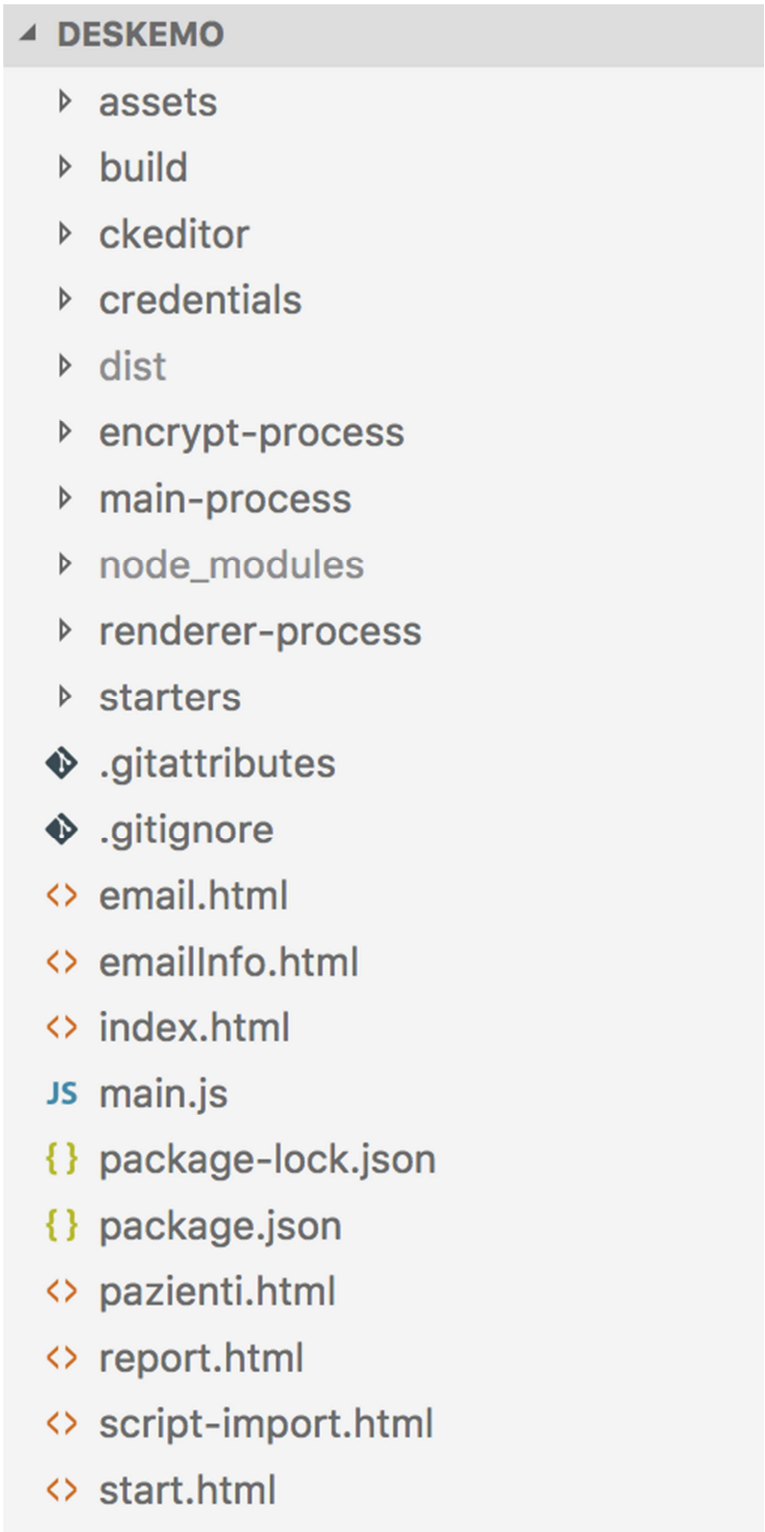

Figura 18: La struttura del codice sorgente di DeskEmo

4.1.2. TEST FUNZIONALE

Il test funzionale è una tecnica di tipo black-box che basa le proprie definizioni dei casi di test sulle specifiche funzionali delle componenti software oggetto di test. È una tipologia di test considerata indispensabile per assicurare che il sistema software implementi esattamente le funzionalità espresse nei requisiti, per scoprire ed eliminare tutte le possibili anomalie (bug, difetti) che non sono stati riscontrati dal team di sviluppo durante la fase di Unit Testing e per verificare che nuovi sviluppi/attività di bug-fixing non abbiano introdotto regressioni nelle componenti software precedentemente testate con successo (Test di Non-Regressione).

Per questa tipologia di test è stato coinvolto anche il team di sviluppo dell'applicazione lato paziente (Hb Meter) in modo da poter verificare il corretto funzionamento della comunicazione con DeskEmo.

