UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di Laurea in Informatica

Applicazione Web per la Gestione di Dati Neuroscentifici: Studio e Implementazione di un Document Database NoSQL

Relatore: Riccardo Martoglia

Laureando: Luca Sala

Anno accademico 2018/2019

RINGRAZIAMENTI

Ringrazio l'Ing. Riccarco Martoglia, mio relatore, per la grande disponibilità e per il suo aiuto durante l'arco di tutto il percorso.

Ringrazio la Dottoressa Giovanna Zamboni e il Professor Giuseppe Pagnoni per avermi dato la possibilità di sviluppare un progetto per una ricerca importante.

Infine ringrazio i miei genitori e la mia ragazza per avermi sempre supportato e creduto in me.

PAROLE CHIAVE

Base di Dati

NoSQL

MongoDB

Flask

Python

Sommario

Parte I

Il caso di studio

Introduzione

Il termine Mild Cognitive Imparment (MCI), tradotto come “compromissione cognitiva lieve”, è utilizzato per descrivere una condizione di deficit cognitivo (maggiore rispetto alla media per una determinata età) diagnosticata in soggetti che non hanno passato la soglia d’età per la demenza.

Questa tesi ha lo scopo di risolvere il grande problema della memorizzazione dei dati raccolti per lo studio condotto dall’Università di Modena e Reggio Emilia.
Prima della soluzione progettata e implementata, la quale sarà il fulcro della tesi, le informazioni ricavate dai pazienti attraverso diversi esami, venivano memorizzate in maniera molto spartana.
Alcuni risultati di esami venivano memorizzati all’interno di file Excel con la problematica aggiuntiva dell’utilizzo di schemi diversi tra i ricercatori di Modena e ricercatori di Reggio Emilia.
In altri casi, i risultati erano mantenuti in stato cartaceo rendendo impossibile a colleghi lontani di poterne prendere visione.

Il gruppo di ricercatori, tenendo conto delle problematiche presenti, ha espresso la volontà di voler creare un sistema di memorizzazione dei dati che semplificasse il lavoro svolto, in modo da rendere i dati sempre disponibili ovunque e a tutte le persone coinvolte nel progetto.
È stata quindi richiesta la creazione di un sistema di memorizzazione che permettesse a un ricercatore, munito di una connessione ad Internet e credenziali di accesso, di inserire, visualizzare e scaricare tutti i dati relativi alla ricerca.

Questa tesi ha avuto come scopo la creazione di tale sistema, dividendo in capitoli quello che è stato il processo che ha portato alla sua conclusione.

Più nello specifico, si parte dal primo capitolo prendendo in considerazione tutti i requisiti che sono stati elaborati grazie a incontri avvenuti con il gruppo dei ricercatori. All’interno del secondo capitolo si vanno ad analizzare quelle che sono le tecnologie prese in considerazione per l’adempimento del progetto. Tramite il terzo capitolo, è possibile seguire passo per passo quello che è stato il processo di progettazione. A questo segue un capitolo di implementazione, il quale mostra l’effettiva creazione delle soluzioni pensate nel capitolo ad esso precedente.
Infine verranno tratte conclusioni e si valuteranno quelli che potrebbero essere gli sviluppi futuri.

1. Requisiti

In questo capitolo sarà descritto il progetto di tesi e ne verranno valutati tutti gli aspetti fondamentali. Si elencheranno quelle che sono le esigenze, i vincoli progettuali e le scelte effettuate in base a questi ultimi. In questo modo si riusciranno a comprendere i passaggi che hanno preceduto la realizzazione del progetto senza entrare in dettagli implementativi, i quali verranno trattati nel terzo capitolo.
In un secondo momento saranno esposti brevi concetti teorici che serviranno a capire appieno gli argomenti trattati all'interno di questa tesi.
A questo capitolo seguirà quello dello studio delle tecnologie.

1.1. Richiesta iniziale

Il progetto iniziale prevedeva la realizzazione di una Web Application affiancata da un database per la memorizzazione e gestione di dati relativi allo studio Mild Cognitive Impairment (MCI), studio realizzato dall'ospedale di Baggiovara in collaborazione con l'ospedale di Reggio Emilia.

Nello specifico, è stato richiesto un sistema che agevolasse i vari gruppi coinvolti all'interno dello studio a immettere, visualizzare, salvare e scaricare tutti i dati necessari tramite la stessa interfaccia, costruita in base alle esigenze.
I gruppi individuati per questo progetto sono i seguenti: ambito clinico, ambito neurologico, ambito epidemiologico e ambito imaging.
Prima di realizzare questo progetto, i dati venivano salvati su normali fogli di calcolo, fogli cartacei e altri sistemi impossibili da collegare tra loro. Lo scopo finale è stato quindi quello di unire tutti i dati per rendere di fatto questi accessibili in modo facile e veloce.

Di seguito verranno esposte le esigenze progettuali, per quanto riguarda i diversi campi di ricerca, raccolte nei vari incontri avvenuti all'interno dell'università (UNIMORE) e all'ospedale di Baggiovara (MO).

1.1.1. Ambito Clinico

Per l'ambito Clinico, le persone di riferimento sono state la Dottoressa Giovanna Zamboni e la Dottoranda Chiara Carbone.

Prima della realizzazione di questo progetto, i dati clinici venivano salvati soltanto all'interno di file Excel. In questo modo era ovviamente ostacolata e resa più macchinosa la condivisione con tutti gli ambiti di ricerca.
La richiesta è stata quella di poter inserire i dati tramite un'interfaccia semplice ed intuitiva dividendo le informazioni cliniche da quelle demografiche in due pagine diverse.
È stata inoltre richiesta la presenza dei dati relativi a due esami di ambito clinico: Blood e Cerebro Spinal Fluid (CSF), immessi in parte da persone del campo clinico e in parte da persone del campo epidemiologico.
Un'ulteriore richiesta è stata quella di poter scaricare i dati in formato CSV in modo da permettere l'elaborazione di essi attraverso software terzi.
La lista dei dati da memorizzare per quanto riguarda le informazioni cliniche e demografiche è stata fornita insieme ad una legenda e a vincoli da rispettare. In questo modo è stato possibile realizzare tutti i controlli opportuni per evitare l'inconsistenza dei dati in fase di inserimento (es: range di valori ammissibili).

1.1.2. Ambito Neurologico

Per l'ambito Neurologico, le persone di riferimento sono state la dottoranda Chiara Carbone e la Professoressa Giovanna Zamboni.

Per questo ambito, le richieste sono state pressoché identiche a quelle dell'ambito Clinico. Lo scopo è stato quindi quello di permettere a chi dovesse inserire i dati di poterlo fare in modo semplice ed intuitivo e di scaricare in formato CSV i dati per poterli elaborare ed analizzare in un secondo momento attraverso programmi terzi.
La pagina contenente i dati relativi al campo neuroscientifico è stata chiamata "NPS"

(abbreviativo di “neuropsicologica”).

1.1.3. Ambito Epidemiologico

Per quanto riguarda l’ambito Epidemiologico, il Ricercatore Tommaso Filippini e il Professore Marco Vinceti sono state le persone di riferimento.

Si possono dividere i dati trattati da questo ambito in due sezioni separate: la prima riguardante due questionari e la seconda riguardante i dati di esami “Blood” e “Cerebro Spinal Fluid” (CSF).

I questionari che sono stati trattati sono due:

- Il primo, riferito con il nome “life style” (ls), riguarda lo stile di vita del paziente preso in esame. Il questionario viene sottoposto al paziente in formato cartaceo e viene successivamente trascritto in un file CSV per l’elaborazione.

- Il secondo, riferito con il nome “Epic”, riguarda le abitudini alimentari. Il questionario viene sottoposto anch’esso in formato cartaceo, ma per ottenere le variabili adatte per l’elaborazione vi è la necessità di passare tramite un programma che gira soltanto su MS-DOS.

Il problema del secondo questionario è quello di dover trasferire i dati in un programma presente su una macchina con sistema operativo MS-DOS in modo da ricevere in output un file da poter inviare ad un istituto di Milano per la ricerca. Oltre ad essere ormai un sistema obsoleto, questo procedimento non lascia margine d’errore poiché non vi è la possibilità di modifica dei dati inseriti rendendo di fatto la procedura di immissione lenta. La richiesta principale è stata quindi quella di poter memorizzare i valori delle singole variabili dei due questionari all’interno del database, per poterle recuperare (nel formato richiesto) e modificare senza alcun tipo di problema in qualsiasi momento e da qualsiasi postazione con un collegamento al server.
Le variabili dei due questionari sono state fornite tramite due file CSV in modo da facilitarne l’immissione.

Una richiesta secondaria è stata quella di caricare i dati direttamente da un altro file CSV in modo da facilitare l'immissione di tutti i questionari che sono già stati sottoposti ai pazienti.

Per quanto riguarda gli esami "Blood" e "CSF", dovranno essere memorizzati insieme ai dati inseriti dal personale clinico.
Questi dati dovranno poter essere scaricati in un secondo momento.

1.1.4. Ambito Imaging

Per quanto riguarda l'ambito imaging, i requisiti e i vincoli progettuali sono stati forniti dal Professore Giuseppe Pagnoni.

L'ultima parte definita è stata quella relativa all'imaging, la quale racchiude due esami: Positron-Emission Tomography (PET) e Magnetic Resonance Imaging (MRI).

- Per quanto riguarda l'esame PET la richiesta è stata quella di poter memorizzare una stringa contenente la directory delle immagini, un numero indefinito di file e un campo note opzionale per ogni file caricato.
  Purtroppo, non è stato possibile creare un collegamento diretto con le immagini soprattutto per motivi di privacy.

- Per quanto riguarda il secondo esame (MRI) è stato richiesto di poter memorizzare un numero anche in questo caso indefinito di file con relativa descrizione facoltativa per ogni "flag" dell'esame. Un flag racchiude parti differenti dello stesso esame.

Per entrambi gli esami, è stata richiesta la possibilità di scaricare i file caricati in modo da svolgere successivamente delle elaborazioni tramite programmi di terzi.

1.2. Specifiche generali

In questo paragrafo seguono le specifiche generali che non dipendono dal tipo di esame.

1.2.1. Richieste grafiche

Le richieste per quanto riguarda la parte grafica sono state principalmente due.

- La prima richiesta è stata quella di creare un'interfaccia semplice ed intuitiva, utilizzabile da chiunque.
- La seconda, suggerita dalla Dottoressa Giovanna Zamboni, è stata quella di creare una struttura uguale per ogni paziente.
  Nella figura 1.1 è possibile vedere l'idea della struttura (non completa) proposta dalla Professoressa.

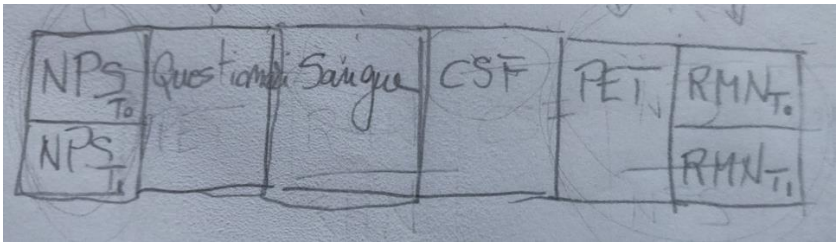


Figura 1.1: Disegno dello schema degli esami

In questo modo, la pagina principale viene visualizzata come un elenco di pazienti (che sono il fulcro del progetto) i cui esami sono direttamente accessibili con un semplice click.
Questa struttura verrà approfondita nella parte di implementazione di questa tesi.

1.2.2. Specifiche aggiuntive

Oltre a tutte le specifiche dei vari campi di ricerca, ne sono state fornite delle ulteriori.
Queste sono:

- Ogni paziente è identificato da un codice univoco composto da una lettera maiuscola ("M" per Modena ed "R" per Reggio Emilia) seguita da tre cifre.
  Il nome di un paziente non verrà mai salvato per motivi legati alla privacy.

- Gerarchia di utenti: è stato richiesto di creare due diverse tipologie di utenti, le quali, sotto nostro consiglio, sono diventate quattro:
    - Amministratore: incaricato di creare ed eventualmente eliminare tutte le tipologie di utenti.
    - Super_Ricercatore: inserisce ed elimina qualsiasi dato all'interno del database.
    - Ricercatore: può inserire i dati di qualsiasi esame, ma può modificare ed eliminare soltanto quelli inseriti da lui stesso.
    - Studente: avrà soltanto la possibilità di visualizzare e scaricare i dati degli esami.

