Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

MIGRAZIONE ED INTEGRAZIONE DI UN'APPLICAZIONE WEB IN UN CONTAINER

Relatore:
Chiar.mo Prof.
DANILO MONTESI

Correlatore:
Dott.
FLAVIO BERTINI

Presentata da:
NIKOLAS ACQUAVIVA

Sessione IV
Anno Accademico 2021/2022

Alla città che mi ha permesso
di crescere.

Sommario

Prima di introdurre effettivamente l'attività svolta per il progetto di tesi è importante chiarificare la connotazione dei concetti di **migrazione** ed **integrazione**. Entrambi possono avere molteplici significati se considerati all'interno del settore informatico. Motivo per cui credo che valga la pena descrivere nel dettaglio il vero senso di questi termini nel caso del progetto in questione.
Per raggiungere quest'obiettivo discuterò brevemente alcune delle operazioni svolte durante il processo.

Il progetto di tesi realizzato ha come scopo di inserire un applicativo web denominato "Cogitab" all'interno di un secondo applicativo web che per semplicità chiameremo "applicativo contenitore", il quale include dapprima altre due applicazioni ed agisce come menù da cui l'utente può selezionare una tra le tre applicazioni complessivamente contenute. Una di queste è Cogitab, che rappresenta l'argomento centrale del progetto.

Con il termine **integrazione** si intende l'insieme di operazioni compiute al fine di ottenere non solo l'inserimento di Cogitab nel contenitore, ma anche la corretta unione e uniformazione di alcune funzionalità che le tre applicazioni hanno in comune, e la reciproca comunicazione dei suddetti servizi.

Tuttavia, Cogitab è stato sviluppato su un'infrastruttura web molto diversa da quella dell'applicativo contenitore. Per questo motivo si rende necessaria la fase di **migrazione**. Essa rappresenta il modo più semplice e corretto, nonché lineare, per preparare l'applicazione alla fase di **integrazione**, e consiste nel modificare l'applicazione al fine di uniformarne l'infrastruttura client a quella dell'applicativo contenitore.

Indice

1 Introduzione **1**

2 Panoramica sui web framework: Angular e AngularJS **2**
2.1 AngularJS e MVC 3
2.2 Angular ed i componenti 3

3 La guida ufficiale di Angular sull'upgrade da AngularJS **5**
3.1 Panoramica generale sulla migrazione 5
3.2 AngularJS Style Guide 6
3.3 Migrazione da JavaScript a TypeScript 6
3.4 Component Directives 7
3.5 La libreria ngUpgrade 7
3.6 Il lazy loading di AngularJS 7
3.7 Unificare il servizio Location di Angular 8

4 Revisione sullo stato dell'arte della ricerca sui declini cognitivi in ambito software **9**

5 Lo stato iniziale di Cogitab **12**
5.1 La struttura 12
5.2 L'obiettivo 13
5.3 I moduli e le viste 16

6 Il container prima della migrazione **17**
6.1 L'organizzazione dei moduli 19

7 La migrazione **22**
7.1 La migrazione dei servizi 22
7.2 Il passaggio da controller a component 26
7.3 L'influenza del framework sulla struttura del documento HTML 31

8 L'integrazione **37**
8.1 L'uniformazione della banca dati 37
8.2 La visualizzazione unificata dei risultati 40

9 Conclusioni **42**

10 Bibliografia **43**

11 Sitografia **44**

Elenco delle figure

5.1 Cogitab Root Directory 13
5.2 Struttura dei moduli script di **cogitab_client**: **Controllers e Services** 15
5.3 Le viste di **cogitab_client**: 16

6.1 Il menù dell'applicativo container 18
6.2 Risultati (prima di aggiungere Cogitab) 19
6.3 Cartella *app* del contenitore 20
6.4 Directory *services* 21
6.5 Directory *components* 21
6.6 Directory *pages* (prima di aggiungere Cogitab) 21

7.1 Servizio Timer di Cogitab (versione AngularJS) 23
7.2 Servizio Timer dopo la riformulazione in Angular 24
7.3 Il component *administrator* 26
7.4 Lo script *end.js* di Cogitab (AngularJS) 28
7.5 Lo script *end.component.ts* di Cogitab (Angular) 29
7.6 La vista della sezione findimage di Cogitab (AngularJS) 33
7.7 La vista della sezione findimage di Cogitab (Angular) 34
7.8 La cartella *cogitab* contenente tutti i componenti e servizi ricreati durante la migrazione 35
7.9 La directory pages dopo la migrazione 36

8.1 La tabella per l'anagrafica dei pazienti 38
8.2 Sequenza di azioni all'inizio di un test Cogitab 39
8.3 Risultati di Cogitab (prima dell'unione) 40
8.4 Risultati uniti (alla fine dell'integrazione) 41

Capitolo 1

Introduzione

Al fine di rendere chiaro il metodo perseguito per produrre il contenuto durante il processo di realizzazione del progetto di tesi e per descrivere nel modo più preciso e dettagliato possibile la sequenza delle azioni intraprese per raggiungere tale risultato, fornirò una spiegazione circa la situazione iniziale di entrambe le applicazioni precedentemente nominate per illustrare il metodo di lavoro adottato e la maniera nella quale esse sono cambiate.

Da notare è anche il fatto che sia l'applicativo contenitore sia Cogitab sono applicazioni web e come tali sono fondate su una architettura client-server.

La fase di migrazione riguarda esclusivamente la parte client di Cogitab. Mentre una breve frazione della fase di integrazione è relativa anche alla parte server dell'applicativo contenitore.

Un'ampia sezione del progetto riguarda quindi i web framework sui quali i due applicativi sono stati sviluppati e il modo in cui alcune parti di codice critiche sono state trasformate per passare dal framework originale di Cogitab, AngularJS, a quello del contenitore, Angular.

Capitolo 2

Panoramica sui web framework: Angular e AngularJS

AngularJS non è altro che la versione primitiva ed ormai deprecata dell'attuale Angular.

Infatti, gli sviluppatori hanno portato avanti il progetto Angular stravolgendone la struttura del framework ed anche i principi logici ed ingegneristici in seguito ad un vasto aggiornamento, Angular 2.

Dal punto di vista strutturale, una applicazione web sviluppata in Angular differisce da una sviluppata in AngularJS principalmente a causa del software pattern architteturale applicato. AngularJS utilizza normalmente il pattern **Model-View-Controller**, mentre Angular è parte della categoria di framework basati sui **componenti**.

Tra le altre differenze, vale la pena nominare il linguaggio di programmazione. AngularJS, come suggerito dal nome, utilizza JavaScript mentre Angular è stato sviluppato in TypeScript. Esso è un sovrainsieme di JavaScript, ovvero ne possiede tutte le funzionalità ed ha anche caratteristiche aggiuntive, per esempio un controllo rigido sui tipi. Infatti, una grande differenza nella programmazione tra JavaScript e Typescript è che il secondo è un linguaggio tipato, a differenza di Javascript, perciò incoraggia fortemente la programmazione orientata agli oggetti tramite l'utilizzo di concetti come classi, interfacce, e incapsulamento.
Infine, è importante notare che dei Web Framework come Angular o An-

gularJS sono influenti sui file che riguardano le viste, ovvero alcune sezioni HTML utilizzano delle direttive peculiari del Web Framework utilizzato che permettono di modificare la struttura della vista basandosi sulla qualità dei dati contenuti all'interno del controller o del componente associato, rispettivamente se viene utilizzato AngularJS o Angular.

2.1 AngularJS e MVC

La struttura tipica di un'applicazione AngularJS è rappresentata dalla suddivisione delle sue sezioni in moduli, ognuno dei quali è composto da tre elementi:

- La vista
- Il controller
- Lo stile

In questo modo infatti, per ogni sezione dell'applicativo, la **vista**, scritta in HTML, si occupa di mostrare all'utente i dati elaborati dal **controller**, un file JavaScript. Lo stile viene usato per personalizzare la grafica della vista, nel caso di Cogitab è un file CSS.

Un'applicazione AngularJS può anche utilizzare dei moduli JS ausiliari, chiamati **services** utili ai controller per svolgere delle elaborazioni.