I test white box hanno emulato uno scenario tipico del contesto di business di uno studio medico privato in cui DeskEmo viene avviato e lasciato in esecuzione su un normale computer. Durante il corso della giornata, tramite Hb Meter, sono stati inviati vari report (da diversi dispositivi mobili) all'indirizzo email configurato sul software DeskEmo in esecuzione sul computer dello studio medico privato. Tramite queste modalità si è potuto constatare il corretto funzionamento di DeskEmo in un contesto reale.

4.2. CONCLUSIONI

L’obiettivo di questo lavoro è stato quello di trovare un metodo per semplificare e velocizzare la comunicazione tra paziente e medico nello scambio dei dati elaborati da Hb Meter che, stando al lavoro precedente, il paziente doveva inviare un’email al medico e quest’ultimo poi, dal suo client di posta elettronica, avrebbe dovuto innanzitutto cercare i vari report ricevuti tra la propria casella email per poi procedere ad una organizzazione manuale degli stessi senza aver alcun modo la possibilità di gestione dello storico delle informazioni ricevute.

Specialmente in uno studio medico, dove il tempo e l’efficienza nell’effettuare qualsiasi tipo di diagnosi assumono valori da non sottovalutare, DeskEmo compie egregiamente il suo compito distinguendo i vari report dalla posta ordinaria, tracciando uno storico dei vari parametri ematologici ed organizzando tutti i dati in maniera efficiente ed efficace così da garantire una consultazione semplice ed immediata.

DeskEmo, come già ampiamente descritto nei capitoli precedenti, è stato progettato e sviluppato in modo da poter essere compatibile con qualsiasi sistema operativo desktop ed utilizza le tecnologie del web garantendo la massima portabilità e la stessa esperienza d’uso indipendentemente dalla macchina su cui viene eseguito. Da non sottovalutare, data la natura web di DeskEmo, una futura integrazione con il RESP (Registro Elettronico Sanitario Personale).

4.2.1. SVILUPPI FUTURI

Le enormi potenzialità delle tencologie web fanno di DeskEmo un software malleabile e capace di integrare sempre nuove funzionalità.

Alcune idee per quanto riguarda gli sviluppi futuri sono:

- Progettazione di un sistema multi-account in modo da poter utilizzare più indirizzi di posta elettronica per ricevere i report.
- Integrazione di DeskEmo con il RESP (Registro Elettronico Sanitario Personale).
- Realizzazione di nuove applicazioni mobili in grado di poter comunicare con DeskEmo ed inoltrare parametri che non siano solo ematologici.
    - Con associato lo sviluppo di una nuova sezione in DeskEmo per assicurare la capacità di comprensione ed elaborazione dei nuovi parametri.
- Implementare un metodo più sicuro di autenticazione e comunicazione per il client mail utilizzando un’architettura di tipo REST (REpresentational State Transfer) e quindi servirsi delle rispettive REST e RESTful API messe a disposizione dai provider di posta elettronica più importanti.