- Ogni Dottore ha un codice univoco. Tutte le volte che immette dati relativi ad un esame, questo codice viene memorizzato in modo da poter risalire a chi ha effettuato certi inserimenti.

- Possibilità di avere diversi ordinamenti dei pazienti all'interno della pagina home.

- Possibilità di scaricare tutti i dati relativi ad un esame di tutti i pazienti, attraverso un file CSV. Per quanto riguarda questo aspetto, è stata richiesta la possibilità di selezionare quali dati di un esame si vogliono scaricare.

1.3. Conclusioni

Dopo aver fatto una panoramica su quelli che sono i vincoli progettuali, sullo scopo del progetto e sulle tecnologie generali che dovranno essere utilizzate, si passa al capitolo dello studio delle tecnologie, per fare luce su certe scelte progettuali.

2. Studio delle tecnologie

Ora che tutti i requisiti sono stati esposti ed analizzati, si passa alla considerazione di tutte le tecnologie fondamentali per la realizzazione del progetto.
In questo capitolo vengono prese in considerazione le tecnologie necessarie per la realizzazione e i principali DBMS. Questi aspetti vengono abbastanza approfonditi per permettere di capire quale tecnologia risulta la più opportuna per raggiungere gli scopi progettuali.
In prima analisi viene considerato il modello classico relazionale, seguito da uno studio dei database che non utilizzano tale modello, per poi passare allo studio di MongoDB (document database) e Flask (micro-framework), i quali sono stati scelti per realizzare il progetto di tesi.

2.1. Tecnologie per la realizzazione

2.1.1. Database (data layer)

Un Database (o base di dati) è un insieme di informazioni (o dati), in genere strutturate, memorizzate all'interno di un sistema informatico.
Il database viene controllato dal cosiddetto Database Management System (DBMS), il quale è progettato per permettere la creazione, la modifica e il recupero dei dati di questo.

Tutti i dati verranno immagazzinati all'interno di un database che, insieme alla web application, è ospitato all'interno di un server interno ad UNIMORE per motivi di privacy dei dati.

2.1.2. Back-end (logic business)

Con il termine back-end, si intende la parte non accessibile all'utente cioè quella che permette il funzionamento dell'interazione tra utente e database. Esso si occupa generalmente dell'elaborazione dei dati passati dal front-end.
Esempi di funzionalità del back-end possono essere:

- Interagire con il database per l'immissione di dati che l'utente ha inserito attraverso il front-end.
- Interagire con il database per recuperare elementi che l'utente ha richiesto attraverso il front-end.
- Interpretare file di tipo diverso.
- Gestione degli utenti (controllo credenziali, creazione sessioni etc.).

2.1.3. Front-end (presentation layer)

Con il termine front-end, si intende la parte accessibile e visibile con la quale interagisce l'utente.
È l'interfaccia attraverso cui navighiamo, visualizziamo informazioni e più in generale, interagiamo con il sito web. È inoltre la parte che acquisisce i dati in input e li elabora in modo da renderli utilizzabili dal back-end.

I linguaggi per quanto riguarda il front-end sono molteplici:

- HTML (HyperText Markup Language): è un linguaggio di markup, utilizzato per dare una struttura a documenti ipertestuali tramite l'utilizzo di tag che ne descrivono le caratteristiche.

- CSS (Cascading Style Sheets): linguaggio utilizzato per definire la formattazione dei documenti HTML. Questo linguaggio funziona grazie a un set di direttive che, applicate a tag HTML, ne definiscono l'apparenza (dal posizionamento al colore).

- JavaScript: linguaggio che permette di creare parti interattive e dinamiche, come ad esempio animazioni o finestre "pop-up".

2.1.4. Web Application

Una Web Application [1] (o webapp) è l'espressione che viene impiegata per indicare tutte le applicazioni distribuite web-based, cioè tutte le applicazioni che non risiedono sulle macchine che le usano, ma che si trovano su server remoti.
Una Web Application è a tutti gli effetti un programma strutturato che funziona su una rete, con la possibilità per un utente di usufruire di un servizio senza però poterne vedere il codice.
Un utente può accedere ad una Web Application attraverso un Web browser (es: Google Chrome, Internet Explorer, Safari, Firefox, etc.).

La struttura di una Web Application è generalmente riassunta dall'architettura multi-tier (su più livelli), la quale nella configurazione più popolare si compone di tre livelli diversi: front-end (presentation layer), back-end (logic business), database (data layer).

2.1.5. Framework

Un Framework (cornice o struttura in inglese) è una struttura logica creata per sviluppare un'applicazione web. Alla base di ogni framework ci sono librerie di codice utilizzate con uno o più linguaggi di programmazione. Alle volte, un framework comprende vari strumenti per lo sviluppo del software come un Integrated Development Enviroment (IDE), un debugger ed altri strumenti per facilitare e velocizzare il raggiungimento dell'obiettivo.

La funzione principale di un framework è quella di creare una struttura di contorno, consentendo al programmatore di creare soltanto il contenuto vero e proprio dell'applicazione, in modo da velocizzare il processo di produzione.
In questo modo il programmatore non deve tutte le volte "reinventare la ruota" ma può utilizzare codice già ben testato.
Un esempio pratico potrebbe essere quello di voler aggiungere alla propria Web Application una parte di e-commerce. Alcuni framework facilitano il lavoro gestendo automaticamente le transazioni e comunicazioni con gli enti interessati.

2.2. Relational database management system (RDBMS)

I database relazionali [2] sono un tipo di database che memorizzano e forniscono l'accesso a dati correlati tra loro. Sono basati sul modello logico di rappresentazione dei dati in tabelle, per questo definiti relazionali. In un RDBMS, ogni riga (o tupla) della tabella è un record con un ID univoco chiamato chiave. Le colonne delle tabelle contengono gli attributi dei dati e in genere ogni record ha un valore per ogni attributo, il che semplifica la definizione delle relazioni tra questi.

Chi progetta utilizzando un modello relazionale ha a disposizione una rappresentazione consistente e logica dell'informazione. La consistenza si ottiene grazie a vincoli appropriati i quali vanno a formare lo schema logico.
Un database di tipo relazionale è ottimo per memorizzare dati strutturati.

Tipologia di dato:

1. I dati strutturati sono organizzati secondo schemi e tabelle rigide, per questo sono la tipologia più adatta per quanto riguarda il modello relazionale.
   Esempi: numeri, stringhe, etc.

2. I dati semi-strutturati [3] sono dati strutturati con una forma non fedele alla struttura formale dei modelli di dato propri dei database relazionali, ma che comunque contengono etichette o marcatori per separare gli elementi semantici e rafforzare le gerarchie di record e campi all'interno del dato. Sono anche conosciuti come dati senza schema o dati con struttura autodescritta.
   Esempi: .xml, .csv, .json, etc.

3. I dati non strutturati sono una forma di dato senza schema, quindi non adatti ad essere salvati in un database relazionale, ragione per cui vengono utilizzati altri tipi di database per la memorizzazione di questi.
   Esempi: file audio, immagini, file contenenti testi, etc.

I database relazionali implementano un processo di normalizzazione, il quale ha lo scopo di

eliminare la ridondanza dalle relazioni. Una volta effettuato il processo di normalizzazione si otterrà un database “in forma normale”, proprietà di una base di dati relazionale, che ne misura la “qualità”.

Recupero di dati

I dati vengono recuperati attraverso query (o interrogazioni) e sono filtrati secondo criteri stabiliti da chi le realizza.

Uno dei punti di forza dei database relazionali è la clausola JOIN, la quale combina le righe di due o più tabelle logicamente correlate.

Esempio di query con JOIN:

Tabella Orders

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

Figura 2.1: Esempio di una tabella di un database relazionale

Tabella Customers

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

Figura 2.2: Esempio di una tabella di un database relazionale

Query

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

Figura 2.3: Esempio di una query con attributo JOIN

Risultato query:

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

Figura 2.4: Risultato della query in esempio

Vantaggi di un database relazionale:

- Tipo di database più utilizzato come si può vedere in figura 2.5 e con una base teorica solida. Per questi motivi può vantare la community di programmatori più numerosa in questo ambito.

Classifica dei 10 DBMS più utilizzati

354 systems in ranking, March 2020

Rank			DBMS	Database Model	Score		
Mar 2020	Feb 2020	Mar 2019			Mar 2020	Feb 2020	Mar 2019
1.	1.	1.	Oracle	Relational, Multi-model	1340.64	-4.11	+61.50
2.	2.	2.	MySQL	Relational, Multi-model	1259.73	-7.92	+61.48
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1097.86	+4.11	+50.01
4.	4.	4.	PostgreSQL	Relational, Multi-model	513.92	+6.98	+44.11
5.	5.	5.	MongoDB	Document, Multi-model	437.61	+4.28	+36.27
6.	6.	6.	IBM Db2	Relational, Multi-model	162.56	-2.99	-14.64
7.	7.	↑ 9.	Elasticsearch	Search engine, Multi-model	149.17	-2.98	+6.38
8.	8.	8.	Redis	Key-value, Multi-model	147.58	-3.84	+1.46
9.	9.	↓ 7.	Microsoft Access	Relational	125.14	-2.92	-21.07
10.	10.	10.	SQLite	Relational	121.95	-1.41	-2.92

Figura 2.5: Classifica DBMS marzo 2020

- Progettazione e realizzazione semplificate dalla struttura fondamentale: la tabella. Le informazioni sono organizzate in righe e colonne, la cui organizzazione è facilmente associabile ad un semplice foglio di calcolo.
- Linguaggio di accesso ai dati standard: Structured Query Language (SQL).
- Presenza della clausola di join delle tabelle. Questo genere di meccanismo non è utilizzabile o è sconsigliato in molti modelli di database NoSQL poiché quando si lavora con dati estremamente grandi (motivo principale per l'impiego di questi database) l'overhead di una singola ricerca è a volte troppo grande.
- Evita la duplicazione dei dati. La struttura di un RDBMS permette facilmente di eliminare la ridondanza tramite la creazione di più tabelle, le quali saranno in relazione tra di loro.

Svantaggi di un database relazionale:

- Struttura rigida dei dati: alcuni tipi di dati non possono essere memorizzati in uno schema di tabelle a due dimensioni. I tipi di dato non strutturati (presenti sempre più spesso in ambito web) non possono essere salvati all'interno del modello di database relazionale.
- Assenza di schema gerarchico: i database relazionali, contrariamente ai database ad oggetti, non forniscono la possibilità di creare schemi con classi strutturate in modo gerarchico. Non è possibile di conseguenza avere entità che ereditano proprietà. Tutte le righe (o tuple) di un database relazionale sono sempre allo stesso livello di gerarchia.
- Meno Performati rispetto ai database NoSQL per grandi moli di dati.
- Segmentazione dei dati: i sistemi di database relazionali suddividono le informazioni all'interno di tabelle separate, portando ad una segmentazione dei dati, dati cioè collegati logicamente ma non sempre memorizzati all'interno della stessa tabella. Questo impatta negativamente sulle prestazioni poiché vi è la necessità di interrogare diverse tabelle per ottenere il risultato desiderato, il che crea un problema di scalabilità.
- Limite della lunghezza dei dati: l'immissione di più informazioni di quante se ne possano memorizzare all'interno di un campo, potrebbe portare alla perdita delle stesse.
- Difficoltà nel ricreare aspetti complessi del mondo reale attraverso strutture tabellari.
- Per creare query specifiche vi è la necessità di conoscere bene l'intera struttura del database.
- Scalabilità: è un problema sia per database SQL che per database NoSQL. I primi sono scalabili "verticalmente": ciò significa che è possibile aumentare la potenza del server (tramite l'aggiunta o espansione di moduli: RAM, CPU, SSD, etc.) in modo da poter aumentare il carico su di esso. I database NoSQL sono invece scalabili "orizzontalmente": ciò significa che tramite varie tecniche, quali sharding o l'aggiunta di server è possibile distribuire il carico su più macchine.

2.3. Not Only SQL (NoSQL) databases

NoSQL[4] è un movimento ormai consolidato da anni che promuove sistemi software la cui persistenza dei dati è in generale caratterizzata dal fatto di non utilizzare il modello relazionale (di solito usato dalle basi di dati tradizionali (RDBMS)). L'espressione "NoSQL" fa riferimento al linguaggio SQL, che è il più comune linguaggio di interrogazione dei dati nei database relazionali, in questo caso preso a simbolo dell'intero paradigma relazionale.
La ragione principale dello sviluppo dei database NoSQL è riconducibile ai bisogni delle grandi aziende, quali Amazon e Google, di gestire grandi moli di dati sia in ingresso che in uscita. L'obiettivo era quindi una tecnologia che riuscisse in tempi brevi a soddisfare le elevate richieste in ingresso e a gestire l'aumento costante della mole di dati da archiviare.