2.2 Angular ed i componenti

I tre elementi soprannominati fanno anche parte dei componenti di Angular, che sono delle direttive che specificano:

- Il selector
- Il file associato alla vista
- Uno o più file per lo stile

Il selector è una stringa che specifica il nome utilizzato come tag HTML personalizzato, che può essere inserito all'interno di un'altra vista e viene renderizzato secondo il codice HTML contenuto nel file specificato dalla stringa relativa al template del componente.

Ogni component non può accedere all'ambiente esterno ed ha il proprio **scope**, che non è accessibile dall'esterno.

Quindi la struttura di un'applicazione Angular è suddivisa in componenti, ognuno dei quali può rappresentare una schermata dell'applicazione o può essere inserito e riutilizzato eventualmente in diverse sezioni tramite l'utilizzo del soprannominato selector.

Capitolo 3

La guida ufficiale di Angular sull'upgrade da AngularJS

Il sito ufficiale del progetto Angular dedica una sezione esclusivamente al passaggio da AngularJS ad Angular fornendo supporto agli sviluppatori che desiderano aggiornare un sito web scritto in AngularJS. Di seguito discuterò quali sono gli step consigliati ufficialmente per una operazione talmente dispendiosa, e nei capitoli successivi verrà descritta la via che ho personalmente deciso di intraprendere per raggiungere il suddetto risultato.

3.1 Panoramica generale sulla migrazione

Nonostante Angular sia l'evoluzione e la versione aggiornata e supportata di AngularJS, le applicazioni progettate sulla prima versione non sono da considerarsi qualitativamente inferiori a quelle sviluppate in Angular, ma semplicemente diverse. Motivo per cui la guida ufficiale consiglia fortemente di considerare il caso specifico e non di migrare ad Angular a prescindere dalla situazione. Nel caso specifico di Cogitab, una migrazione risulta essere necessaria al fine dell'inserimento dell'applicazione all'interno del container, ma in molti altri casi mantenere l'architettura alla versione AngularJS si rivela essere una miglior scelta.

È vivamente consigliato un processo graduale nella migrazione, aggiornando componente dopo componente e mantenendo in parallelo alcuni

componenti scritti in AngularJS e altri in Angular(quelli aggiornati fino a quel momento). Tale modo di procedere rende possibile la realizzazione dell'upgrade anche per applicazioni molto complesse senza rischiare di rovinare alcuna parte del sistema.

3.2 AngularJS Style Guide

Generalmente ci sono applicazioni scritte in AngularJS che sono più semplici da aggiornare di altre. Ciò è causato dall'uso di patterns e buone prassi che se implementate permettono di produrre un codice facile da gestire. Sono raccolte tali metodologie di progettazione e diverse informazioni su come scrivere buon codice in AngularJS nella guida ***"AngularJS Style Guide"***. Una delle regole più importanti secondo tale guida è la *regola dell'uno*. Essa afferma che ci dovrebbe essere un solo componente per file, il che ci permetterebbe di aggiornare ogni componente separatamente uno alla volta, modificandone il linguaggio di programmazione e il framework.

3.3 Migrazione da JavaScript a TypeScript

Nel caso in cui l'upgrade ad Angular comprenda anche l'utilizzo del linguaggio di programmazione TypeScript, installarne il compilatore prima di cominciare la migrazione è cosa buona e giusta. Ciò permetterebbe di cominciare ad utilizzare le peculiarità di TypeScript all'interno del codice AngularJS.

Il passaggio a TypeScript può comprendere alcuni dei seguenti passaggi:

- per applicazioni che usano *module loader* possono essere utilizzati gli import ed export di TypeScript per l'organizzazione del codice in moduli
- può essere realizzata l'aggiunta graduale dei tipi alle variabili e funzioni per godere dei benefici di TypeScript come il controllo sugli errori a tempo di compilazione

- i services e i controllers possono essere modificati riscrivendoli in *classi* in modo da renderli facilmente compatibili all'upgrade

3.4 Component Directives

In Angular i component rappresentano il più importante costrutto sulle quali le interfacce sono costruite. Anche in AngularJS è possibile definire i moduli in *component directives*, che permettono di specificare i controllers e le viste associate al componente. Tale modalità di definizione dei moduli è preferibile rispetto all'utilizzo di caratteristiche a basso livello come le direttive HTML(per esempio *ng-controller*) nei confronti dell'upgrade.

3.5 La libreria ngUpgrade

Lo strumento *ngUpgrade* è utile per l'aggiornamento da AngularJS ad Angular perché permette di mescolare i component di entrambi i framework all'interno della stessa applicazione.

Nella pratica l'utilizzo di tale strumento permette di mandare in esecuzione allo stesso tempo sia AngularJS che Angular. Ciò avviene senza l'implementazione di alcuna emulazione permettendo allo sviluppatore di possedere tutte le peculiarità di entrambi i framework.

3.6 Il lazy loading di AngularJS

Durante lo sviluppo di applicazioni web è necessario che solo le risorse utilizzate, e quindi necessarie, vengano caricate in un dato momento. Tali risorse possono essere file multimediali o codice, e caricarle solo quando necessario permette all'applicazione di funzionare in modo efficace. Questo concetto prende il nome di *lazy loading* ed assume un valore fondamentale nel momento in cui vengono mandati in esecuzione diversi frameworks allo stesso tempo.

Quando si sceglie di migrare una grande applicazione da AngularJS ad Angular utilizzando un approccio ibrido è fortemente consigliata prima la migrazione di funzionalità utilizzate molto comunemente, e l'utilizzo delle funzionalità meno richieste solo quando necessario.

Nei casi in cui entrambi i framework sono utilizzati per renderizzare l'applicazione, entrambi vengono caricati nel pacchetto che viene mandato inizialmente al client causando la creazione di un unico modulo di dimensioni notevolmente incrementate.

Per affrontare questa problematica è possibile isolare l'applicazione AngularJS in un bundle separato il cui caricamento viene posticipato al momento nel quale la parte relativa a tale applicazione risulta necessaria. Questa soluzione permette di massimizzare l'efficienza del sistema evitando il caricamento contemporaneo di entrambi i framework a meno che non sia strettamente necessario.

3.7 Unificare il servizio Location di Angular

In AngularJS viene utilizzato il service *$location* per gestire tutte le configurazioni, navigazioni, codifiche di URL e redirect, tutte utili per l'attività di routing.

Durante la migrazione vale la pena spostare quanto più possibile la responsabilità di tali operazioni verso Angular, cosicché diventi possibile la messa in atto delle nuove funzionalità.

Per assistere lo sviluppatore nella realizzazione di questo compito Angular fornisce il modulo *LocationUpgradeModule* che permette di unificare i servizi di routing trasportando le responsabilità della direttiva *$location* di AngularJS al service *Location* di Angular

Capitolo 4

Revisione sullo stato dell'arte della ricerca sui declini cognitivi in ambito software

Data la notevole crescita del numero di persone anziane riscontranti un disturbo cognitivo è necessario tentare di promuovere uno stile di vita sano con l'intento di ritardare la progressione di declini cognitivi, quali demenza senile, e disturbo cognitivo lieve (MCI). Il numero di persone affette da demenza è destinato a crescere e si stima che raggiungerà il valore di 139 milioni nel mondo nel 2050. Il rapido e avanzato progresso della tecnologia ha aumentato la quantità di interventi informatizzati in campo cognitivo, sopratutto grazie all'utilizzo di **intelligenze artificiali**.

Sapendo che ad oggi non esiste una cura per la demenza, sono state svolte ricerche per identificare i fattori che potrebbero ritardare lo sviluppo di declini cognitivi nelle persone anziane. Un concetto importante è quello di **riserva cognitiva**, la quale è stata descritta come la capacità del cervello di far fronte a cambiamenti associati all'avanzare dell'età o ad infortuni.

Un importante metodo per il miglioramento del funzionamento cognitivo è la **stimolazione cognitiva**. Essa rappresenta un insieme di tecniche e strategie che hanno l'obiettivo di migliorare l'efficacia delle abilità cognitive quali memoria, attenzione e percezione. La stimolazione cognitiva agisce su quelle capacità che sono ancora mantenute andandone a promuovere e ad attivarne il funzionamento.

Infine, la **riabilitazione cognitiva** è un processo che ha l'obiettivo di recuperare capacità cognitive nel caso di deteriorazione di tali causate da patologie mentali.