5. LISTATO DEL CODICE SORGENTE

5.1. PROCESSI PRINCIPALI

5.1.1. MAIN

```
const {
    app,
    BrowserWindow
} = require('electron')
const path = require('path')
const url = require('url')
const separator = setSeparator();
var fs = require('fs');


/**
 * Imposta il separatore delle directory in base al sistema operativo
 * WINDOWS = \
 * MAC,LINUX = /
 */
function setSeparator() {
    if (process.platform === 'win32')
        return '\\'
    return '/'
}

/**
 * Se non esiste la directory dell'application support --> creala
 * In questa directory verranno salvati tutti i file necessari
 * al corretto funzionamento dell'applicazione
 *
 * Su Mac la directory è ~/Library/Application Support
 * Su Windows è %APPDATA%
 * Su Linux è $XDG_CONFIG_HOME oppure ~/.config
 */
function createApplicationSupportFolder() {
    if (!fs.existsSync(getApplicationSupportFolderPath())) {
```

```
        fs.mkdirSync(getApplicationSupportFolderPath());
    }
}

// Keep a global reference of the window object, if you don't, the window will
// be closed automatically when the JavaScript object is garbage collected.
let win
//let firstTimeWindows;

function createWindow() {
    // Creo il menu dell'app (File - Modifica - Visualizza - Aiuto ecc..)
    require('./main-process/main-menu')

    // Creo la cartella Application Support (per Mac) o %APPDATA% (per Windows)
    createApplicationSupportFolder()

    // Create the browser window.
    win = new BrowserWindow({
        /*webPreferences: {
            nodeIntegration: false
        },*/
        width: 1220,
        height: 700
    })

    //Check if is the first time
    var filesDir = getApplicationSupportFolderPath() + 'files' + separator
    fs.stat(filesDir + 'info.json', function (err, stat) {
        if (err == null) {
            // file exists and load the index.html of the app.
            win.loadURL(url.format({
                pathname: path.join(__dirname, 'index.html'),
                protocol: 'file:',
                slashes: true
            }))

        } else if (err.code == 'ENOENT') {
            // file does not exist
            // and load the startup form
            // firstTimeWindows();
            win.loadURL(url.format({
                pathname: path.join(__dirname, 'start.html'),
                protocol: 'file:',
                slashes: true
```

```
            }))
        } else {
            console.log('Some other error: ', err.code);
        }
    });

    // Open the DevTools.
    //win.webContents.openDevTools()

    // Emitted when the window is closed.
    win.on('closed', () => {
        // Dereference the window object, usually you would store windows
        // in an array if your app supports multi windows, this is the time
        // when you should delete the corresponding element.
        win = null
    })
}

function firstTimeWindows() {

    let firstTimeWindows = new BrowserWindow({
        parent: win, // indica che win è la finestra genitore
        modal: true, // disabilita momentanemente la finestra genitore finche' questa e' in vita
        width: 600,
        height: 600,
        /* qua andremo a mettere false nel momento in cui la pubblicheremo
        per il momento lasciamo true così ci è più facile allargare la finestra
        e usare l'ispezione web */
        resizable: true
    })

    firstTimeWindows.loadURL(url.format({
        pathname: path.join(__dirname, 'startup-new.html'),
        protocol: 'file:',
        slashes: true
    }))

    firstTimeWindows.on('ready-to-show', () => {
        firstTimeWindows.show()
    })

    firstTimeWindows.on('closed', () => {
        firstTimeWindows = null
    })
```

```
}

// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
// Some APIs can only be used after this event occurs.
app.on('ready', createWindow)

// Quit when all windows are closed.
app.on('window-all-closed', () => {
  // On macOS it is common for applications and their menu bar
  // to stay active until the user quits explicitly with Cmd + Q
   if (process.platform !== 'darwin') {
      app.quit()
   }
})

app.on('activate', () => {
   // On macOS it's common to re-create a window in the app when the
   // dock icon is clicked and there are no other windows open.
   if (win === null) {
      createWindow()
   }
})

// In this file you can include the rest of your app's specific main process
// code. You can also put them in separate files and require them here.




/**
 * Ritorna il percorso della directory speciale
 * In cui sono collocati i file di supporto all'applicazione
 *
 * Esempio di stringa ritornata:
 * /Users/zermo/Library/Application Support/deskemo/
 */
function getApplicationSupportFolderPath() {
   return app.getPath('appData') + separator + app.getName() + separator
}
```

```
var events = require('events');
var eventEmitter = new events.EventEmitter();


//Create an event handler:
var myEventHandler = function () {
    require('./main-process/mail-listener');
}

//Assign the event handler to an event:
eventEmitter.on('checkemail', myEventHandler);


/**
 * Esporta la funzione così da poter essere utilizzata in altri script
 * quando viene chiamata con il require(main.js)
 */
module.exports = {
    getApplicationSupportFolderPath,
    separator,
    eventEmitter
}
```