I database NoSQL nella maggior parte dei casi non hanno uno schema fisso, vengono detti per questo "Schemaless" (senza schema), sono facili da scalare "orizzontalmente" mentre le operazioni di JOIN in ambito Big Data sono sconsigliate, in quanto degradano le prestazioni del sistema di recupero dei dati. Non avendo uno schema predefinito, sono perfetti per ambienti di sviluppo in rapida evoluzione o cambiamento consentendo agli sviluppatori di apportare modifiche velocemente senza influire sulle applicazioni sottostanti.

Vantaggi dei database NoSQL:

- Leggerezza computazionale: i database NoSQL non prevedono operazioni di aggregazione sui dati (es. operazione di JOIN), poiché tutte le informazioni sono presenti all'interno di un unico documento, che è associato all'oggetto su cui effettuare operazioni. Negli ambienti SQL la complessità delle operazioni di JOIN cresce con l'aumentare delle dimensioni del database.
  Per questo i NoSQL database sono i più adatti in campo Big Data.

- Assenza di schema (schemaless): i database NoSQL sono privi di schema in quanto il documento JSON contiene tutti i campi necessari, senza necessità di definizione. In questo modo, possiamo arricchire le nostre applicazioni di nuovi dati e informazioni, definibili liberamente all'interno dei documenti JSON senza rischi per l'integrità dei dati. I database non relazionali, a differenza di quelli SQL, si rivelano quindi adatti ad inglobare velocemente nuovi tipi di dati e a conservare dati semi-strutturati o non strutturati.

- Scalabilità: i database NoSQL sono “orizzontalmente” scalabili senza grandi problemi. Grazie alla possibilità di aggiungere server per distribuire il carico di lavoro, anche con grandi moli di dati o di richieste in entrata, i tempi di risposta non cambiano. Tutto questo è reso possibile poiché l’aggregazione dei dati è “Schemaless”, rendendo facilmente duplicabile la collezione. La scalabilità facilitata rende di fatto la vulnerabilità “Single Point of Failure” (SPOF) più semplice da risolvibile.

Svantaggi dei database NoSQL:

- Duplicazione delle informazioni (database “denormalizzato”): i dati necessari durante la lettura da database devono essere presenti all’interno dello stesso documento (o entità). Questo fa sì che ci siano ridondanze all’interno di tutti i documenti (o entità) che richiedono le stesse informazioni, rendendo impossibile il controllo di integrità dei dati da parte del DBMS.
  Esempio: si vogliono visualizzare codice esame, e dati anagrafici di un paziente riguardo una certa visita specialistica. Le informazioni del paziente devono essere salvate anche nel documento della visita in modo da non dover fare un’operazione di JOIN tra i documenti che contengono le visite e documenti che contengono i pazienti**.**

- Operazioni di aggiornamento più complicate: quando si deve aggiornare un dato bisogna aggiornarlo in tutti i documenti (o entità) in cui è presente

- Esportazione dati difficile: non sono disponibili molto spesso API per poter esportare tutti i dati. Questo può essere un grande svantaggio nel caso in cui si volesse passare ad un altro modello o ad un’altra soluzione.

- Operazioni di Data Fixing più complicate: non avendo un linguaggio standard a supporto come l’SQL, le operazioni di correzione dei dati devono essere attuate attraverso logiche progettate ad hoc.

- Linguaggio di interrogazione: non essendoci uno standard, contrariamente ai RDBMS, ogni DBMS NoSQL ha un linguaggio di interrogazione diverso, questo fa sì che logiche create per recuperare dati tramite SQL debbano essere riprogrammate.

- Tecnologia relativamente giovane: essendo stati introdotti successivamente al consolidamento dei database relazionali, i quali esistono da più di 40 anni, sono considerati più “volatili”, in alcuni casi meno documentati e con community più ristrette.

I principali modelli di database sono: DataGraph Database, Network Database, Object Database, e Document Databases.

2.3.1. DataGraph Database

In informatica una base di dati a grafo [5], o database a grafo, è una tipologia di database che utilizza nodi e archi per rappresentare e archiviare l'informazione. La rappresentazione dei dati mediante grafi offre un'alternativa al modello relazionale, ai database orientati al documento e più in generale ai sistemi ad archivi strutturati (structured storage).
Le relazioni vengono trattate come oggetti separati per poter avere performance migliori.
Questo sistema vanta due modelli diversi per archiviare dati: il primo utilizza liste “double linked”, (concatenate) mostrate nella figura 2.6, tramite questo metodo il tempo che viene impiegato per attraversare il grafo non cambia in base alle sue dimensioni. Di contro va detto che le prestazioni per quanto riguarda le operazioni di scrittura risentono molto della struttura. Il secondo modello utilizza una lista di adiacenza, mostrata nella figura 2.7, rendendo più veloci di fatto le operazioni di scrittura, se confrontate con il primo metodo (double linked list).

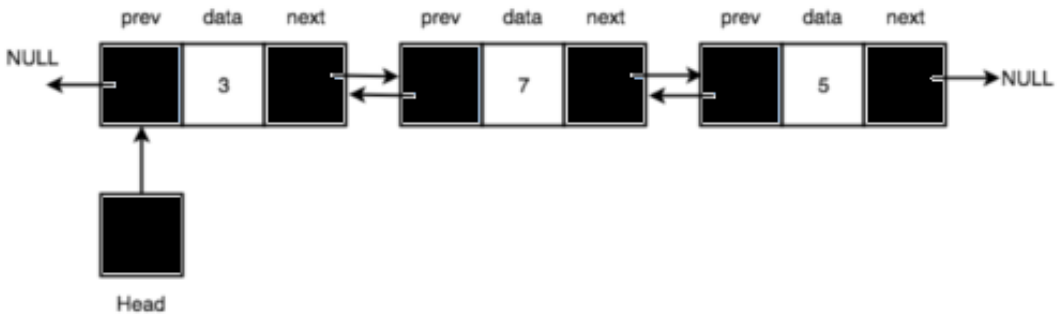


Figura 2.6: Doubly linked list

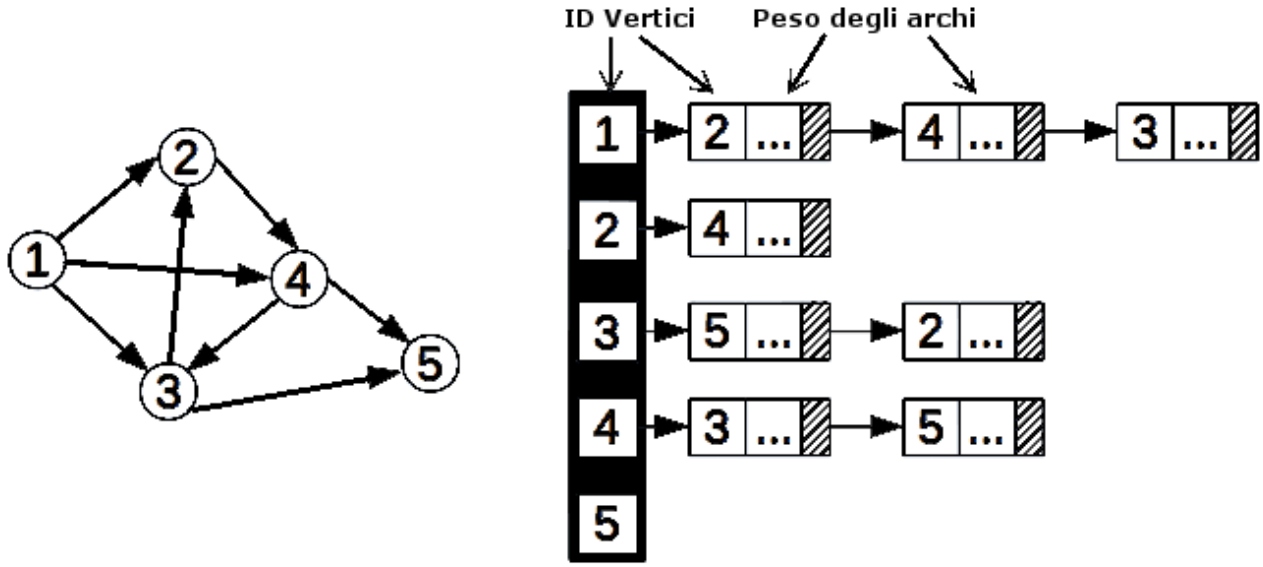


Figura 2.7: Adjacency list

I due linguaggi di interrogazione più utilizzati sono Cypher (il quale può essere definito come linguaggio dichiarativo) e Gremlin (simile ad un linguaggio di programmazione).

Vantaggi DataGraph Database:

- Velocità: grazie alla mappatura utilizzata, i database a grafo sono spesso più veloci di quelli relazionali nel recupero delle informazioni.
  Con l'incremento del volume dei dati, le relazioni tra questi crescono ancora più velocemente. In questo scenario una base di dati a grafo è la soluzione migliore.

- Schema dei dati: Sono adatti a gestire dati e schemi che cambiano nel tempo. Vengono preferiti in ambito aziendale per questa flessibilità, poiché il team di sviluppo può facilmente seguire la costruzione step by step senza creare una struttura standard a priori.

Svantaggi DataGraph Database:

- Supporto: essendo una soluzione tecnologica abbastanza nuova in campo database, è difficile ricevere supporto in caso di problemi.

- Version Control: I fornitori di graph database non forniscono supporto nativo per il versioning basato sul tempo, per questo non sono adatti per tenere traccia delle modifiche.

- Linguaggio di interrogazione non standardizzato: una delle cause principali per la quale questo tipo di database viene lasciato da parte dalle scelte progettuali.

- Linguaggio dichiarativo assente: Molti non offrono un linguaggio di tipo dichiarativo e talvolta piattaforme diverse adottano linguaggi diversi creando di fatto una difficoltà nel migrare codice da una piattaforma ad un'altra.

- Complessità delle operazioni: l'inserimento, l'aggiornamento e l'eliminazione di record richiedono numerose operazioni di aggiornamento dei puntatori rallentando di fatto l'intero sistema. Questo fa capire immediatamente che operazioni di modifica della struttura del database sono molto difficili e onerose dal punto di vista logico.

2.3.2. Network Database

In informatica il modello reticolare (o Network Database) [6] è una tipologia di implementazione di un database ideato nel 1973 e perfezionato nel 1978 . Si basa sulla logica di record e puntatori e ne esiste una versione standard, detta CODASYL. Questo modello a differenza di quello relazionale, che si basa su gruppi di record, considera un record alla volta. Esso è molto vicino alla struttura fisica di memorizzazione dei record, infatti per poter "navigare" all'interno della base di dati si devono utilizzare puntatori, proprio come avviene a basso livello nella memoria fisica di un computer. Le relazioni possono essere immaginate come una rete (o network) poiché ogni elemento può puntare più elementi e viceversa.

Vantaggi Network Database:

- Relazioni "più larghe": a differenza del modello relazionale, dove un record figlio può avere soltanto un record padre, il modello reticolare permette di avere più record padri e più record figli. In questo modo possono essere modellate più facilmente relazioni del mondo reale

- Comprensione: il modello reticolare è concettualmente semplice e ha una curva di

apprendimento simile al modello gerarchico e relazionale.

Svantaggi Network Database:

- Poca Flessibilità: la maggiore flessibilità del modello relazionale con l'avvento di nuove tecnologie più veloci ha fatto sì che il modello reticolare fosse sempre meno utilizzato.

- Complessità di navigazione: tutti i dati vengono gestiti tramite puntatori rendendo di fatto molto più complicata la struttura del database stesso.

2.3.3. Hierchical Database

Il modello gerarchico [7] è stato il primo modello di database ad affermarsi sul mercato.
Questo modello prevede che i dati siano organizzati secondo strutture ad albero, che si suppone rifletta una gerarchia esistente tra le entità che appartengono ad esso.
Ogni albero ha un'unica radice, i nodi figli rappresentano le entità e gli archi le associazioni.
Ogni padre può avere diversi figli ma un figlio può avere un solo padre consentendo di fatto soltanto relazioni di tipo "uno a molti".
La struttura del modello gerarchico è riconducibile a quella dei file system.
Questo tipo di database è stato largamente utilizzato negli anni 60' e 70' all'interno dei primi mainframe, sostituiti poi da DBMS reticolare (network database) e successivamente dai database relazionali.