Molti interventi cognitivi sono stati adattati all'utilizzo su dispositivi tecnologici moderni come smartphone o tablet, essendo tali considerate valide alternative dal punto di vista dei costi rispetto alle metodologie tradizionali. È stato dimostrato inoltre che interventi cognitivi informatizzati hanno effetti benefici sia a breve che a lungo termine su pazienti con funzionalità cognitive mantenute.

Gli interventi informatizzati hanno diversi vantaggi rispetto a quelli tradizionali, tra cui:

- possono essere pensate per essere immersive e piacevoli per i pazienti
- forniscono un feedback istantaneo
- possono essere diretti ad un'abilità cognitiva specifica (per esempio l'attenzione)
- possono essere continuamente aggiornati basandosi sulle performance del paziente

Generalmente per l'implementazione di metodi informatizzati che utilizzano tecniche avanzate di intelligenza artificiale è necessaria una notevole mole di dati per allenare la stessa. Tale mole di dati deve essere valida sia in quantità che in qualità. Nel caso di studio specifico, è necessaria anche la disponibilità di utenti che testino e provino la validità dello studio.

Di seguito realizzerò una nota sintetica su uno degli studi presi in considerazione, il quale utilizza anche il concetto di **cognitive training**. Tale metodo consiste in un insieme di tecniche che hanno l'obiettivo di mantenere le performance delle abilità cognitive ad un livello nella media, oppure di portarle al massimo livello nel caso di capacità mantenute e preservate al livello medio. Questo processo è sottoposto solitamente a una persona anziana oppure ad un atleta o studente che ha l'obiettivo di massimizzare le proprie performance nel campo in cui lavora.

Lo studio discusso ha utilizzato un programma informatizzato per il cognitive training, chiamato Captain's Log. Sono stati studiati i comportamenti di pazienti affetti da demenza o disturbo cognitivo lieve nei confronti del test. I risultati hanno dimostrato che le persone con disturbo cognitivo lieve hanno migliorato il proprio dominio di memoria logica per 8 settimane utilizzando tecniche di stimolazione cognitiva.

Tale studio ha realizzato un'analisi anche sull'utilizzo di diversi programmi che mettono alla prova le capacità cognitive tramite reabilitazione cognitiva e stimolazione cognitiva. In conclusione di tale studio, viene enfatizzato il vantaggio di realizzare tali test in modo informatizzato tramite l'utilizzo di programmi sui diversi tipi di device moderni. Viene data prova del fatto che gli interventi cognitivi automatizzati sono utili all'assistenza verso declini cognitivi come la demenza senile e non solo, infatti è stata trovata utilità anche nei confronti di demenza frontotemporale e demnza vascolare. Ciò denota una notevole flessibilità per quanto riguarda le soluzioni informatizzate per affrontare i declini cognitivi. Inoltre, dato che oggigiorno i dispositivi come tablet o smartphone sono alla portata di tutti, gli interventi cognitivi informatizzati rappresentano una grande agevolazione dal punto di vista dei costi.

Capitolo 5

Lo stato iniziale di Cogitab

Cogitab è una applicazione originariamente sviluppata in JavaScript con AngularJS come web framework. Il lato client è ben separato dal backend, il quale è stato scritto in Python mediante l'utilizzo del backend framework Django. Anche il backend dell'applicativo container utilizza la stessa combinazione di linguaggio di programmazione e framework di Cogitab. Ecco il motivo per cui gran parte del progetto di tesi riguarda il lavoro svolto sui client framework. Infatti l'attività svolta sul backend occupa una minima frazione della fase di integrazione, ma la fase di migrazione riguarda esclusivamente il lato client di Cogitab.

5.1 La struttura

Di seguito viene mostrata l'organizzazione di alcune delle directory principali della precedente versione di Cogitab al fine di chiarificare la situazione presente al momento iniziale dell'attività progettuale svolta.

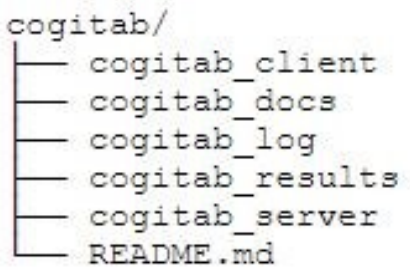

Figura 5.1: Cogitab Root Directory

La cartella **cogitab_server** rappresenta il backend di Cogitab, contiene quindi codice in Python e file di configurazione di Django, nonché i modelli del database. Per quest'ultimo, sia Cogitab che il container utilizzano PostgreSQL DB.

La cartella **cogitab_docs** contiene alcuni file sulla documentazione di Cogitab.

Le cartelle **cogitab_client**, **cogitab_log** e **cogitab_results** sono applicazioni AngularJS separate che gestiscono le tre principali sezioni dell'applicativo.

5.2 L'obiettivo

Cogitab è un sistema pensato per essere utilizzato in campo medico con lo scopo di sottoporre un test al paziente per realizzare una sorta di screening in modo che il medico possa essere supportato nell'eventuale individuazione e prevenzione di un disturbo cognitivo quale la demenza senile.

Il test consiste in una sequenza di minitest, ognuno dei quali ha un punteggio, a cui il paziente deve reagire e sulla base delle risposte fornite viene elaborato il risultato finale come somma totale dei punteggi ottenuti nelle prove sostenute.

L'applicazione **cogitab_client** ospita l'inserimento preliminare delle informazioni del paziente e l'intero test da svolgere.
Per ogni paziente che ha sostenuto almeno un test, vengono mostrati i risultati ottenuti in ognuno di essi all'interno dell'applicazione **cogitab_results**. Mentre nella sezione **cogitab_log** vengono mostrate informazioni sugli accessi come data e tipo di dispositivo.

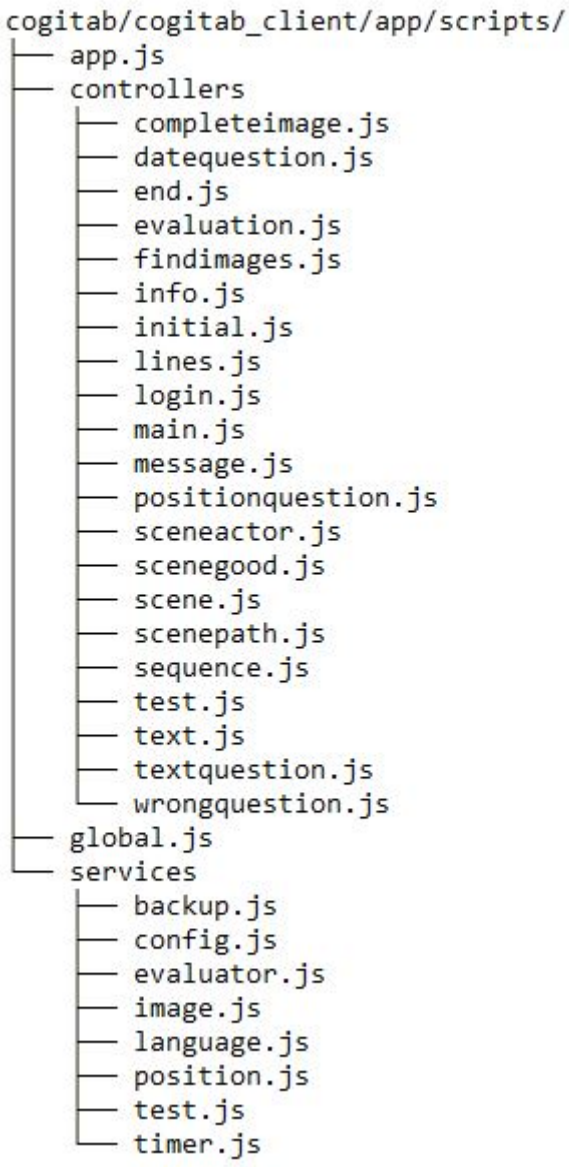

Figura 5.2: Struttura dei moduli script di **cogitab_client**: **Controllers e Services**

5.3 I moduli e le viste

La cartella **controllers** contiene tutti i moduli JavaScript che svolgono le elaborazioni per ogni sezione dell'applicazione **cogitab_client**, ovvero per ogni minitest. Ogni modulo è un controller AngularJS e svolge un ruolo in una certa sezione contenuta nei file HTML, ovvero è possibile specificare il nome di un controller nella direttiva *ng-controller*, che agisce come attributo di un elemento HTML. Così facendo, è possibile utilizzare dei nomi di variabili appartenenti allo scope di quel controller all'interno dell'elemento HTML che ne specifica il nome tramite la suddetta direttiva.