5.1.2. MAIL LISTENER

```
var mainProcess = require('../main.js')
const separator = mainProcess.separator;

var Imap = require('imap'),
inspect = require('util').inspect;

const simpleParser = require('mailparser').simpleParser;

var fs = require('fs'),
fileStream;

var firstStart = false;
```

```
//Decriptazione del file json
var filesDir = mainProcess.getApplicationSupportFolderPath() + 'files' + separator;
var encryptedJson = fs.readFileSync(filesDir + 'info.json', 'utf8');
var algorithm = 'aes-256-ctr';
var password = 'd6F3Efeq';
var json = JSON.parse(decrypt(encryptedJson, algorithm, password));


var imap = new Imap({
        user: json['email'],
        password: json['password'],
        host: 'imap.gmail.com',
        port: 993,
        tls: true,
        socketTimeout: 0,
        keepalive: true
});

function openInbox(cb) {
        console.log('Apro la inbox..')
        imap.openBox('INBOX', false, cb);
}


imap.once('ready', function () {
console.log('Verifico le email..')
firstStart = true;
        checkEmail();
});

function checkEmail() {
openInbox(function (err, box) {

var dir = mainProcess.getApplicationSupportFolderPath() + 'email';
var dir_anagrafiche = mainProcess.getApplicationSupportFolderPath() + 'anagrafiche';
//Crea la cartella email se non esiste
if (!fs.existsSync(dir)) {
fs.mkdirSync(dir);
}
//Crea la cartella anagrafiche se non esiste
if (!fs.existsSync(dir_anagrafiche)) {
fs.mkdirSync(dir_anagrafiche);
}
```

```
if (err) throw err;
imap.search(['UNSEEN'], function (err, results) {
if (err) throw err;
if (results.length != 0) {
var f = imap.fetch(results, {
bodies: '',
struct: true,
markSeen: true
});
f.on('message', function (msg, seqno) {
console.log('Message #%d', seqno);

msg.on('body', function (stream, info) {
console.log('Body');
var buffer = '';
stream.on('data', function (chunk) {
buffer += chunk.toString('utf8');
});

stream.on('end', function () {
var messageID = buffer.match("Message-ID: <(.*)>");
console.log('STRING BUFFER:' + messageID[1]);
console.log('END BUFFER');

simpleParser(buffer, (err, mail) => {
var emailAddress = mail.from.text.match("<(.*)>");
var emailAddressDir = mainProcess.getApplicationSupportFolderPath() + 'email' +
separator + emailAddress[1];
var anagraficaDir = mainProcess.getApplicationSupportFolderPath() + 'anagrafiche' +
separator + emailAddress[1];
var emailDir = emailAddressDir + separator + messageID[1];

//Controlla esistenza cartella indirizzo email
if (!fs.existsSync(emailAddressDir))
fs.mkdirSync(emailAddressDir);

var html = mail.html;

//Sostituisci<br> con \n
//var brRegex = /<br\s*[\/]?>/gi
var brRegex = /<\/?br[^>]*>/g
var spanRegex = /<\/?span[^>]*>/g
var divRegex = /<\/?div[^>]*>/g
html = html.replace(brRegex, "\n")
```

```
html = html.replace(spanRegex, "")
html = html.replace(divRegex, "")

//Rimuovi il resto del codice html dalla stringa
var regex = /(<([^>]+)>)/ig;
result = html.replace(regex, "");
result = result.trim();

console.log(result)

//Controlla esistenza cartella anagrafica
if (!fs.existsSync(anagraficaDir)) {
fs.mkdirSync(anagraficaDir);

//Creazione stringa anagrafica da salvare su file
console.log("HTML:" + mail.html);
var bDay = mail.html.match(/\d{2}([\/.-])\d{2}\1\d{4}/g);
var nomeCognome = mail.from.text.match('(.*)<');
var email = mail.from.text.match('<(.*)>');
var codiceFiscale = result.match('Fiscale: (.*)\n');
console.log(bDay + '  <---Date\n' + nomeCognome + '  <---Nome e Cognome\n' + email +
'  <---Email\n' + codiceFiscale + '  <---Cod Fiscale\n')
var anagrafica = "NomeCognome:" + nomeCognome[1] + ";\n" + "Bday:" + bDay[1] + ";\n"
+ "CodiceFiscale:" + codiceFiscale[1] + ";\n" + "IndirizzoEmail:" + email[1] + ";";

//Salvataggio dell'anagrafica nella directory corrispondente
fs.writeFile(anagraficaDir + separator + 'anagrafica.txt', anagrafica, function (err) {
if (err) {
return console.log("Errore nel salvataggio dell'allegato" + err);
}
console.log("The file was saved!");
});

}
//Verifica cartella email corrispondente al message id
if (!fs.existsSync(emailDir)) {
fs.mkdirSync(emailDir);
console.log(mail.headers.get('subject'));


var mailInfo = "FROM:" + mail.from.text + ";\nSUBJECT:" + mail.subject + ";\n\nBODY:" +
result + ";";

//Salvataggio degli allegati
```

```
var arrayLength = mail.attachments.length;
for (var i = 0; i < arrayLength; i++) {
var filename = mail.attachments[i].filename;
var find = ':';
var re = new RegExp(find, 'g');
filename = filename.replace(re, '_');
fs.writeFile(emailDir + separator + filename, mail.attachments[i].content, function (err) {
if (err) {
return console.log("Errore nel salvataggio dell'allegato" + err);
}
console.log("The file was saved!");
});

//Salvataggio del corpo dell'email
fs.writeFile(emailDir + separator + 'emailInfo.txt', mailInfo, function (err) {
if (err) {
return console.log("Errore nel salvataggio dell'allegato" + err);
}
console.log("The file was saved!");
});
}
}
});

/* SALVATAGGIO DELL'INTERO FILE EMAIL -- INUTILE AL MOMENTO
fs.writeFile(emailDir + '/msg-' + seqno + '-body.txt', buffer, function(err) {
if(err) {
return console.log(err);
}

console.log("The file was saved!");
}); */

});
});
});
f.once('error', function (err) {
console.log('Fetch error: ' + err);
});
f.once('end', function () {
console.log('Done fetching all messages!');
//imap.end();
});
}
```

```
});
});

}
imap.once('error', function (err) {
if (err.code === 'ETIMEDOUT') {
console.log('Errore di Timeout\nProvo a riconnettermi..\n')
imap.connect();
} else {
console.log("\nERRORE ")
}
console.log(err);
});

imap.once('end', function () {
console.log('Connection ended');
});

imap.on('mail', function (numNewMsgs) {
console.log('New Mail');
if (firstStart != true)
checkEmail();
firstStart = false;
});

console.log('Effettuo la connessione..\n')
imap.connect();

function decrypt(text, algorithm, password) {
var crypto = require('crypto');
var decipher = crypto.createDecipher(algorithm, password)
var dec = decipher.update(text, 'hex', 'utf8')
dec += decipher.final('utf8');
return dec;
}
```