Vantaggi Hierchical Database:

- facilità di interrogazione: le interrogazioni vengono rese più semplici quando la realtà da rappresentare è gerarchica.

Svantaggi Hierchical Database:

- Relazione "molti a molti": è impossibile ottenere questo tipo di relazione senza duplicare i dati.

- Modello poco utilizzato: al giorno d'oggi è uno dei modelli meno utilizzati poiché

considerato obsoleto e inferiore in quasi tutti gli aspetti rispetto ai modelli odierni.

- Struttura rigida: questo modello non è adatto per rappresentare realtà che non sono di tipo gerarchico. A causa di questo ordine, i livelli non adiacenti non hanno la possibilità di interagire. La struttura gerarchica del database è quindi tanto intuibile quanto rigida.

- Velocità delle interrogazioni: è necessario attraversare tutto l'albero partendo dalla radice fino al nodo interessato per recuperare i dati.

2.3.4. Object-Oriented Database

Un modello di base di dati a oggetti [8] o database ad oggetti (ODBMS, cioè: Object Database Management System) è un modello in cui l'informazione è rappresentata sotto forma di oggetti come nei linguaggi di programmazione ad oggetti.
Come nei linguaggi sopra citati questo tipo di database seguon i principi fondamentali della programmazione ad oggetti, dove un oggetto è un'entità del modo reale e una classe è una collezione di oggetti. L'approccio object-oriented considera tutte le entità come oggetti: questi conterranno proprietà e metodi che operano sui dati dell'oggetto stesso. Ogni oggetto è identificato tramite una chiave univoca.
I database ad oggetti vantano tutte le proprietà dei linguaggi di programmazione omonimi cioè: incapsulamento, ereditarietà e polimorfismo.
L'idea di questo tipo di database risale al 1985 e questi vengono utilizzati tutt'ora quando sono richiesti calcoli e risultati veloci, per questo vengono adottati per sistemi real-time, modellazioni 3d etc.

Vantaggi Object-Oriented Database:

- Salvataggio degli oggetti: i database ad oggetti permettono di salvare in modo permanente un oggetto per poi poterlo recuperare ed utilizzare in un secondo momento. Nell'ipotesi in cui si lavori con oggetti e un RDBMS bisognerebbe leggere i dati memorizzati all'interno delle tuple, caricarli all'interno dell'oggetto e solo dopo questo passo i dati sono pronti ad essere utilizzati.

- Gestire dati complessi: i database ad oggetti sono ottimi per gestire rappresentazioni precise della realtà poiché quasi tutto è raffigurabile tramite un oggetto.

- Multi-linguaggio: alcuni database ad oggetti possono essere utilizzati con più linguaggi di programmazione (es: GemStone/s supporta c++ e Java).

- Sistema di caching: i database ad oggetti mettono a disposizione un sistema di caching il quale crea una replica parziale del database. Grazie a questo metodo, gli accessi al disco diminuiscono in modo importante, consentendo all'applicazione di accedere ai dati direttamente dalla memoria del programma.

- Recupero informazioni: una caratteristica importante di questi database è la possibilità di recuperare gli oggetti tramite backup nel caso in cui si verifichi un problema con il sistema o con l'applicazione.

Svantaggi Object-Oriented Database:

- Community piccola: il numero di programmatori che sviluppa sopra queste piattaforme è molto minore rispetto al numero di programmatori su sistemi relazionali. Per questo motivo vi è una maggiore difficoltà nel reperire informazioni qualora si incappasse in problemi o errori.

- Standard di interrogazione: come per molti altri database NoSQL, non vi è uno standard per quanto riguarda il linguaggio di interrogazione.

- Scelta limitata del linguaggio: non tutti i linguaggi di programmazione supportano i database ad oggetti.

- Steep learning curve: i database ad oggetti risultano più difficili da imparare rispetto ad altri, soprattutto per chi non ha mai utilizzato un linguaggio object-oriented.

2.3.5. Document Database

I document database sono database di tipo non relazionale [9] progettati per memorizzare e cercare dati come documenti di tipo JSON. Questi semplificano agli sviluppatori la ricerca e la memorizzazione di dati all'interno del database, utilizzando lo stesso modello di documento che

viene adoperato nel codice dell'applicazione.
Questo sistema salva i dati in strutture dette documenti, i quali possono contenere qualsiasi tipo di dato (dal paziente di un ospedale ai dati per l'apprendimento di un sistema autonomo). I documenti a loro volta sono organizzati in collezioni (o collection), le quali potranno essere collegate tra di loro attraverso riferimenti.
Per fare una comparazione con i database relazionali, una collezione è analoga ad una tabella e consiste in una raccolta di documenti i quali sono analoghi ai record.

La natura gerarchica, semi-strutturata e flessibile dei documenti permette di ampliare e perfezionare la struttura in base alle esigenze.
I document database forniscono la possibilità di creare un'indicizzazione totalmente flessibile in modo da garantire velocità nel recupero dei dati.
Un indice è una struttura speciale che memorizza una parte del set di dati della raccolta in un formato facile da "attraversare".
Questo tipo di database è ritenuto uno dei migliori nel caso si debbano memorizzare dati semi-strutturati o non strutturati e nel caso in cui ci sia il bisogno di ampliare l'applicazione, Infatti, non ci sono problemi nell'aggiungere collezioni di documenti e nell'aggiungere nuovi campi per i dati. Nel caso in cui le collezioni debbano essere modificate, sarà necessario aggiornare soltanto i documenti interessati.

I Database orientati ai documenti memorizzano e codificano i dati in formati comuni che possono essere: JSON, XML, YAML, e formati binari come: BSON, PDF, MS Word, Excel.
Quale formato viene utilizzato dipende soltanto dalla piattaforma scelta (es: MongoDB utilizza il formato BSON).

Vantaggi:

- Modello NoSQL più utilizzato: i document database sono i database largamente più utilizzati per quanto riguarda il panorama dei NoSQL database. Per questa ragione, sono quelli che vantano la community più numerosa e il numero maggiore di informazioni reperibili online.

- Flessibilità: non essendoci uno schema fisso, è facile apportare cambiamenti in corso d'opera senza dover ripensare l'intera struttura del database. Si tratta della soluzione

perfetta quando la struttura dei documenti non è rigida (es: e-mail e articoli non hanno sempre la stessa struttura).

- Apprendimento facile: rispetto ad un database relazionale, ha un apprendimento facilitato in quanto non esistono schemi o procedure da seguire, come ad esempio la normalizzazione.

- Velocità a portata di mano: nel caso si volessero velocizzare alcuni tipi di ricerche è sufficiente indicizzare i campi interessati.

- Gestire un numero elevato di attributi: i database relazionali non sono per nulla adatti quando vi è il bisogno di gestire migliaia di attributi. Al contrario, i document database sono ottimi sotto questo aspetto dato che gli attributi possono essere salvati all'interno di un unico documento. In questo modo viene velocizzata la lettura e facilitata la gestione dell'intero sistema poichè modificare gli attributi all'interno di un documento non avrà conseguenze sugli altri.

- Scrittura e recupero di informazioni facilitato: Rispetto ai classici database relazionali, l'immissione e soprattutto il recupero dei dati è reso molto più semplice grazie alla struttura del database stesso.

- Motore di ricerca: Molti Document Database forniscono motori di ricerca permettendo di effettuare information retrieval (IR) [10] (in italiano recupero delle informazioni) ossia l'insieme delle tecniche utilizzate per gestire la rappresentazione, la memorizzazione, l'organizzazione e l'accesso ad oggetti contenenti informazioni quali documenti, pagine web, cataloghi online e oggetti multimediali.

Svantaggi:

- Utilizzo di Map Reduce: quando si lavora con grandi moli di dati è necessario utilizzare Map Reduce ([11] framework software brevettato e introdotto da Google per supportare la computazione distribuita su grandi quantità di dati in cluster di computer. Il framework è ispirato alle funzioni map e reduce usate nella programmazione funzionale)

- Limiti di dimensione indici: gli indici sono strumenti ottimi e fondamentali quando si parla di document database. Bisogna prestare per questo motivo particolare attenzione poiché la loro dimensione ha un limite. Ogni piattaforma ha una gestione differente per questo problema. (es: MongoDB non crea l'indice se questo eccede nelle dimensioni)

Di DBMS orientati ai documenti ne esistono diversi e tra i più utilizzati troviamo: MongoDB, Amazon dynamoBD, Arango DB, Azure Cosmos DB, Amazon Document DB.

Al Primo posto troviamo MongoDB, il quale sorpassa per molti aspetti tutti gli altri.
Si tratta del DBMS più utilizzato nel campo dei Document Database, non dipende da alcuna piattaforma (al contrario di Amazon DynamoDB, Amazon Document DB e Azure Cosmos DB, dove i primi due sono proprietari di Amazon e il terzo di Microsoft).

Arango DB è una valida alternativa a Mongodb essendo anch'esso open source.

Le principali differenze tra le due tecnologie sono riportate di seguito:

- MongoDB è il database basato su documenti più popolare e può vantare una community molto più grande di quella di ArangoDB.
- ArangoDB è adatto quando sono necessari più modelli per la memorizzazione di dati. Mentre MongoDB è un document database, ArangoDB è in grado di affiancare anche un modello a grafo e/o un modello chiave-valore.
- MongoDB supporta più del doppio dei linguaggi di programmazione.
- MongoDB è ritenuto più stabile in quanto rilasciato tre anni prima, ed anche per questo viene preferito in ambito business.

2.4. MongoDB

MongoDB (da "humongous", enorme) è un DBMS open source orientato ai documenti, che supporta diversi tipi di dato. È nato nel 2009 per l'uso di applicazioni in ambito big data e per scenari dove la rigidità del modello relazionale non è adatta. Nell'ambito dei database NoSql è il più conosciuto ed utilizzato, sia in ambito privato che aziendale. Per questo vanta la community di programmatori più numerosa per quanto riguarda i database non relazionali.
Come tutti i database NoSQL, MongoDB è definito "senza schema" (o schemaless), questo fa sì

che non debbano essere dichiarati a priori il numero o il tipo degli elementi da inserire.
I dati appartenenti ad una entità (o documento) saranno memorizzati come coppia chiave-valore dando in questo modo la possibilità di avere dati molto diversi all'interno della stessa entità.

2.4.1. Organizzazione dei dati all'interno di MongoDB

In MongoDB, i dati sono archiviati all'interno di "collezioni" (o collection).
Le collezioni, come mostrato nella figura 2.8, sono a tutti gli effetti delle raccolte di documenti e prendono il posto della struttura tabellare dei database relazionali.
I documenti all'interno di una collezione possono avere campi diversi e sono correlati tra loro tramite un collegamento logico. Una collezione appartiene a un solo database.

Figura 2.8: Esempio di collection

Un documento è un file BSON (binary JSON) che contiene informazioni legate tra di loro da una relazione logica. Essendo che i tipi BSON sono un superset dei tipi JSON, i dati che vi possono essere memorizzati sono molteplici: tipi standard di JSON (integer, dougle, array, boolean, object e null), date, object id, code, regular expression, binary data, file e geographic coordinates.
BSON, più nello specifico, è un formato binario creato per occupare il minor spazio possibile e avere una maggiore velocità di ricerca. In questi documenti vi è un campo lunghezza prefissato in modo da rendere più veloce il recupero delle informazioni (a volte questo porta ad utilizzare più spazio rispetto a un documento JSON).
I documenti sono composti da coppie di campi e valori, come mostrato nella figura 2.9.

```
{
    _id: ObjectId("556d54df87gb86df56b45")
    name : "Luca",
    surname : "Sala",
    email : "esempio@gmail.com",
    birthdate : "05/05/1997"
}
```

Figura 2.9: Esempio di documento

Per quanto riguarda l'ordine dei campi di un documento, il campo _id, il quale funge da chiave primaria sarà sempre il primo. Nel caso questo campo venisse omesso durante un inserimento, MongoDB lo genererà automaticamente.
Di default, MongoDB crea un indice utilizzando il campo "_id" di una collezione
Il limite di dimensione dei documenti BSON è di 16 megabytes in modo da assicurare che un documento non sfrutti in modo eccessivo spazio all'interno della RAM o la larghezza di banda durante il trasferimento.
Nel caso si presentasse la necessità di memorizzare documenti più grandi MongoDB fornisce l'API GridFS, la quale memorizza i documenti all'interno di oggetti.