I controller che gestiscono le operazioni dei minitest utilizzano diverse funzioni ausiliarie che permettono di utilizzare un ***timer***, oppure elaborano la ***valutazione*** di un certo minitest. Queste funzioni sono definite all'interno di moduli JavaScript che non specificano alcun tipo di vista HTML. Questo genere di moduli è chiamato **service**, contenuto nella cartella **services** ed agisce nella stessa maniera sia in AngularJS che in Angular.

Infine, ogni controller occupa una frazione di una determinata vista HTML.

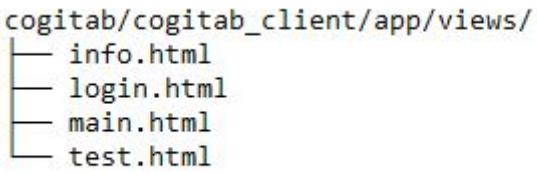

Figura 5.3: Le viste di **cogitab_client**:

Analogamente, anche le applicazioni AngularJS per i log ed i risultati hanno a loro volta dei controllers e delle viste. Tutti i file sono stati presi in considerazione per lo svolgimento della fase di migrazione.

Capitolo 6

Il container prima della migrazione

L'applicativo container è la applicazione Angular all'interno della quale è necessario inserire Cogitab. Il sistema offre la possibilità di scegliere un'applicazione dalla schermata del menù. Ogni applicazione, così come Cogitab, permette di sottoporre un test al paziente ed elabora i risultati di ogni test. Prima dell'inserimento di Cogitab il container include già due sottoapplicazioni, Speechtab e Bloodtab. Speechtab è un tipo di test che richiede al paziente di comunicare verbalmente e basa le valutazioni su aspetti quali il tono della voce, o le pause svolte dal paziente durante l'espressione vocale. Invece Bloodtab non è altro che la visualizzazione di una raccolta di dati basata sugli esami del sangue. Il fine ultimo del progetto di tesi consiste nell'unire Cogitab alle altre due applicazioni tramite l'inserimento dello stesso all'interno del container, ed uniformare il salvataggio dei dati anagrafici dei pazienti con l'obiettivo di produrre per ogni paziente registrato nella banca dati gli esiti dei test sostenuti, permettendo così l'elaborazione di una panoramica complessiva del paziente nei confronti delle diverse prove realizzate.

Perciò, inizialmente la sezione dei risultati di questo applicativo contiene solo i risultati ottenuti nei test Speechtab e Bloodtab. È quindi compito della fase di integrazione quello di unire i risultati di Cogitab a quelli di Speechtab e Bloodtab, solo dopo aver omologato il salvataggio dei dati anagrafici tra le tre applicazioni. Anche quest'ultimo lavoro fa parte dell'attività progettuale svolta, infatti precedentemente i pazienti per Speechtab e Bloodtab venivano salvati manualmente sul database.

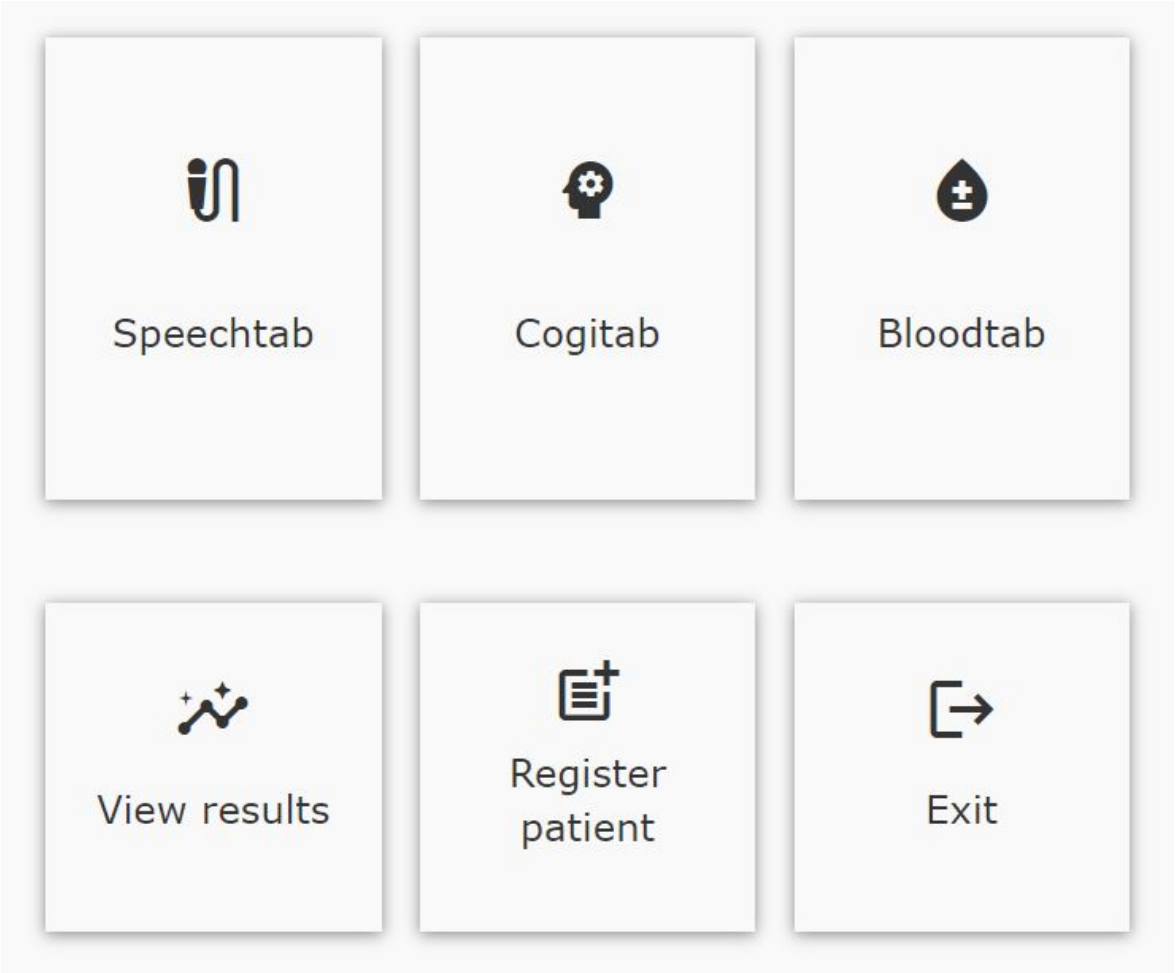

Figura 6.1: Il menù dell'applicativo container

La sezione di Cogitab e della registrazione dei pazienti sono state aggiunte durante lo svolgimento del progetto di tesi.

6.1 L'organizzazione dei moduli

Per ogni schermata dell'applicativo container esiste un componente Angular che ne gestisce l'elaborazione e la visualizzazione dei dati. Inoltre esistono alcuni componenti ausiliari che vengono inseriti all'interno delle viste dei componenti principali per occupare una parte della schermata.