5.1.3. MAIL CHECK

```
var cImap = require('imap'),
    inspect = require('util').inspect;


var email = $('#email').val();
var psw = $('#password').val();
```

```
var nomeCognome = $('#nomecognome').val();

var checkImap = new cImap({
    user: email,
    password: psw,
    host: 'imap.gmail.com',
    port: 993,
    tls: true
});

function openInbox(cb) {
    checkImap.openBox('INBOX', false, cb);

}

function checkEmail() {
    openInbox(function (err, box) {

    });
}


checkImap.once('error', function (err) {

    $.blockUI({
        message: '<div class="pace-demo" align=center><div class="pace_progress" data-
progress-text="60%" data-progress="60"></div><div class="pace_activity"></div><i
class="icon-alert"></i><br>Attenzione, le credenziali inserite non sono corrette!!</div>',
        timeout: 3000,
        overlayCSS: {
            backgroundColor: '#1b2024',
            opacity: 0.8,
            cursor: 'wait'
        },
        css: {
            border: 0,
            color: '#fff',
            padding: 0,
            backgroundColor: 'transparent'
        }
    });


});
```

```
checkImap.once('ready', function () {
    checkImap.end();
    createJson(nomeCognome, email, psw);
    $.blockUI({
        message: '<div class="pace-demo" align=center><div class="pace_progress" data-
progress-text="60%" data-progress="60"></div><div class="pace_activity"></div><i
class="icon-checkmark"></i><br>Login effettuato con successo</div>',
        timeout: 1000,
        overlayCSS: {
            backgroundColor: '#1b2024',
            opacity: 0.8,
            cursor: 'wait'
        },
        css: {
            border: 0,
            color: '#fff',
            padding: 0,
            backgroundColor: 'transparent'
        }
    });

    setTimeout(function () {
        window.location.href = "index.html";
    }, 1000);
});

checkImap.connect();
```

5.1.4. ENCRYPT

```
// Nodejs encryption with CTR

function encrypt(text, algorithm, password) {
    var crypto = require('crypto');
    var cipher = crypto.createCipher(algorithm, password)
    var crypted = cipher.update(text, 'utf8', 'hex')
    crypted += cipher.final('hex');
    return crypted;
}

function decrypt(text, algorithm, password) {
    var crypto = require('crypto');
    var decipher = crypto.createDecipher(algorithm, password)
```

```
    var dec = decipher.update(text, 'hex', 'utf8')
    dec += decipher.final('utf8');
    return dec;
}
```