2.3.2. Panoramica di MongoDB

MongoDB mette a disposizione Mongo Shell, un'interfaccia interattiva JavaScript utilizzata per effettuare query, update dei dati e per lanciare operazioni di amministrazione riguardanti il sistema.
Aggiungendo la stringa "<mongodb installation dir> /bin" alla variabile d'ambiente, sarà possibile lanciare il servizio digitando soltanto "mongo" all'interno della shell.
La porta di default utilizzata è la 27017 per la connessione locale. Per quanto riguarda la connessione ad una macchina remota verrà utilizzato il comando:
#mongo "mongodb://mongodb0.example.com:numeroporta".
Altre opzioni sono disponibili all'interno della documentazione ufficiale.

2.4.4. Creazione del database

Tramite la Mongo Shell è possibile selezionare il database da utilizzare tramite il comando:
use <database> (come mostrato in figura 2.10).
Nel caso il database scelto non esista, appena verrà effettuato il primo inserimento (per esempio di una collection), esso verrà creato.

```
use newDatabase
db.myCollection.insertOne( { name: Luca } );
```

Figura 2.10: Esempio creazione database

2.3.5. Operazioni create, read, update, delete (CRUD operations)

Create

- db.collection.insertOne(): inserisce un solo documento e ritorna un documento contenente l'id del nuovo documento inserito.

```
db.inventory.insertOne(
   { item: "pen", qty: 100, price: 1, type: { thiskness: 0.5, hardness: "2B"} }
)
```

Figura 2.11: Esempio insertOne()

- db.collection.insertMany(): permette di inserire molteplici documenti in una collection. Questo metodo necessita di un array di documenti in ingresso.

```
db.inventory.insertMany([
   { item: "pen_pencil", qty: 100, price: 1, type: { thiskness: 0.5, hardness: "2B"} },
   { item: "pencil", qty: 50, price: 0.90, type: { thiskness: 0.3, hardness: "HB"} },
   { item: "pen_pencil", qty: 200, price: 0.95, type: { thiskness: 0.3, hardness: "H"} }
])
```

Figura 2.12: Esempio insertMany()

Read

- db.collection.find({}): passando un documento vuoto ("{}"),questa operazione permette di recuperare tutti i documenti di una collezione. È l'equivalente di "SELECT * FROM table" per i database SQL.

- db. collection.find({ name: "Luca", surname: "Sala" }): in questo modo è possibile specificare delle condizioni di uguaglianza.

- db.collection.find({ item: { $in: [" pencil ", "pen_pencil"] } }): l'operatore "$in" permette di recuperare tutti i documenti che (in questo caso) hanno come item "pencil" o "pen_pencil".

- db.collection.find({ item: "pencil", qty: { $lt: 100 } }): l'operatore "$lt" permette di

recuperare tutti i documenti che hanno quantità (in questo caso) minore di 100.

- Le condizioni che possono essere utilizzati sono: $eq, $gt, $gte, $in, $lt, $lte, $ne $nin

- db.collection.find({ $or: [{ item: "pencil" }, { qty: { $lt: 30 } }] }): è possibile inoltre utilizzare in questo modo gli operatori $or, $and, $not e $nor.

Update

MongoDB fornisce le seguenti operazioni per gli aggiornamenti:

- db.collection.updateOne(): aggiorna un singolo documento
- db.collection.updateMany(): aggiorna più documenti contemporaneamente
- db.collection.replaceOne(): rimpiazza un singolo documento all'interno di una collezione

Delete

Le operazioni di eliminazione permettono di rimuovere uno o più documenti da una collezione. Le Operazioni che MongoDB mette a disposizione per questa procedura sono:

- db.collection.deleteOne(): permette di eliminare un documento che soddisfa un certo filtro

```
db.collection.deleteOne(
    { _id: "8676g86sd96gs56d" }
)
```

Figura 2.13: Esempio deleteOne()

Nell'esempio 2.x verrà eliminato il documento avente "_id" specificato nel filtro.

- db.collection.deleteMany(): permette di eliminare più documenti che soddisfano un certo filtro

```
db.collection.deleteMany(
    { name: "Luca" }
)
```

Figura 2.14: Esempio deleteMany()

Nell'esempio 2.x verranno eliminati dalla collection tuti i documenti aventi "name" uguale a "Luca".

Tutte le operazioni CRUD sono rese disponibili tramite: Mongo Shell, Compass, Python, Java, Node.JS, PHP, Motor, Java(ASYNC) C#, Perl, Ruby, Scala e Go.

Le operazioni sono sempre le stesse, ma la sintassi cambia leggermente da linguaggio a linguaggio.

Per comprendere a pieno i termini utilizzati da MongoDB rispetto a quelli SQL viene riportata nella figura 2.15 la legenda ufficiale presente sul sito di MongoDB [12].

SQL Terms/Concepts	MongoDB Terms/Concepts
database	database
table	collection
row	document or BSON document
column	field
index	index
table joins	`$lookup`, embedded documents
primary key Specify any unique column or column combination as primary key.	primary key In MongoDB, the primary key is automatically set to the _id field.
aggregation (e.g. group by)	aggregation pipeline See the SQL to Aggregation Mapping Chart.
SELECT INTO NEW_TABLE	`$out` See the SQL to Aggregation Mapping Chart.
MERGE INTO TABLE	`$merge` (Available starting in MongoDB 4.2) See the SQL to Aggregation Mapping Chart.
transactions	transactions

Figura 2.15: comparazione termini MongoDB e SQL

2.5. Flask

Flask è un Web Server Gateway Interface (WSGI) leggero riferito anche come “micro-framework” scritto in Python, il quale è stato progettato per rendere le operazioni di setting iniziali facili e veloci. Ha la capacità di adattarsi molto bene ad applicazioni complesse e ad oggi è uno dei Web framework in Python più utilizzati.
Il termine “micro” significa che lo scopo di Flask è quello di mantenere un core tanto semplice quanto estendibile. La maggior parte delle scelte è effettuata dall’utente (come per esempio la scelta del database).

Insieme MongoDB, Flask è il web framework più utilizzato e permette di realizzare in modo intuitivo e facile una Web Application in Python attraverso i diversi strumenti e librerie messe a disposizione. Include un web server il quale può essere utilizzato a scopo di test e sviluppo.

Flask dipende da Jinja, un motore di template veloce e largamente utilizzato (scritto in Python e per il linguaggio Python) e dal toolkit WSGI Werkzeug (dal tedesco: werk (“work”), zeug (“stuff”)).
Werkzeug è una libreria WSGI di Web Application completa. Partita come una piccola collezione di strumenti per applicazioni, è diventanda oggi una delle migliori librerie nel suo ambito.
Flask utilizza Werkzeug per gestire il WSGI e al tempo stesso fornisce strutture e schemi per la realizzazione di applicazioni più o meno complesse.

Parte II

Progetto e Sviluppo

3. Progettazione

3.1. Introduzione

Questo capitolo ha come obiettivo quello di illustrare tutte le parti che vanno a comporre il progetto finale. Ora che le tecnologie sono state analizzate, si procede a sviscerare quelli che sono i requisiti funzionali dell'applicazione: in questo modo sarà possibile seguire passo per passo lo sviluppo di essa.

Una volta delineati e approfonditi i requisiti funzionali, si passerà alla progettazione dell'interfaccia utente e successivamente ai casi d'uso.

Infine, verranno valutati gli aspetti riguardanti MongoDB e Flask, in modo da spiegare anche i motivi che hanno portato all'utilizzo di queste tecnologie.

3.2. Requisiti funzionali

I requisiti sono una delle componenti più importanti per quanto riguarda la realizzazione di qualsiasi progetto. Per questo motivo, prima di considerare qualsiasi tipo di tecnologia e soluzione è bene avere presenti in modo chiaro i requisiti principali.
Tutte le indicazioni principali sono state fornite dai Professori e Dottori che utilizzeranno il sistema sviluppato.
È stato di fondamentale importanza poter avere più incontri con i diretti interessati durante l'arco della realizzazione del progetto, in modo da avere feedback continui e in modo da poter costruire insieme quelle che sono poi diventate le linee guida.
In questo capitolo si farà riferimento alle persone che hanno contribuito a delineare i requisiti come

“gruppo di ricerca”.

3.2.1. Funzionalità di base

Le funzionalità di base sono tutte quelle funzioni presenti nella maggior parte delle Web Application (quali login e registrazione). Di seguito sono elencate insieme ad una spiegazione di come sono state progettate.

- **Registrazione**: In questo progetto, la registrazione è stata gestita diversamente rispetto ad una qualsiasi Web App. Solitamente è presente (vicino, o comunque a portata di mano) un bottone o un link che permette ad un utente di registrarsi. Poiché all’interno di questo progetto potranno accedervi solo persone autorizzate, si è deciso insieme al gruppo di ricerca che nessuno sarà in grado di registrarsi autonomamente.
  Quando un nuovo Dottore, Professore, Ricercatore o studente dovrà accedere alla Web Application dovrà richiedere esplicitamente all’amministratore di aggiungere una registrazione per la sua utenza.

- **Login**: Una volta che il Dottore, Professore, Ricercatore o studente è stato autorizzato dall’Amministratore ad accedere alla Web Application, potrà effettuare il login utilizzando un codice ed una password fornitagli dall’Amministratore. Per quanto riguarda la password al primo accesso, per motivi legati alla sicurezza, verrà richiesto di cambiarla.

- **Utenti**: Insieme al gruppo di ricerca sono stati pensati quattro gruppi di utenti, tra cui:

  - **Amministratore**: persona incaricata di gestire la creazione e l’eliminazione di tutti i gruppi di utenti. Chi ricopre questo incarico è quindi in grado di fornire l’accesso con vari livelli di interattività a chi ne ha bisogno.

  - **Super Ricercatore**: persona che potrà inserire ed eliminare gli esami di qualsiasi paziente. Questo ruolo è stato pensato per avere un controllo generale e per poter modificare i dati di ricercatori non più presenti all’interno del progetto.

  - **Ricercatore**: persona che potrà inserire qualsiasi tipo di esame, ma che potrà soltanto modificare ed eliminare quelli da lui inseriti. In questo modo è possibile fornire a certi ricercatori permessi ristretti.

- **Studente**: durante la fase di progettazione, insieme al gruppo di ricerca è stato individuato un tipo di utente che potrà risultare molto utile. Si è deciso di dare all'account "studente" la possibilità di visualizzare tutti i dati e di poterli scaricare. In questo modo sarà possibile fornire ad alcuni alunni dei professori coinvolti in questa ricerca la possibilità di accedere al sistema in "sola lettura".

3.2.2. Funzionalità aggiuntive

Le funzionalità aggiuntive sono tutte quelle che sono state richieste per personalizzare e rendere facilitare l'interazione con l'applicazione, esse sono:

- **Schema degli esami**: come richiesto dal gruppo di ricerca, è stato progettato lo schema di visualizzazione degli esami del singolo paziente in modo da soddisfare al meglio le loro esigenze. Dopo varie bozze, è stato deciso come schema definitivo quello che si può vedere nella figura 3.1.

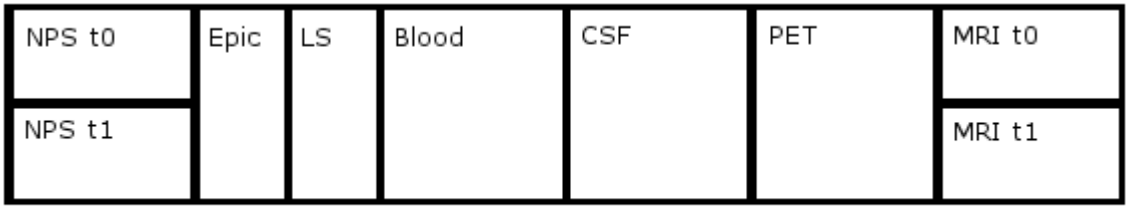


Figura 3.1: Schema esami in bianco e nero.

Il prodotto finale è stato reso ancora più user-friendly colorando un esame soltanto quando al suo interno vengono effettivamente salvati dei dati utilizzando colori diversi per ogni esame. In questo modo è possibile capire a colpo d'occhio quali esami sono stati immessi e quali lo devono ancora essere.
Nella figura 3.2 è possibile capire facilmente che i dati relativi all'esame "NPS t1" devono ancora essere inseriti.

Figura 3.1: Schema esami a colori.

Arrivati a questo punto è stato necessario aggiungere i dati clinici e demografici (i quali non sono veri e propri esami) e fare in modo che questa tabella fosse presente per ogni paziente. Il risultato finale per un singolo paziente si può vedere nella figura 3.3.