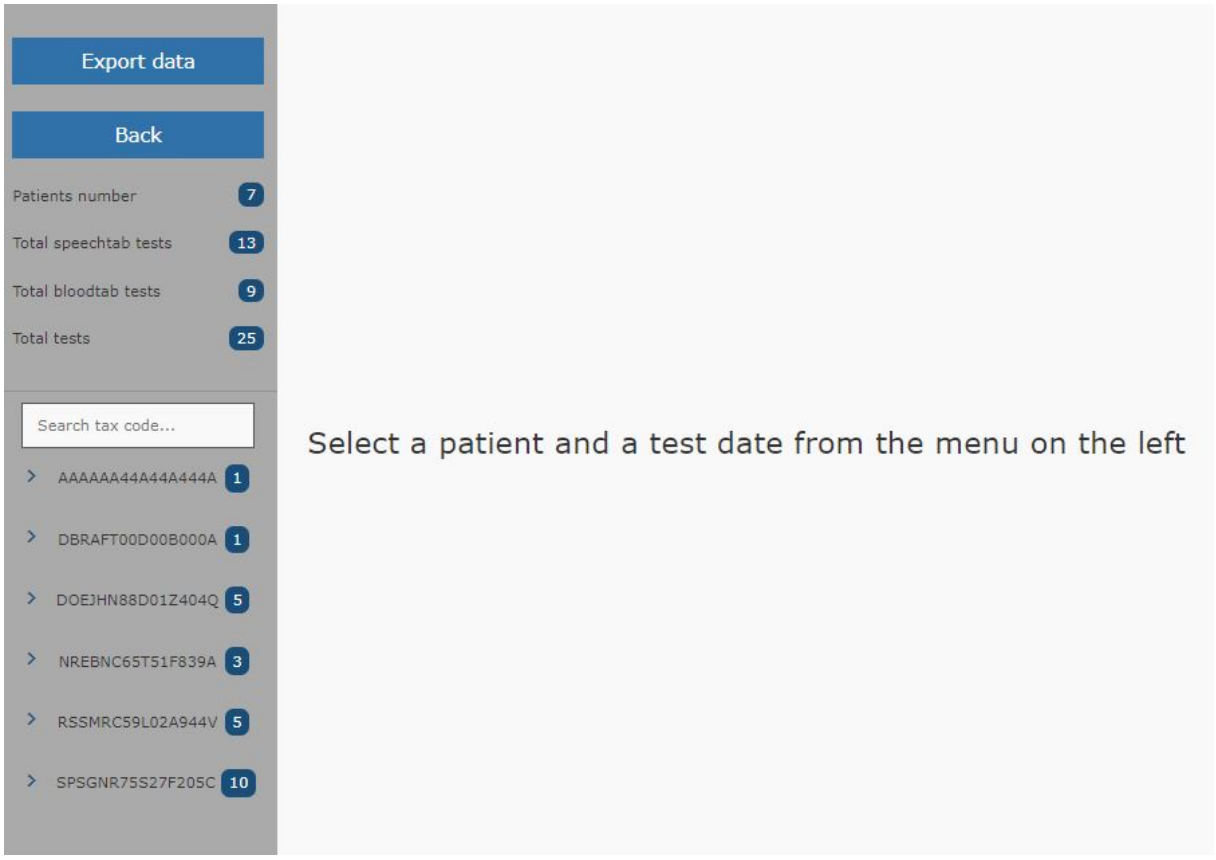

Figura 6.2: Risultati (prima di aggiungere Cogitab)

Per esempio, la vista del componente dei risultati contiene un secondo componente d'ausilio che gestisce esclusivamente il menù laterale sulla si-

nistra.

Così come Cogitab, anche l'applicazione contenitore include una cartella dedicata ai **services**.

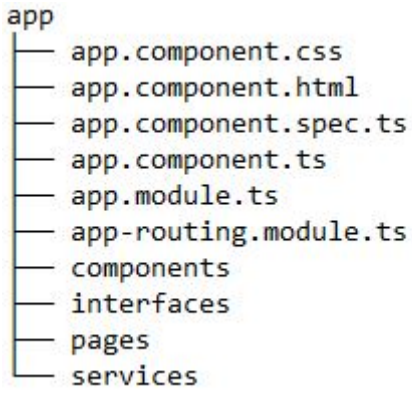

Figura 6.3: Cartella *app* del contenitore

La cartella ***app*** contiene i moduli che rappresentano lo scheletro dell'applicativo. All'interno della cartella ***components*** vi sono i componenti ausiliari come il menù laterale discusso sopra. La cartella ***services*** contiene i servizi, e la cartella ***pages*** contiene i componenti rappresentanti le schermate principali del contenitore e tutte le sottoapplicazioni che contiene.
Come è possibile vedere in figura, quando un componente viene generato esso specifica quattro file:

- La vista
- Lo stile
- Lo script
- Il modulo test (estensione *spec.ts*)

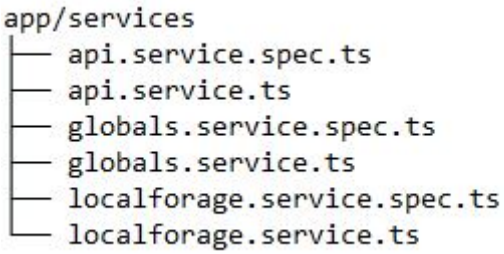

Figura 6.4: Directory *services*

Un servizio Angular a differenza di un componente non specifica un file per la vista.

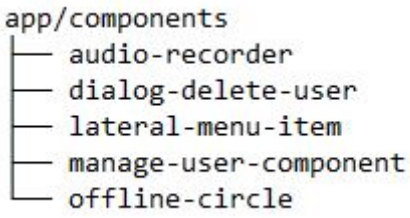

Figura 6.5: Directory *components*

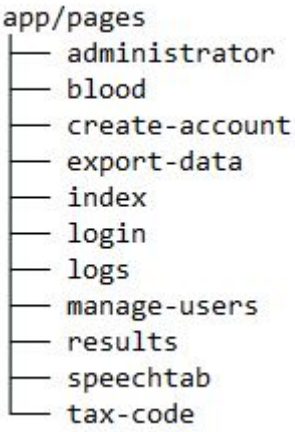

Figura 6.6: Directory *pages* (prima di aggiungere Cogitab)

Capitolo 7

La migrazione

Come discusso in precedenza, la migrazione consiste nel modificare l'infrastruttura di Cogitab al fine di renderla congrua alle necessità di Angular per realizzarne l'inserimento nel container.
Dovendo passare da AngularJS ad Angular, diventa necessario quindi:

1. Riscrivere i services da JavaScript a TypeScript
2. Riscrivere le triple $< Controller - HTML - CSS >$ in dei componenti Angular riscrivendo lo script da JavaScript a TypeScript e specificando nella sua configurazione gli stili(CSS) e la vista(HTML) associata.

7.1 La migrazione dei servizi

Per trasferire la piattaforma di Cogitab da AngularJS ad Angular sono partito dalla riscrittura dei ***services***. Le principali differenze sui servizi che ci sono tra i due framework sono:

1. Il modo in cui avviene la registrazione di un servizio
2. Il paradigma di programmazione Object-Oriented che Angular richiede per lo sviluppo dei principali elementi dell'applicazione

A proposito del secondo punto, Angular infatti definisce una classe per ogni componente e ogni servizio. Essa a sua volta può implementare un

interfaccia od estendere un'altra classe. In diversi casi vedremo implementare l'interfaccia *OnInit* da parte dei componenti. Il metodo ***ngOnInit*** viene richiamato nel momento in cui Angular ha inizializzato le proprietà intrinseche del componente. In altre parole, quando Angular ha terminato l'inizializzazione ed istanziazione del componente stesso. È utile implementare questo metodo nel caso in cui sia necessario inizializzare alcuni campi della classe non appena Angular ne termina le associazioni dopo la creazione.

Di seguito viene mostrato un esempio di riscrittura di un ***service*** da AngularJS ad Angular.

```
angular.module('Cogitab')
        .service('Timer', function Timer() {
                var start, end, running = false;

                this.start = function() {
                        start = new Date().getTime();
                        running = true;
                };

                this.stop = function() {
                    if (running) {
                       end = new Date().getTime();
                       running = false;
                    }
                };

                this.getSeconds = function getSeconds() {
                    var time = Math.floor((end-start) / 1000);
                    return time ? time : 0 ;
                };
        });
```

Figura 7.1: Servizio Timer di Cogitab (versione AngularJS)

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class TimerService {
  startTime: any;
  end: any;
  running: boolean;
  constructor() {
        this.running = false;
  }
  public start() {
        this.startTime = new Date().getTime();
        this.running = true;
  };

  public stop() {
        if (this.running) {
                this.end = new Date().getTime();
                this.running = false;
            }
  };

  public getSeconds() {
                var time = Math.floor((this.end-this.startTime) / 1000);
                return time ? time : 0 ;
  };
}
```

Figura 7.2: Servizio Timer dopo la riformulazione in Angular

Una differenza che si nota subito è la definizione di campi e metodi. In AngularJS entrambi si definiscono come attributi dell'oggetto referenziato da *this*, e possono essere inizializzati senza essere previamente definiti. Invece in Angular gli attributi vanno definiti prima come nomi di classe, e poi inizializzati nei metodi. Questi ultimi vengono definiti come tipicamente fatto nella programmazione orientata agli oggetti.

Per quanto riguarda il service scritto in Angular si può anche notare l'utilizzo di un decorator denominato ***@Injectable***. I decorator sono funzioni speciali che permettono di aggiungere informazioni circa la configurazione di elementi quali component o services. Il decorator ***@Injectable*** permette di comunicare al compiler che una determinata classe può essere importata in un determinato modulo (specificato dall'attributo *providedIn*).

7.2 Il passaggio da controller a component

Così come i services, anche i component sono a loro volta delle classi e dal punto di vista del codice sono molto simili ai servizi.