5.2. PROCESSI DI RENDERING

5.2.1. PAZIENTI TABLE

```
const mainProcess = require('electron').remote.require('./main.js')
const separator = mainProcess.separator;
var folderAnagrafiche = mainProcess.getApplicationSupportFolderPath() + 'anagrafiche' +
separator
var fileSystemAnagrafiche = require('fs');
var addressDirAnagrafiche = new Array();
var i = 0;
//Memorizza i percorsi delle directory contenute nella directory anagrafiche, nell'array
addressDir
fileSystemAnagrafiche.readdirSync(folderAnagrafiche).forEach(file => {
    // Ignora i file spazzatura tipici di MacOS (.DS_Store)
    if (file.charAt(0) != '.') {
        addressDirAnagrafiche[i] = folderAnagrafiche + file + separator;
        i++;
    }
});


//Riempi la datatable pazienti
for (var i = 0; i < addressDirAnagrafiche.length; i++) {
    var anagrafica = fileSystemAnagrafiche.readFileSync(addressDirAnagrafiche[i] +
'anagrafica.txt', 'utf8');
    var nomeCognome = anagrafica.match('NomeCognome:(.*) ;');
```

```
    var bDay = anagrafica.match('Bday:(.*);');
    var codiceFiscale = anagrafica.match('CodiceFiscale:(.*);');
    var email = anagrafica.match('IndirizzoEmail:(.*);');

    var pazientiTable = $('#pazienti').DataTable();

    pazientiTable.row.add([nomeCognome[1], codiceFiscale[1], bDay[1], email[1], '<button
type="button" class="btn btn-info legitRipple" id="' + addressDirAnagrafiche[i] + '" data-
toggle="modal" data-target="#modal_pazientiInfo"
onclick=setModalContentAnagrafiche(this.id)><i class="icon-user position-left"></i>
Visualizza</button>', '']).draw();

}

//Copia dell'altra funzione in report-table (rivista per i pazienti)
function setModalContentAnagrafiche(path) {

    var pazienteInfo = fileSystemAnagrafiche.readFileSync(path + 'anagrafica.txt', 'utf8');
    var intestazione = pazienteInfo.match("NomeCognome:(.*);");
    var dataDiNascita = pazienteInfo.match("Bday:(.*);");
    var codiceFiscale = pazienteInfo.match("CodiceFiscale:(.*);");
    var indirizzoEmail = pazienteInfo.match("IndirizzoEmail:(.*);");

    var email = path.match('anagrafiche(.*)');
    var emailPath = mainProcess.getApplicationSupportFolderPath() + 'email' + separator
+ email[1];
    var emailDir = new Array();
    var analysis = new Array();

    var i = 0;
    var z = 0;

    //Memorizza i percorsi di tutte le mail ricevute dall'indirizzo corrente, nell'array emailDir
    fileSystemAnagrafiche.readdirSync(emailPath).forEach(file => {
        // Ignora i file spazzatura tipici di MacOS (.DS_Store)
        if (file.charAt(0) != '.') {
            emailDir[i] = emailPath + file + separator;
            i++;
        }
    });

    for (var j = 0; j < emailDir.length; j++) {
        var emailInfo = fileSystemAnagrafiche.readFileSync(emailDir[j] + 'emailInfo.txt',
'utf8');
```

```
    var date = emailInfo.match(/\d{2}([\/.-])\d{2}\1\d{4}/g);
    var aStar = emailInfo.match('[*]: (.*)\n');
    analysis[z] = new Array();
    analysis[z][0] = date[0];
    analysis[z][1] = Number(aStar[1]);
    z++;
}
analysis.sort(sortFunction);

var xArray = [];
var yArray = [];

$.each(analysis, function (index, value) {
    xArray.push(value[0]);
    yArray.push(value[1]);
});
yArray.unshift('data');
$("#modal_title").text(intestazione[1]);
$('#nomeCognomeModalPaziente').text(intestazione[1]);
$('#dataNascitaModalPaziente').text(dataDiNascita[1]);
$('#codiceFiscaleModalPaziente').text(codiceFiscale[1]);
$('#indirizzoEmailModalPaziente').text(indirizzoEmail[1]);

// Generate chart
var line_chart = c3.generate({
    bindto: '#c3-grafico-astar',
    point: {
        r: 4
    },
    size: {
        height: 400
    },
    color: {
        pattern: ['#F4511E']
    },
    data: {
        columns: [yArray],
        names: {
            data: 'A*'
        }
    },
    axis: {
        x: {
            label: 'Tempo',
```

```
            type: 'category',
            categories: xArray
        },
        y: {
            label: 'A*'
        }
    },
    grid: {
        y: {
            show: true
        }
    },
    legend: {
        show: false
    }
});

// Grandissima zozzeria per ridimensionare
// il grafico in base alla grandezza della modal
setTimeout(function () {
    line_chart.resize();
}, 170);
}

function sortFunction(a, b) {
    if (a[0] === b[0]) {
        return 0;
    } else {
        return (a[0] < b[0]) ? -1 : 1;
    }
}
```