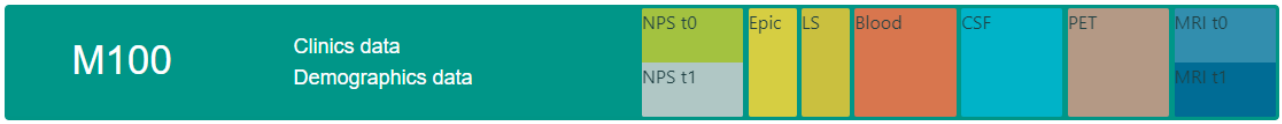


Figura 3.1: paziente con il proprio schema degli esami.

- **Download dei dati**: per quanto riguarda la possibilità di scaricare i dati relativi ad un esame sono state progettate e realizzate due opzioni: la prima permette di scaricare tutti i dati relativi ad un esame di tutti i pazienti.
  La seconda, permette di selezionare soltanto i campi che si vogliono scaricare.
  Per agevolare quest'ultima è stata inserita una checkbox che permette di selezionare e deselezionare tutti gli attributi che si vogliono scaricare, rendendo di fatto più veloce un download di pochi campi.

- **Ordinamento pazienti:** L'ordinamento dei pazienti è stato pensato per poter gestire al meglio la pagina principale (ovvero quella che contiene la lista di pazienti con esami annessi). Gli ordinamenti ideati sono stati i seguenti:
  - Data firma consenso al trattamento dei dati più recente (senza la firma presente non è possibile per motivi di privacy inserire dati di esami).
  - Codice identificativo del paziente (in modo da raggruppare i pazienti reclutati a Reggio Emilia e quelli reclutati a Modena).
  - Esami svolti (NPS, Epic, LS, Blood, CSF, PET, MRI) in modo da avere in alto tutti quelli che hanno effettuato un certo esame.

3.3. Progettazione dell'interfaccia della Web Application

Un punto che molte volte viene sottovalutato durante la progettazione di una Web Application, è la usability (o usabilità) dell'interfaccia grafica.
Anche per quanto riguarda questo punto, si sono rivelati fondamentali i feedback continui del gruppo di ricerca.
Quello che si è cercato di fare è stato limitare le pagine annidate. Tutte le pagine presenti sono state pensate per essere il più accessibili possibile in modo da rendere fluida la navigazione e l'utilizzo generale della Web App.
Partendo dalla home, tutto è accessibile con pochi click, come è possibile vedere nei diagrammi dei casi d'uso all'interno delle figure 3.2 (Super Ricercatore e Ricercatore), 3.3 (Studente), 3.4 (Amministratore).

3.3.1. Struttura dell'interfaccia grafica

In questo paragrafo, vengono elencate le componenti principali dell'interfaccia grafica della Web App.

- Home (elenco dei pazienti)
- Pagina Inserimento/modifica dati esame
- Pagina Download
- Pagina Download esame

3.3.2. Casi d'uso

I casi d'uso (o use case) in informatica indicano i "modi" nei quali un utente può utilizzare il sistema. Questi possono essere rappresentati come scenari di interazione che coinvolgono un utilizzatore e il sistema.
I casi d'uso non devono essere scomponibili in casi d'uso più piccoli.
Tramite questa tecnica è possibile progettare la soluzione finale basandosi sull'interazione che si vorrà avere, lasciando da parte tutte le attività interne e di contorno.
Questo step dello studio di progettazione è molto importante poiché va a definire in modo abbastanza delineato quella che è l'usabilità del sistema (o usability).
I punti fondamentali per spiegare lo use case diagram progettato sono cinque:

- **Attore:** colui che esegue i casi d'uso presenti all'interno del sistema. Solitamente rappresenta un utente che interagisce con il sistema (può essere anche un dispositivo hardware o un altro sistema).
- **Freccia <<extend>>:** freccia dal caso che descrive l'evento alternativo al caso "standard".
- **Freccia <<include>>:** freccia dal caso "chiamante" al caso che descrive le azioni da includere.
- **Caso d'uso:** elemento che dà una rappresentazione visuale di una funzionalità distinta e non scomponibile in un sistema (di solito raffigurata da una figura ovale).
- **Sistema:** elemento che racchiude tutti i casi d'uso che definiscono il sistema. Ogni cosa presente al suo interno rappresenta una funzionalità diversa (generalmente raffigurato tramite un quadrato).

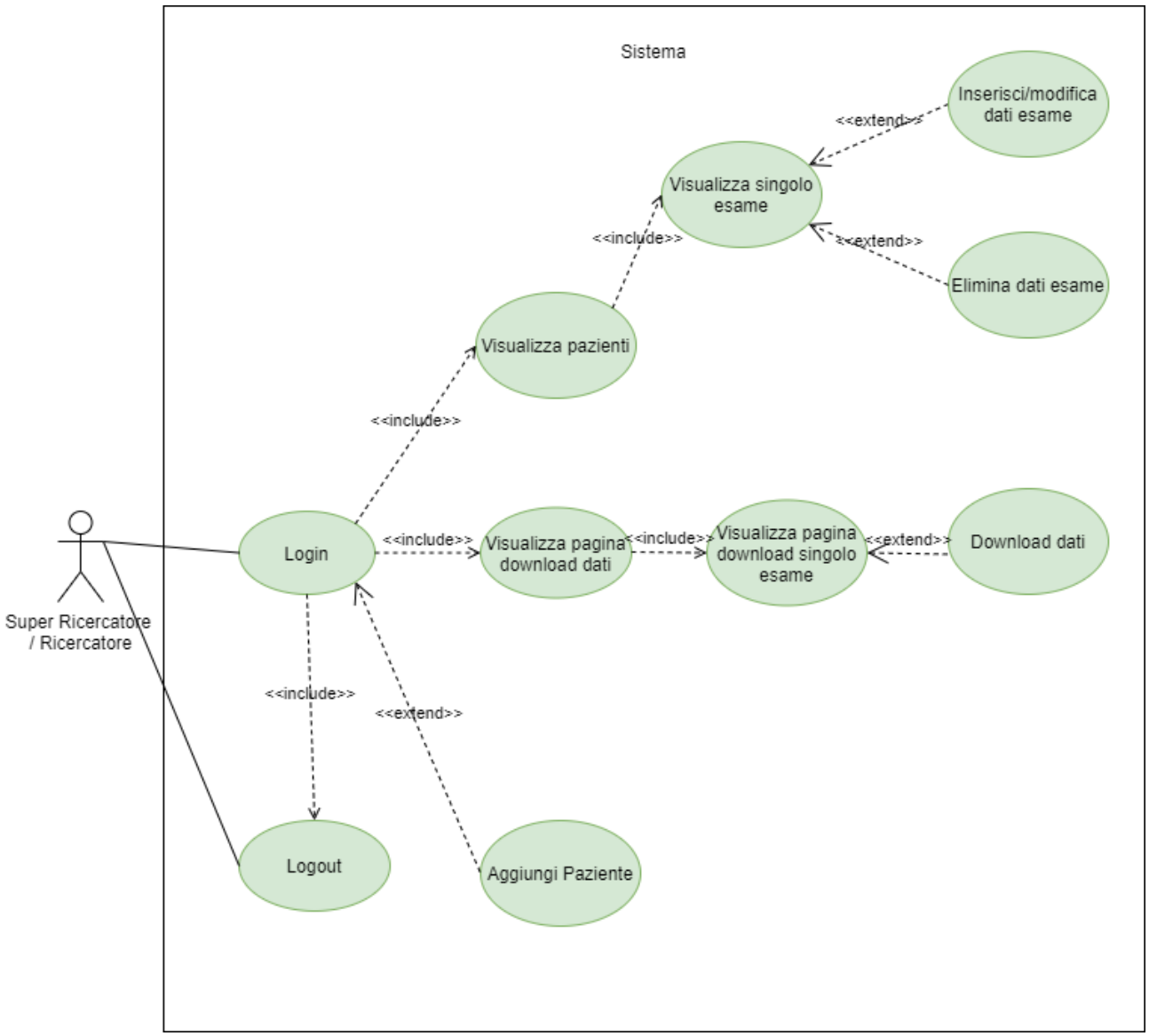


Figura 3.2: diagramma casi d'uso Super Ricercatore e Ricercatore.

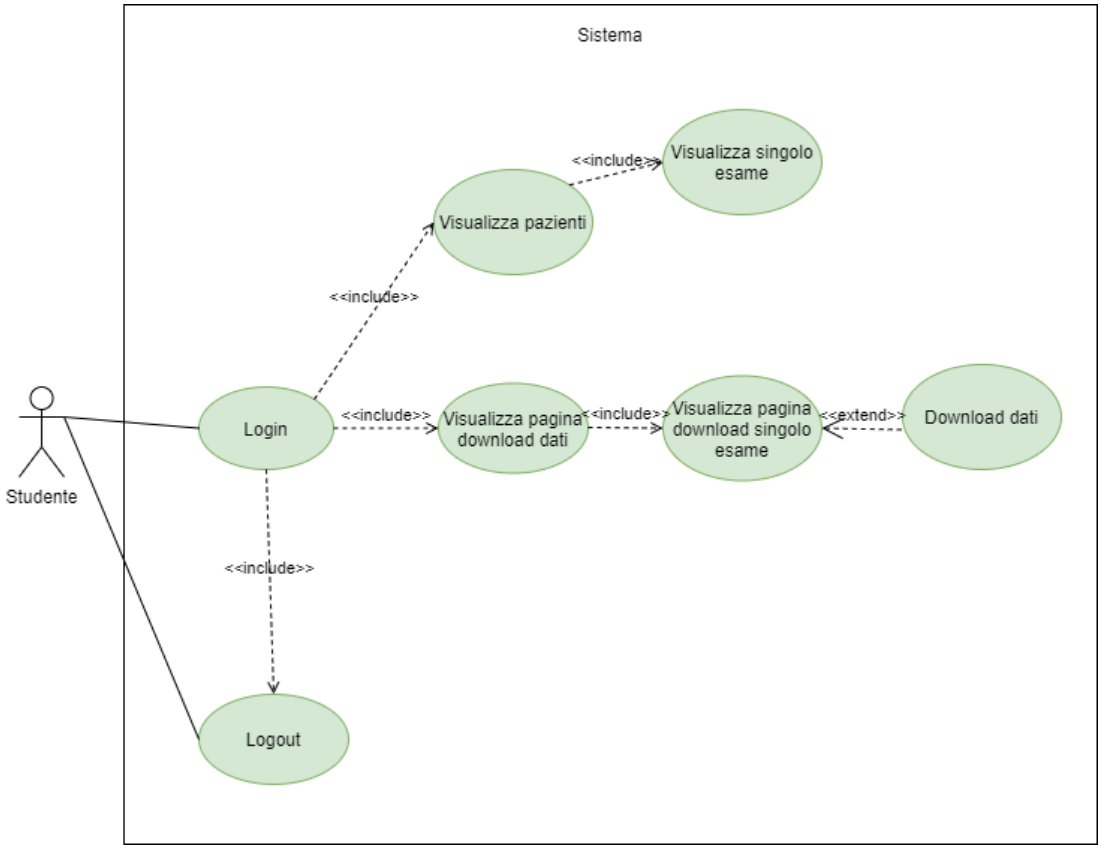


Figura 3.3: diagramma casi d'uso Studente.