```
@Component({
  selector: 'app-administrator',
  templateUrl: './administrator.component.html',
  styleUrls: ['./administrator.component.css']
})
export class AdministratorComponent implements OnInit {

  manageAccount: string = "";
  logs: string = "";
  exit: string = "";

  constructor(private localforage: LocalforageService,
              private globals: GlobalsService) { }

  ngOnInit(): void {
    this.globals.isTokenExpired()
    this.localforage.get("languageData").then((data:any) => {
      data = data.administrator
      this.manageAccount = data.manageAccounts;
      this.logs = data.logs;
      this.exit = data.exit;
    })
  }

  logout(){
    localStorage.clear()
  }

}
```

Figura 7.3: Il component *administrator*

Le due differenze con i *services* che subito saltano all'occhio sono:

- Il decorator utilizzato
- L'implementazione dell'interfaccia *OnInit*

Il decorator ***@Component*** permette alla classe di specificare il selector precedentemente discusso, il percorso del file per la vista in *templateUrl* e gli stili in *styleUrls*.
Il metodo ***ngOnInit*** è definito nell'interfaccia *OnInit* ed è utile implementarlo nel caso in cui ci sia bisogno di eseguire delle operazioni non appena il component viene inizializzato.

Di seguito vediamo un esempio sulla differenza tra lo script di uno stesso modulo di Cogitab in Angular ed in AngularJS.

```
angular.module('Cogitab')
   .controller('EndCtrl', function ($scope,
               $location, Test, Language, $interval,Backup) {

      Backup.removeScrollIfIpadChrome();

      $scope.notSaved = Language.get('notSaved');
      $scope.end = Language.get('end');
      $scope.subEnd = Language.get('subEnd');
      $scope.endBtn = Language.get('endBtn');
      $scope.checkSaved = false;

      $scope.next = function () {
         if (Test.allSaved()) {
            Backup.removeBackup();
            window.location.replace('/#/main/' + Language.getSelectedLanguage());
         } else {
            $scope.checkSaved = true;
            var interval = $interval(function () {
               if (Test.allSaved()) {
                   $interval.cancel(interval);
                   var lang =  Language.getSelectedLanguage();
                   Backup.removeBackup();
               }
            }, 1000);
         }
      };
   });
```

Figura 7.4: Lo script *end.js* di Cogitab (AngularJS)

```
@Component({
  selector: 'app-cogitab-end',
  templateUrl: './cogitab-end.component.html',
  styleUrls: ['./cogitab-end.component.css',
              '../cogitab-main/cogitab-main.component.css']
})
export class CogitabEndComponent implements OnInit{
        notSaved: string = "";
        end: string = "";
        subEnd: string = "";
        endBtn: string = "";
        checkSaved: boolean = false;
        constructor(private router: Router, private globals: GlobalsService,
                    private Backup: BackupService, private Test: TestService){
        }

        ngOnInit(): void {
                this.Backup.removeScrollIfIpadChrome();
                this.notSaved = this.globals.cogitab_languageData.notSaved;
                this.end = this.globals.cogitab_languageData.end;
                this.subEnd = this.globals.cogitab_languageData.subEnd;
                this.endBtn = this.globals.cogitab_languageData.endBtn;
        }

        next() {
                if (this.Test.allSaved()) {
                        this.Backup.removeBackup();
                        this.router.navigateByUrl('/cogitab/main');
                }
                else {
                        this.checkSaved = true;
                        var interval = setInterval(() => {
                                if (this.Test.allSaved()) {
                                        clearInterval(interval);
                                        var lang = this.globals.language;
                                        this.Backup.removeBackup();
                                }
                        }, 1000);
                }
        };
}
```

Figura 7.5: Lo script *end.component.ts* di Cogitab (Angular)

È interessante discutere il parallelismo sulla modalità secondo la quale la vista associata al controller, nel caso di AngularJS, o al modulo TypeScript del component, nel caso di Angular, accede ai campi e metodi definiti all'interno del modulo script associato ad essa.

AngularJS definisce un controller tramite la chiamata del metodo ***controller(args...)*** che prende in input una stringa rappresentante il nome da associare al controller e una funzione che a sua volta deve ricevere in input i servizi e le direttive built-in di AngularJS da voler usare all'interno del controller stesso, come per esempio la direttiva ***$http*** che permette di eseguire chiamate HTTP.
Quando si definisce un controller, una delle direttive da utilizzare per mantenere una comunicazione bidirezionale tra script e HTML è ***$scope***. Esso è un oggetto JavaScript i cui campi sono direttamente accessibili dalla vista HTML associata al controller mediante l'utilizzo dell'attributo ng-controller.
Analogamente, in Angular la vista HTML accede ai campi e metodi definiti dalla classe che specifica il decorator *@Component*.
All'interno del file per la vista HTML di un component è sufficiente specificare il nome di un attributo o di un metodo della classe rappresentante il component associato ad essa.

Nell'esempio del controller in figura definito in *end.js*, viene definita una funzione assegnata al campo *next* dell'oggetto ***$scope***. Quindi la frazione HTML all'interno dell'elemento che specifica come attributo ***ng-controller*** il valore *'EndCtrl'* potrà richiamare tale funzione associandola per esempio ad un evento di tipo *click* tramite l'attributo HTML peculiare di AngularJS, ***ng-click***.

Questo genere di attributi HTML built-in di web framework come AngularJS o Angular, e sopratutto i modi in cui si sono evoluti durante il passaggio dal primo al secondo, sono parte dell'attività di migrazione e sarà discusso in maggior dettaglio all'interno della prossima sezione.

7.3 L'influenza del framework sulla struttura del documento HTML

Angular e AngularJS forniscono delle direttive che possono agire come attributi di elementi HTML al fine di aggiungere potere espressivo alla renderizzazione del documento. Siccome i nomi e le modalità di utilizzo delle suddette direttive sono cambiati al momento dell'aggiornamento della piattaforma, da AngularJS ad Angular, è parte della fase di migrazione aggiornare anche i file HTML per le viste di Cogitab affinché la sintassi si attenga alle regole fornite dalla versione aggiornata.

Di seguito descriverò alcune delle direttive più utilizzate in Cogitab e come sono cambiate dopo la migrazione, ovvero portandole sulla piattaforma Angular.

- **ng-if**: in base al valore di verità dell'argomento mostra o meno il codice HTML interno
- **ng-repeat**: itera sugli elementi della struttura dati passata in input
- **ng-click**: Esegue il codice descritto dall'argomento
- **ng-show**: Mostra o nasconde il codice HTML interno in base al valore di verità dell'argomento

Viene mostrato di seguito l'aggiornamento di questi costrutti:

- **ng-if** si riscrive in ***ngIf**
- **ng-repeat** si riscrive in ***ngFor**
- **ng-click** si riscrive in **(click)**
- **ng-show** si riscrive in [**hidden**] ed inoltre è necessario negare l'espressione originale. Questo poiché [**hidden**] nasconde il codice HTML contenuto all'interno se l'espressione risulta essere *vera*, ovvero il comportamento opposto di **ng-show**

Per concludere questo capitolo illustrerò la stessa vista di una schermata di Cogitab scritta sia in AngularJS che in Angular.