5.2.2. REPORT TABLE

```
showLoaderCheckNewReport(true);
const mainProcess = require('electron').remote.require('./main.js')
const separator = mainProcess.separator;
var folderEmail = mainProcess.getApplicationSupportFolderPath() + 'email' + separator;
var fileSystemEmail = require('fs');
var addressDirEmail = new Array();
var readEmail = new Array();
var unreadEmail = new Array();
var i = 0;
```

```
var z = 0;
//Memorizza i percorsi delle directory contenute nella directory email, nell'array
addressDir
fileSystemEmail.readdirSync(folderEmail).forEach(file => {
    showLoaderCheckNewReport(true);
    // Ignora i file spazzatura tipici di MacOS (.DS_Store)
    if (file.charAt(0) != '.') {
        addressDirEmail[i] = folderEmail + file + separator;
        //console.log(addressDirEmail[i])
        i++;
    }
});

//Memorizza i percorsi delle directory delle singole email lette e non lette in due array
distinti
var j = 0;
var k = 0;
for (z = 0; z < addressDirEmail.length; z++) {
    var newFolder = addressDirEmail[z];
    fileSystemEmail.readdirSync(newFolder).forEach(file => {
        //console.log('sono dentro il for')
        showLoaderCheckNewReport(true);
        // Ignora i file spazzatura tipici di MacOS (.DS_Store)
        if (file.charAt(0) != '.') {
            if (fileSystemEmail.existsSync(newFolder + file + separator + "read.txt")) {
                readEmail[k] = newFolder + file + separator;
                //console.log(readEmail[k])
                k++;
            } else {
                unreadEmail[j] = newFolder + file + separator;
                //console.log(unreadEmail[j])
                j++;
            }
        }
    });
}

//Riempi la datatable report con le email non ancora lette
for (var i = 0; i < unreadEmail.length; i++) {
    showLoaderCheckNewReport(true);
    var emailInfo = fileSystemEmail.readFileSync(unreadEmail[i] + 'emailInfo.txt', 'utf8');
    var paziente = emailInfo.match("FROM:(.*) <");
    var emailAddress = emailInfo.match("<(.*)>");
    var info = emailInfo.match("BODY:(.*)\n");
```

```
    var reportTable = $('#report').DataTable();

    // Nascondo la prima colonna della tabella con gli ID
    reportTable.column(0).visible(false)

    // Creo l'ID della mail da inserire nella colonna nascosta con gli ID
    var id_mail = 'mailid' + getIDsenzaCaratteriSpeciali(unreadEmail[i])

    // Prendo tutti i valori della prima colonna con gli ID nascosti
    var valori_colonna = reportTable.column(0).data()

    // Se l'email è già visualizzata nella tabella, non aggiungere nessuna nuova riga
    if ($.inArray(id_mail, valori_colonna) != -1) {
        showLoaderCheckNewReport(false);
        continue;
    }

    var id_label = getIDsenzaCaratteriSpeciali(unreadEmail[i])

    reportTable.row.add([id_mail, paziente[1], emailAddress[1], info[1], '<span id="' +
id_label + '" class="label label-success">Nuova</span>', '<button type="button"
class="btn legitRipple" id="' + unreadEmail[i] + '" data-toggle="modal" data-
target="#modal_emailInfo" onclick=setModalContent(this.id)><i class="icon-enlarge7
position-left"></i> Visualizza</button>', ']).draw();

}

//Riempi la datatable report con le email già lette
for (var i = 0; i < readEmail.length; i++) {
    showLoaderCheckNewReport(true);
    var emailInfo = fileSystemEmail.readFileSync(readEmail[i] + 'emailInfo.txt', 'utf8');
    var paziente = emailInfo.match("FROM:(.*) <");
    var emailAddress = emailInfo.match("<(.*)>");
    var info = emailInfo.match("BODY:(.*)\n");

    var reportTable = $('#report').DataTable()

    // Nascondo la prima colonna della tabella con gli ID
    reportTable.column(0).visible(false)

    // Creo l'ID della mail da inserire nella colonna nascosta con gli ID
    var id_mail = 'mailid' + getIDsenzaCaratteriSpeciali(readEmail[i])
```

```
    // Prendo tutti i valori della prima colonna con gli ID nascosti
    var valori_colonna = reportTable.column(0).data()

    // Se l'email è già visualizzata nella tabella, non aggiungere nessuna nuova riga
    if ($.inArray(id_mail, valori_colonna) != -1) {
        showLoaderCheckNewReport(false);
        continue;
    }

    reportTable.row.add([id_mail, paziente[1], emailAddress[1], info[1], '<span class="label
label-default">Letta</span>', '<button type="button" class="btn legitRipple" id="' +
readEmail[i] + '" data-toggle="modal" data-target="#modal_emailInfo"
onclick=setModalContent(this.id)><i class="icon-enlarge7 position-left"></i>
Visualizza</button>', ]).draw();

}

function setModalContent(path) {
    var emailInfo = fileSystemEmail.readFileSync(path + 'emailInfo.txt', 'utf8');
    var paziente = emailInfo.match("FROM:(.*) <");
    var intestazione = emailInfo.match("SUBJECT:(.*);");
    var emailAddress = emailInfo.match("<(.*)>");
    var info = emailInfo.match("BODY:((.|\n)*);");
    // info[1].html(obj.html().replace(/\n/g,'<br/>'));
    var imageArray = new Array();
    var i = 0;
    fileSystemEmail.readdirSync(path).forEach(file => {
        if (file != "read.txt" && file != "emailInfo.txt" && file.charAt(0) != '.') {
            imageArray[i] = file;
            i++;
        }
    });

    $("#modal_title").text(intestazione[1]);
    $("#modal_body").text(info[1]);

    $("#image1").attr("src", path + imageArray[0]);
    $("#image2").attr("src", path + imageArray[1]);

    // Segna report come letto
    fileSystemEmail.writeFile(path + "read.txt", "AT Software", function (err) {
        if (err) {
            return console.log(err);
        }
```

```
    });

    // Cambia a runtime il label della tabella
    $("#" + getIDsenzaCaratteriSpeciali(path))
        .removeClass('label-success')
        .addClass('label-default')
        .text('LETTA');
}

function getIDsenzaCaratteriSpeciali(id_con_caratteri_speciali) {
    return id_con_caratteri_speciali.replace(/[`~!@#$%^&*()_|+\-=?;:'",.<>\{\}\[\]\\\/\s]/gi, '');
}

function showLoaderCheckNewReport(boolean) {
    if (boolean) { // if true
        $('#check-new-report').show();
    } else { // if false
        $('#check-new-report').hide();
    }
}
```

INDICE DELLE FIGURE

BIBLIOGRAFIA

Denise D. Wilson, “Manual of Laboratory and Diagnostic Tests”, 2007 Ed. McGraw-Hill

S. Suner, G. Crawford, J. McMurdy e G. Jay “Non-invasive determination of hemoglobin by digital photography of palpebral conjunctiva”, 2007

EM. Lackritz, CC Campbell, TK Ruebush II, et al. “Effect of blood transfusion on survival among children in a Kenyan hospital. Lancet” pp. 340-524, 1992.