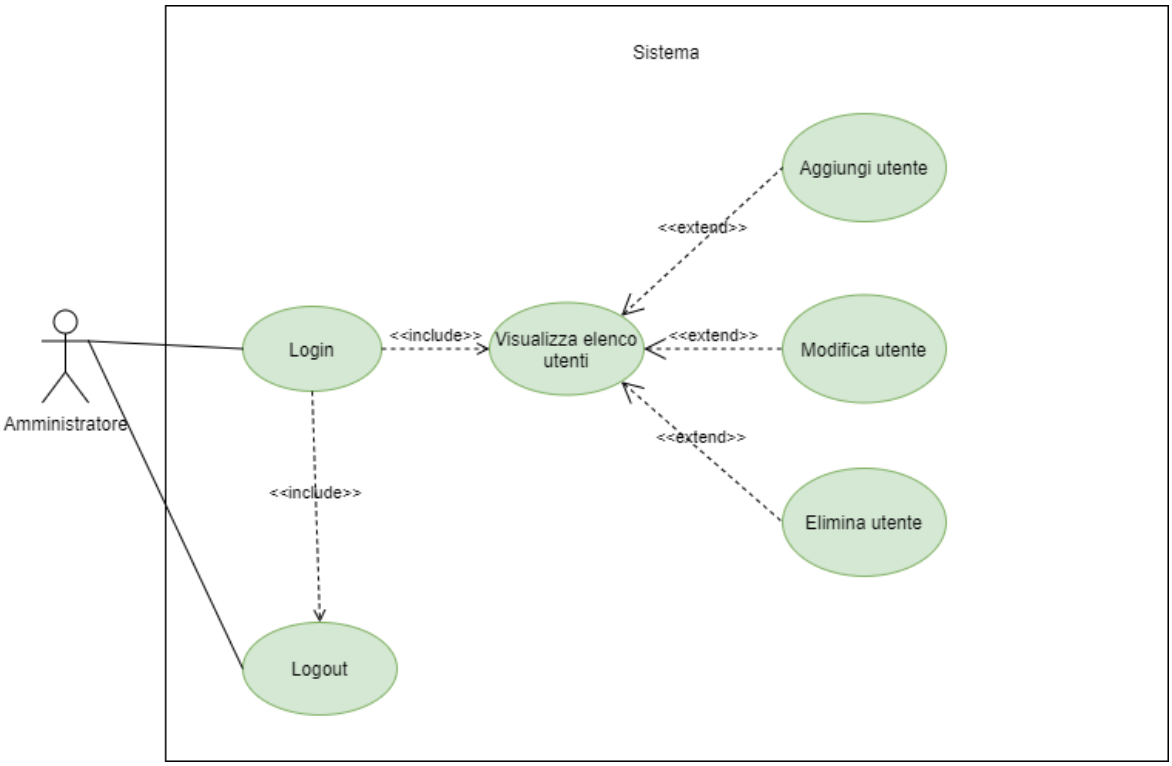


Figura 3.4: diagramma casi d'uso Amministratore.

3.3.3. Spiegazione casi d'uso

Di seguito vengono illustrati nello specifico due scenari differenti per un utente Ricercatore.

Aggiunta dei dati dell'esame "Blood" al paziente "M003"

- Il Ricercatore deve aver effettuato con successo il Login.
- Il Ricercatore cerca il paziente M003 aiutandosi eventualmente tramite i diversi ordinamenti possibili.
- Clicca la casella contenente il testo "Blood" del paziente M003.
- Inserisce i dati all'interno del form che gli viene presentato.
- Infine, clicca il pulsante invio per completare l'operazione di immissione.
    - In caso di problemi in fase di inserimento verrà mostrato un messaggio di errore.
    - In caso di avvenuto inserimento, verrà ricaricata automaticamente la stessa pagina con la visualizzazione dei dati immessi in modo da confermare l'immissione.
- A questo punto il ricercatore può decidere di modificare i dati (attraverso l'apposito bottone) o tornare alla home (tramite il bottone indietro o cliccando la casella che contiene il nome dell'applicazione).

Download dati relativi all'esame "Blood"

- Il Ricercatore deve aver effettuato con successo il Login.
- Clicca sul bottone "Download data" il quale porta ad una pagina dove è possibile scegliere quale esame scaricare.
- Clicca la casella contenente il testo "Blood" la quale caricherà una pagina contenente un form con tutti gli attributi (affiancati da una checkbox) relativi al suddetto esame.
    - Nel caso il Ricercatore voglia scaricare pochi attributi, potrà deselezionarli tutti in un solo momento e selezionare di conseguenza soltanto quelli desiderati.
    - Nel caso voglia scaricare tutti gli attributi, gli basterà cliccare sul bottone "download", salvando sulla macchina del Ricercatore un file CSV contenente tutti gli attributi.

3.3.4. Conoscenze per la realizzazione

Per realizzare quanto riportato nei vari diagrammi durante la fase di progettazione sono state individuate cinque tecnologie, la cui conoscenza risulta fondamentale per la creazione di questi:

- Conoscenza del linguaggio di markup HTML per poter creare la struttura delle pagine.
- Conoscenza del linguaggio CSS per personalizzare e gestire al meglio il layout della pagina.
- Conoscenza del linguaggio JavaScript per poter inserire effetti dinamici e interattivi. Gli eventi di natura dinamica vengono innescati da certe azioni che l'utente esegue.
- Si è deciso di integrare l'utilizzo di Bootstrap (framework che comprende un insieme di elementi grafici, stilistici, di layout e Javascript pronti all'uso), in modo da poter inserire velocemente elementi come la "navbar".
- Conoscenza del linguaggio di programmazione Python utilizzato dal framework Flask per la gestione del back end.

3.4. Progettazione del database

La progettazione del database è a tutti gli effetti uno dei passi fondamentali, se non il più importante. Essendo responsabile dell'organizzazione logica dei dati, in questo paragrafo vengono sviscerati quelli che sono gli aspetti fondamentali del database e vengono spiegati i motivi di tutte le decisioni che hanno portato alla conclusione della progettazione di questa parte di progetto.

3.4.1. Motivi della scelta

Le decisioni che hanno portato all'utilizzo di un document database sono state molteplici.
Il motivo più importante si può trovare nella dimensione potenzialmente esponenziale del volume di dati. Per il momento, questa ricerca nasce sul territorio di Modena e Reggio Emilia. L'idea per il futuro è quella di espanderla a livello regionale ed eventualmente ampliarla sempre di più.
Nel caso un giorno si riuscisse ad estendere questo progetto, sarà a tutti gli effetti disponibile una base solida, la quale, oltre la capacità di gestire una grande mole di dati, potrà essere distribuita su più server grazie al metodo di sharding.
Un altro motivo che ha portato a questa scelta progettuale è stata la richiesta di poter immettere nuovi attributi all'interno del database. Essendo questo progetto ancora in una fase iniziale, non è stato possibile definire al cento per cento quali saranno (per esempio tra 2 anni) gli attributi necessari da

memorizzare. Grazie all'utilizzo di un document database, questo non risulta un problema, poiché con poche operazioni è possibile aggiungere qualsiasi tipo di dato (situazione ben differente nel caso di un database relazionale).

Per quanto riguarda MongoDB, esso è stato scelto dopo un'attenta analisi della tecnologia per diversi motivi: facilità di comprensione per essere una nuova tecnologia, supporto fornito, documentazione vasta ed esplicativa, percentuale di adozione maggiore rispetto al competitor principale ArangoDB.
Le altre tecnologie prese in considerazione quali: Amazon dynamoBD, Azure Cosmos DB, Amazon Document DB, sono state scartate a priori poiché proprietarie rispettivamente di Amazon (le prime due) e Microsoft.

3.4.2. Progettazione in MongoDB

Una delle caratteristiche principali di MongoDB è la possibilità di poter inserire all'interno delle collezioni documenti annidati sotto forma di oggetto.
Tale tecnica, permette ad ogni documento (corrispondente di una tabella nei database relazionali), di contenere altri documenti, creando di conseguenza un legame tra i due (simile ad una relazione tramite chiave esterna).
L'effetto ottenuto tramite questa operazione è concettualmente rappresentabile da una matrioska.
Questo approccio ha portato alla creazione di dieci collezioni (oltre a quelle di default per gli utenti e per la memorizzazione dei file), strutturate nel seguente modo:

- **Patient**: i Documenti al suo interno, che rappresentano i singoli pazienti, contengono tutte le informazioni necessarie per ciascun paziente. Tali informazioni sono: codice paziente, data firma consenso e dieci variabili booleane che permettono di sapere se un esame è già stato inserito o meno.

  Queste variabili sono essenziali per velocizzare la pagina home della Web Application. Nel progettare la home come richiesto dal gruppo di ricerca, si è palesata subito la problematica di colorare le label degli esami già inseriti. Grazie a queste 10 variabili booleane, non è necessario scorrere tutte le collezioni e controllare ogni singolo esame per sapere se è stato inserito o meno, riducendo drasticamente i tempi di risposta e il carico di lavoro.

Questa soluzione risulta estremamente utile anche per poter effettuare gli ordinamenti richiesti in modo veloce per gli stessi motivi sopracitati.
Di seguito saranno elencate le collezioni che contengono gli esami di tutti i pazienti.

- **NPS0**: contiene tutti i documenti relativi all'esame NPS eseguito al tempo zero.
- **NPS1**: contiene tutti i documenti relativi all'esame NPS eseguito al tempo uno.
- **MRI0**: contiene tutti i documenti relativi all'esame MRI eseguito al tempo zero.
- **MRI1**: contiene tutti i documenti relativi all'esame MRI eseguito al tempo uno.

I documenti memorizzati all'interno delle collezioni il cui nome termina con "1" contengono gli stessi attributi di quelli memorizzati all'interno delle collezioni il cui numero termina con "0".
Questa scelta di tenerli divisi è stata pensata per un fattore di velocità e comodità in quanto il gruppo di ricercatori ha la necessità di inserirli, modificarli e visualizzarli sempre separati.
Tenerli divisi permette inoltre di costruire la logica della Web Application trattandoli come tutti gli altri esami. Questo fa sì di avere maggiore velocità e maggiore riutilizzo del codice (in quanto non vi è stato il bisogno di creare metodi ad hoc).

- **Epic**: contiene tutti i documenti relativi al questionario sulle abitudini alimentari.
- **LS**: contiene tutti i documenti relativi al questionario "life style".
- **Blood**: contiene tutti i documenti relativi all'esame Blood.
- **Pet**: contiene tutti i documenti relativi all'esame PET.
- **CSF**: contiene tutti i documenti relativi all'esame CSF.

Per ogni esame è stata creata una collezione diversa per motivi di velocità di recupero dati.
Ogni documento di un esame, oltre agli attributi relativi all'esame stesso, contiene il codice del paziente e il codice del ricercatore che lo ha creato.
Grazie a questa divisione, sarà necessario conoscere soltanto il codice del paziente per recuperare un certo esame.
Riportiamo un esempio per comprendere al meglio qual è il reale vantaggio.

Poniamo di dover visualizzare i dati relative all'esame CSF del paziente "M001". Una volta individuato quest'ultimo all'interno della home, si dovrà cliccare la label contenente il nome CSF. A

questo punto, grazie alla struttura dell'applicazione, conoscendo il codice del paziente per il quale vogliamo recuperare l'esame, sarà sufficiente effettuare una ricerca all'interno della collezione "CSF".

All'interno della figura 3.5 viene illustrata la struttura di una collezione (in questo caso la collection "patient").

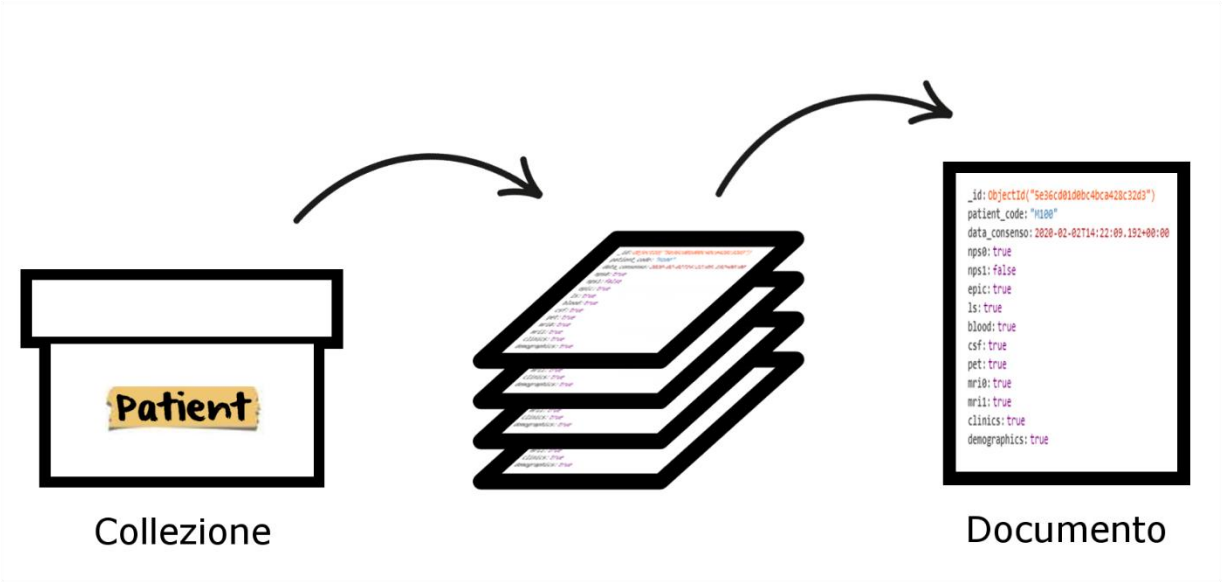


Figura 3.5: rappresentazione struttura collection.

È possibile comprendere al meglio la struttura completa del database attraverso la figura 3.6.