```
<section ng-controller="FindimagesCtrl">
         <div ng-if="intro" class="slide slide-center">
            <h1>{{introText}}{{rights.length == 1 ? thisImg : these}}:</h1>
            <div>
               <img ng-repeat="right in rights" ng-src="media/{{right.image}}"
                    class="findimages-sample centered preview"/>
            </div>
            <h3>{{canListen}}
               <a href="" ng-click="play()">
                  <img src="/media/imgs/audio.png" class="audioIcon" />
               </a>
            </h3>
            <button class="btn btn-big btn-primary" ng-click="start()">
               {{introBtn}} <span class="glyphicon glyphicon-play"></span>
            </button>
         </div>
         <div ng-show="!intro && !timeout" class="slide slide-surface">
            <div id="canvas-container">
               <canvas id="canvas" class="full-canvas" resize></canvas>
            </div>
            <button class="btn btn-next btn-primary" ng-click="next()">
               {{nextBtn}} <span class="glyphicon glyphicon-play"></span>
            </button>
         </div>
         <div class="slide slide-center" ng-if="timeout">
            <h1>{{timeoutText}}</h1>

            <h3>{{subTimeout}}</h3>
            <button class="btn btn-big btn-primary" ng-click="timeout_next()">
               {{nextBtn}} <span class="glyphicon glyphicon-play"></span>
            </button>
         </div>
</section>
```

Figura 7.6: La vista della sezione findimage di Cogitab (AngularJS)

```
<div *ngIf="intro" class="slide slide-center">
       <h1>{{introText}}{{rights.length == 1 ? thisImg : these}}:</h1>
       <div>
            <img *ngFor="let right of rights" src='{{ "media/" + right.image }}'
                class="findimages-sample centered preview"/>
       </div>
       <h3>{{canListen}}
            <a href="javascript:void(0)" (click)="play()">
               <img src="/media/imgs/audio.png" class="audioIcon" />
            </a>
       </h3>
       <button class="btn btn-big btn-primary" (click)="start()">
            {{introBtn}} <span class="glyphicon glyphicon-play"></span>
       </button>
</div>