D. Llewellyn-Jones “Severe anaemia in pregnancy (as seen in Kuala Lumpur, Malaysia). Aust N Z J Obstet and Gynaecol” ,1965.

GW Gardiner, VR Edgerton, B Senewiratne, RJ Barnard RJ, Y. Ohira “Physical work capacity and metabolic stress in subjects with iron deficiency anemia”. Am J Clin Nutr, 1977.

T. N. Sheth, N. K. Choudhry, M. Bowes, A. S. Detsky et al., “The relation of conjunctival pallor to the presence of anemia,” Journal of general internal medicine, vol. 12, no. 2, pp. 102–106, 1997.

R. M. d. Silva and C. A. Machado, “Clinical evaluation of the paleness: agreement between observers and comparison with hemoglobin levels,” Revista Brasileira de Hematologia e Hemoterapia, vol. 32, no. 6, pp. 444–448, 2010.

M. G. N. Spinelli, J. M. P. Souza, S. B. de Souza, and E. H. Sesoko, “Reliability and validity of palmar and conjunctival pallor for anemia

detection purposes," Revista de Saude P ´ ublica ´ , vol. 37, no. 4, pp. 404–408, 2003.

N. Tsumura, N. Ojima, K. Sato, M. Shiraishi, H. Shimizu, H. Nabeshima, S. Akazaki, K. Hori, and Y. Miyake, "Image-based skin color and texture analysis/synthesis by extracting hemoglobin and melanin information in the skin," in ACM Transactions on Graphics (TOG), vol. 22, no. 3. ACM, 2003, pp. 770–779

C. I. Sanchez-Carrillo, T. de Jesus Ramirez-Sanchez, B. J. Selwyn et al., "Test of a noninvasive instrument for measuring hemoglobin concentration," International journal of technology assessment in health care, vol. 5, no. 04, pp. 659–667, 1989.

O. Kim, J. McMurdy, G. Jay, C. Lines, G. Crawford, and M. Alber, "Combined reflectance spectroscopy and stochastic modeling approach for noninvasive hemoglobin determination via palpebral conjunctiva," Physiological reports, vol. 2, no. 1, 2014.

Daniele Bochicchio, Stefano Mostarda, HTML5 Con CSS e JavaScript, Hoepli, 2015, p. 20.

Jeffrey Zeldman, Ethan Marcotte, Sviluppare siti con gli standard web, Milano, Apogeo, 2010.

Mark Pilgrim, HTML5: Guida operativa, Milano, Hops Tecniche nuove, 2010.

Gabriele Gigliotti, HTML5: Sviluppare oggi il Web di domani, Milano, Apogeo, 2012.

Michel Dreyfus: JavaScript (Addison Wesley Longman Italia - 2002)

David Flanagan: JavaScript versione 1.5 (Apogeo - 2002)

Emily A. Vander Veer: JavaScript (con CD-ROM) (Apogeo - 2001)

Roberto Abbate: Imparare JavaScript (Edizioni Master - 2006)

Shelley Powers: Programmare in JavaScript (Tecniche Nuove - 2007)

Douglas Crockford: JavaScript - Le tecniche per scrivere il codice migliore (Tecniche Nuove - 2009)

Briggs, O. – Champeon, S. – Costello, E. – Patterson, M., Cascading Style Sheet (CSS): Fogli di stile per il web, Hoepli Informatica, Milano 2002.

Douglas E. Comer, “The Internet Book: Everything You Need to Know About Computer Networking and How the Internet Works” 3rd ed., 2000, Prentice-Hall.

J. Klensin, ed. “Simple Mail Transfer Protocol” RFC 2821 (2001)

P. Resnick, ed. “Internet Message Format” RFC 2822

Ian Sommerville, Ingegneria del Software, capitoli 22- 23-24

Pressman, Principi di Ingegneria del Software, 5° edizione, Capitoli 15-16

Ghezzi, Jazazeri, Mandrioli, Ingegneria del Software, 2° edizione, Capitolo 6

LIBRERIE UTILIZZATE NELLA REALIZZAZIONE DEL SOFTWARE

Electron - https://electronjs.org

NodeJS - https://nodejs.org

NPM - https://www.npmjs.com

jQuery - https://jquery.com

Data-Driven Documents - https://d3js.org

C3.js D3-based reusable chart library - http://c3js.org

Bootstrap - https://getbootstrap.com

Blockui - http://malsup.com/jquery/block/

RippleJS - http://ripplejs.github.io

DataTables - https://datatables.net

ValidateJS - https://validatejs.org

SweetAlert - https://sweetalert.js.org

Node-Imap - https://github.com/mscdex/node-imap

MailParser - https://github.com/nodemailer/mailparser

Crypto - https://nodejs.org/api/crypto.html