<div [hidden]="!(!intro && !timeout)" class="slide slide-surface">
        <div id="canvas-container">
           <canvas id="canvas" class="full-canvas" resize></canvas>
        </div>
        <button class="btn btn-next btn-primary" (click)="next()">
           {{nextBtn}} <span class="glyphicon glyphicon-play"></span>
        </button>
</div>
<div class="slide slide-center" *ngIf="timeout">
        <h1>{{timeoutText}}</h1>
        <h3>{{subTimeout}}</h3>
        <button class="btn btn-big btn-primary" (click)="timeout_next()">
           {{nextBtn}} <span class="glyphicon glyphicon-play"></span>
        </button>
</div>
```

Figura 7.7: La vista della sezione findimage di Cogitab (Angular)

Un'ulteriore differenza che vale la pena sottolineare è che nella versione AngularJS è specificato il controller tramite la direttiva associata *ng-controller* così che i dati utilizzati all'interno dell'elemento HTML si riferiscano ai campi dell'oggetto *$scope* del controller specificato.

In Angular questo comportamento è ottenuto tramite la specificazione della vista associata al component mediante l'utilizzo del decorator ***@Component***.

A seguire una visione sulla struttura dell'applicativo container al momento della terminazione della fase di migrazione.

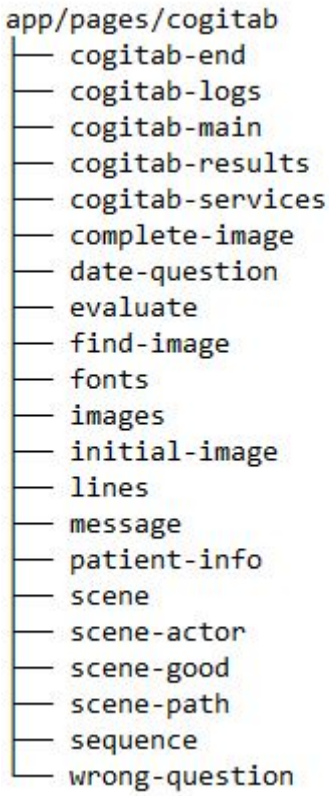

Figura 7.8: La cartella *cogitab* contenente tutti i componenti e servizi ricreati durante la migrazione

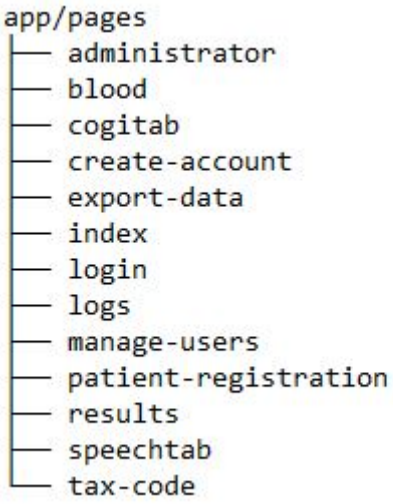

Figura 7.9: La directory pages dopo la migrazione

Capitolo 8

L'integrazione

Una volta terminata la fase di migrazione e quindi l'inserimento di Cogitab nel container, l'obiettivo finale è quello di omologare e standardizzare le tre applicazioni, Cogitab, Speechtab e Bloodtab, in modo da rendere unica la banca dati nella quale vengono registrati i pazienti e, per ogni paziente, rendere possibile la visualizzazione congiunta dei risultati prodotti dai test delle diverse applicazioni.

8.1 L'uniformazione della banca dati

Inizialmente Cogitab non aveva una collezione di dati per le informazioni dei pazienti che svolgevano i test, ma solo una tabella che permette di salvare le informazioni relative ad un dato test. Quindi oltre ai dati su un certo test come la valutazione finale dello stesso, un record di dati per quel test contiene anche le informazioni relative al paziente che l'ha eseguito. In altre parole, i dati anagrafici dei pazienti che hanno svolto un test di Cogitab venivano salvati direttamente nello stesso record nel quale venivano salvate le altre informazioni relative a quello stesso test, e non in una tabella contenente esclusivamente i dati anagrafici dei pazienti registrati.

In questo modo, tutti i dati anagrafici venivano quindi richiesti all'inizio dello svolgimento di un test Cogitab, rendendo possibili delle ambiguità su tali dati.

Essi sono:

1. **Codice fiscale**
2. **Data di nascita**
3. **Sesso**
4. **Anni di scuola frequentati**
5. **Presenza di aspetti ansiosi, depressivi o familiarità con demenza**

Erano possibili quindi ambiguità come per esempio coppie di dati di tipo < *Codice fiscale* - *Data di nascita* > incongruenti tra loro.

Invece in Speechtab e Bloodtab esisteva già una tabella contenente esclusivamente i dati anagrafici per i pazienti registrati, ma essa veniva popolata manualmente tramite l'utilizzo di comandi SQL per l'inserimento di dati nel database.

patient_id	surname	name	tax_code	birthdate	male	max_school	school_years
4	Asfalti	Debro	DBRAFT00D00B000A	1989-10-29	t	Università	18
5	Belcri	Armando	ARDBLC00A00B000C	1999-05-03	t	Scuola superiore	13
7	test	test	AAAAAA00A00A000A	2000-03-12	t	Scuola media	8

Figura 8.1: La tabella per l'anagrafica dei pazienti

La tabella in figura rapprensenta l'attuale schema per il salvataggio dei dati anagrafici dei pazienti. Essa ha subito alcune modifiche per soddisfare alcuni vincoli sui salvataggi dei dati di Cogitab.

Con l'assunzione che i dati richiesti al paziente relativi al punto 5 della lista sopra descritta possano cambiare nel tempo, tali dati non sono stati inseriti invece nella sezione della banca dati riguardante i dati anagrafici dei pazienti registrati. Essi vengono invece richiesti ogni volta che un paziente svolge il test di Cogitab. Diversamente, tutti gli altri dati, tranne

il codice fiscale, non sono più richiesti prima di cominciare il test di Cogitab, ma vengono richiesti nella sezione *Registrazione Paziente* che è stata aggiunta all'applicativo durante questa fase.

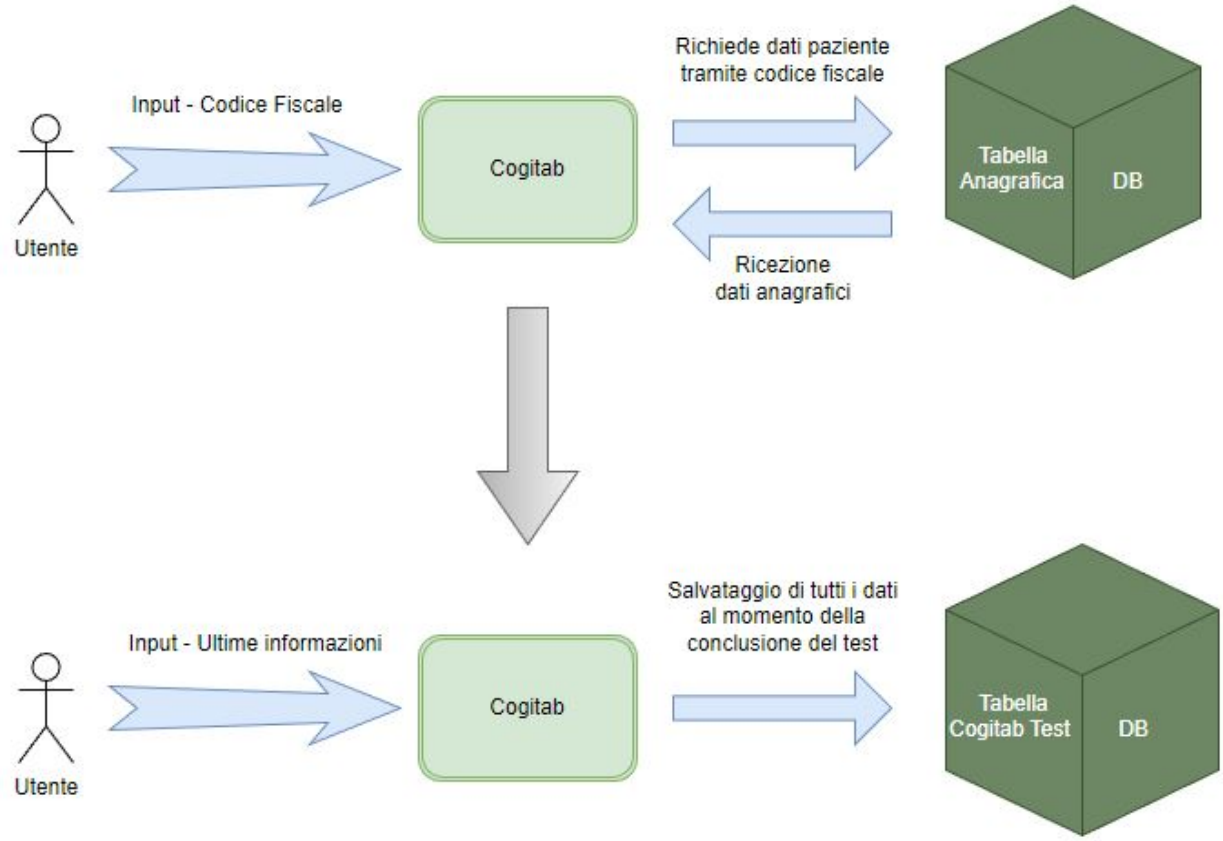

Figura 8.2: Sequenza di azioni all'inizio di un test Cogitab

Perciò dopo l'aggiunta della sezione per la registrazione dei pazienti, quando un paziente comincia un test di Cogitab ha bisogno di inserire solamente il codice fiscale, utilizzato per ricevere tramite una chiamata al database gli altri dati anagrafici e scolastici dalla tabella delle anagrafiche, e le informazioni sugli aspetti depressivi, ansiosi e sulla presenza o meno di familiarità con demenza. Dopodiché tutti questi dati vengono raccolti per salvare il test Cogitab sulla tabella relativa al salvataggio dei test, poiché quest'ultima richiede anche i dati anagrafici del paziente oltre alle informazioni sul test stesso.

8.2 La visualizzazione unificata dei risultati

Nel momento che precede lo svolgimento dell'integrazione sono presenti quindi due sezioni distinte per i risultati, una per Speechtab-Bloodtab ed una per Cogitab. Permettere all'utente di visualizzare per un dato paziente gli esiti dei vari tipi di test svolti è l'obiettivo finale del progetto di tesi.

Per ottenere tale risultato è necessario integrare la vista e lo script del component per la sezione dei risultati di Cogitab nel component per la sezione dei risultati di Speechtab. Inoltre sono stati modificati alcuni campi degli oggetti definiti per mostrare i test di Cogitab in modo da renderlo compatibile all'oggetto creato per mostrare i risultati utilizzato dal component di Speechtab.

Figura 8.3: Risultati di Cogitab (prima dell'unione)

Figura 8.4: Risultati uniti (alla fine dell'integrazione)

Capitolo 9

Conclusioni

In conclusione, l'obiettivo del progetto rappresentato dall'unione di Cogitab a Speechtab e Bloodtab mediante la visualizzazione congiunta dei risultati dei diversi tipi di test e l'unificazione delle banche dati relative alle anagrafiche dei pazienti è stato conseguito successivamente al riuscito inserimento di Cogitab nel contenitore. Tale risultato è stato raggiunto tramite la traduzione di ogni schermata di Cogitab nell'analoga versione riscritta secondo le richieste del framework Angular, modificando quindi non solo il pattern software di progettazione ed il linguaggio di programmazione ma anche i servizi, i controller divenuti component e le direttive HTML aggiunte dalle piattaforma di sviluppo AngularJS ed Angular.

Il lavoro svolto sul lato backend è confinato ad una leggero cambiamento nella tabella del database dedicata alle anagrafiche con lo scopo di soddisfare le esigenze del salvataggio dei test di Cogitab. Le modifiche apportate a tale tabella rappresentano l'azione necessaria al corretto sviluppo della sezione di registrazione pazienti.

Capitolo 10

Bibliografia

1. Gates, Nicola J., et al. "Computerised cognitive training for preventing dementia in people with mild cognitive impairment." Cochrane Database of Systematic Reviews 3 (2019).

2. García-Casal, J. Antonio, et al. "Computer-based cognitive interventions for people living with dementia: a systematic literature review and meta-analysis." Aging & mental health 21.5 (2017): 454-467.

3. Contreras-Somoza, Leslie María, et al. "Usability and user experience of cognitive intervention technologies for elderly people with MCI or dementia: a systematic review." Frontiers in Psychology 12 (2021): 636116.

4. Irazoki E, Contreras-Somoza LM, Toribio-Guzmán JM, Jenaro-Río C, van der Roest H and Franco-Martín MA (2020) Technologies for Cognitive Training and Cognitive Rehabilitation for People With Mild Cognitive Impairment and Dementia. A Systematic Review. Front. Psychol. 11:648. doi: 10.3389/fpsyg.2020.00648

5. Breton, Alexandre, Daniel Casey, and Nikitas A. Arnaoutoglou. "Cognitive tests for the detection of mild cognitive impairment (MCI), the prodromal stage of dementia: Meta-analysis of diagnostic accuracy studies." International journal of geriatric psychiatry 34.2 (2019): 233-242.

Capitolo 11

Sitografia

1. Documentazione Angular: `https://angular.io/docs`
2. Migrazione AngularJS - Angular:
   `https://angular.io/guide/upgrade`
3. AngularJS Style Guide:
   `https://github.com/johnpapa/angular-styleguide`
4. Documentazione TypeScript:
   `https://www.typescriptlang.org/docs`
5. Documentazione Django:
   `https://docs.djangoproject.com/en/4.1`
6. Stimolazione cognitiva:
   `https://www.bitbrain.com/blog/cognitive-stimulation`
7. Captain's Log:
   `https://www.braintrain.com/captains-log-mindpower-builder`

Ringraziamenti

Voglio ringraziare la città di Bologna che mi ha permesso di intraprendere un cammino che mi ha formato non solo didatticamente, ma mi ha anche permesso di vivere una notevole crescita dal punto di vista personale.

A tal proposito desidero dedicare un particolare ringraziamento a tutti i compagni della residenza nella quale ho alloggiato durante quest'esperienza. Con loro ho avuto il piacere di condividere quest'importante obiettivo, che con il tempo è divenuto un punto cardine del mio percorso.
Tra di essi, un ringraziamento speciale va a Luca, il mio compagno di stanza, con il quale ho trascorso sia momenti positivi che negativi, ed ho condiviso ogni obiettivo raggiunto durante questo viaggio.