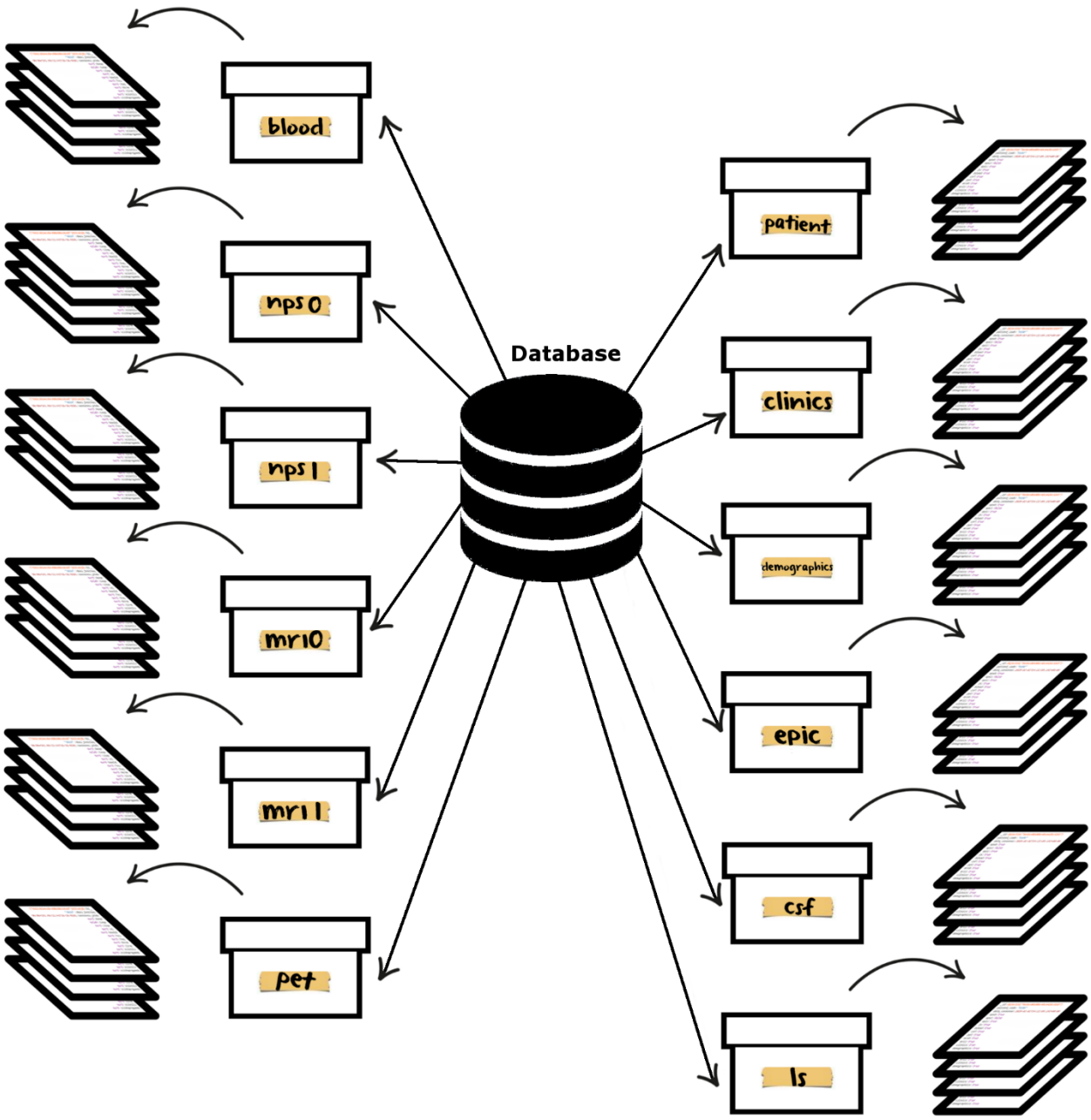


Figura 3.6: schema database.

Un lato negativo di questa divisione lo si può riscontrare nel download dei dati. Per motivi legati alle elaborazioni effettuate dei ricercatori, quando viene richiesto il download dei dati relativi ad un certo esame, questo dovrà contenete tutti gli utenti, anche quelli che non lo hanno svolto (lasciando vuoti tutti gli attributi). Questo è necessario ai ricercatori per poter lavorare su file con un numero di righe uguali.

Questo svantaggio è però considerato accettabile in quanto la funzione di download sarà una parte del sistema utilizzata molto raramente rispetto alla parte di inserimento e visualizzazione.

3.4.3. Casi alternativi di collezioni

Di seguito vengono esposti i tre casi alternativi presi in considerazione per la struttura delle collezioni, spiegando i motivi per cui non sono stati scelti come configurazione finale.

Caso alternativo 1

Collection “NPS0” unita a “NPS1” e “MRI0” unita a “MRI1”.
Questa configurazione è stata scartata in quanto rendeva obbligatorio creare metodi per la gestione di questi esami diversi rispetto a quelli utilizzati per i restanti cinque esami. Poiché MongoDB gestisce molto bene un numero così basso di collezioni, si è deciso di mantenerle separate.
Un semplice schema del database risultante è riportato all’interno della figura 3.7.

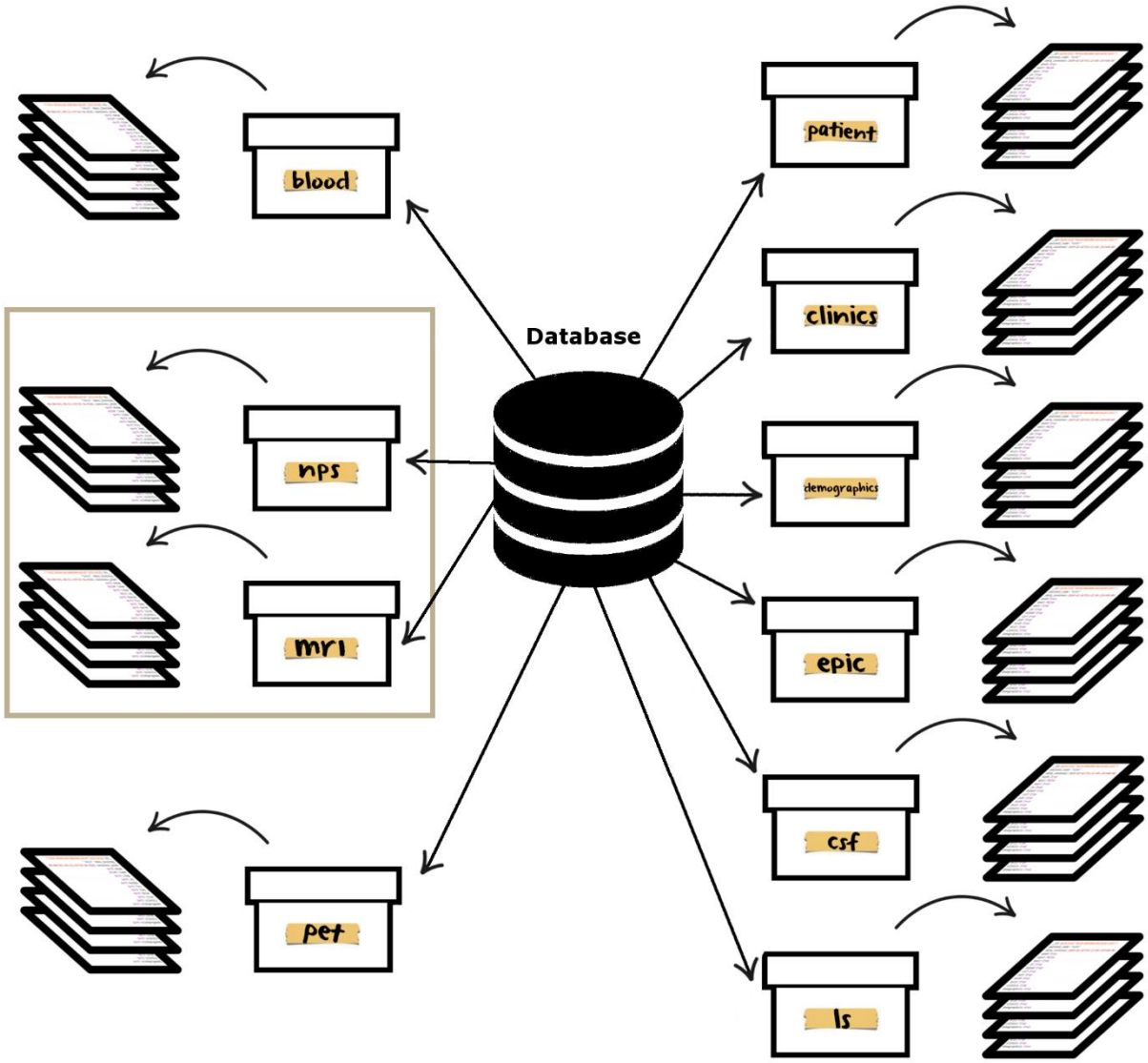


Figura 3.7: schema database alternativo 1.

Caso alternativo 2

Creazione di una singola collezione chiamata Pazienti, la quale contiene la lista dei pazienti ed ognuna di queste contiene a sua volta la lista dei propri esami.

Tale configurazione è stata scartata poiché in uno sviluppo futuro, effettuare ricerche o statistiche divise per esame diventerebbe sempre più lento con il crescere del numero dei pazienti. Questo accadrebbe poiché, se si volessero recuperare tutti i documenti di un determinato esame, sarebbe necessario scorrere tutti i pazienti e recuperare quel determinato esame all'interno di esso.

Un semplice schema del database risultante è riportato all'interno della figura 3.8.

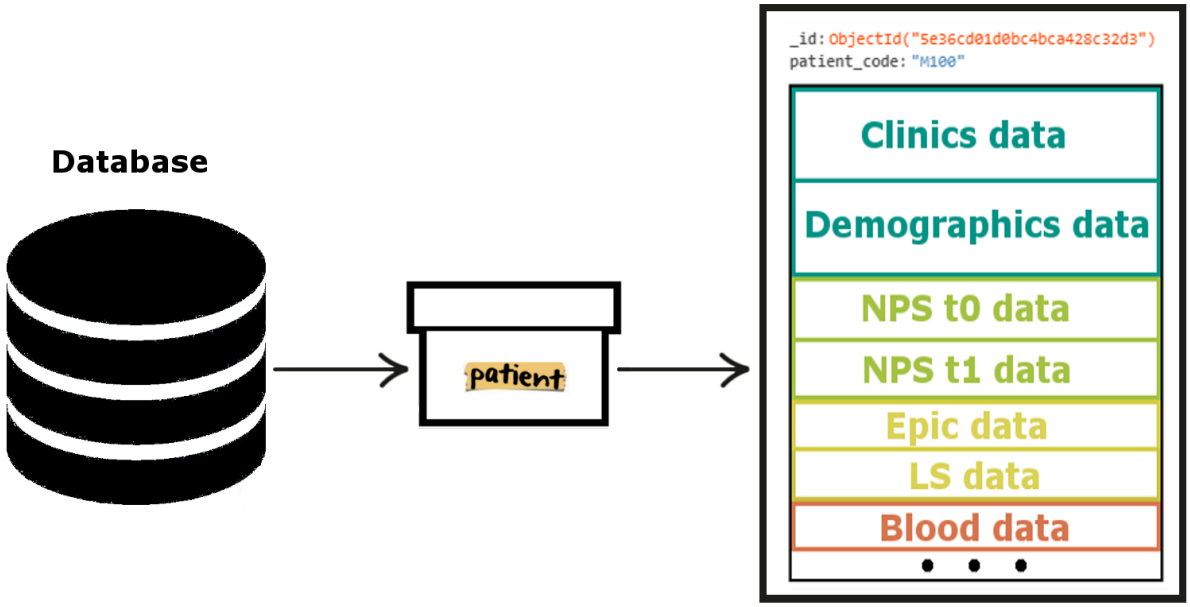


Figura 3.8: schema database alternativo 2.

Caso alternativo 3

Creazioni di solo due collezioni: Pazienti ed Esami.

Questa configurazione è stata esclusa poiché come nel “caso alternativo 2” in uno sviluppo futuro, effettuare ricerche o statistiche divise per esame diventerebbe sempre più lento con il crescere del numero dei pazienti.

Nel caso in cui venissero ricercati tutti i documenti inerenti ed un determinato esame, sarebbe necessario scorrere tutti quelli presenti nella collezione "esami". Questo rappresenta un problema poiché con il crescere della collezione diventerebbe un’operazione sempre più lenta e onerosa dal punto di vista computazionale.

Un semplice schema del database risultante è riportato all’interno della figura 3.9.

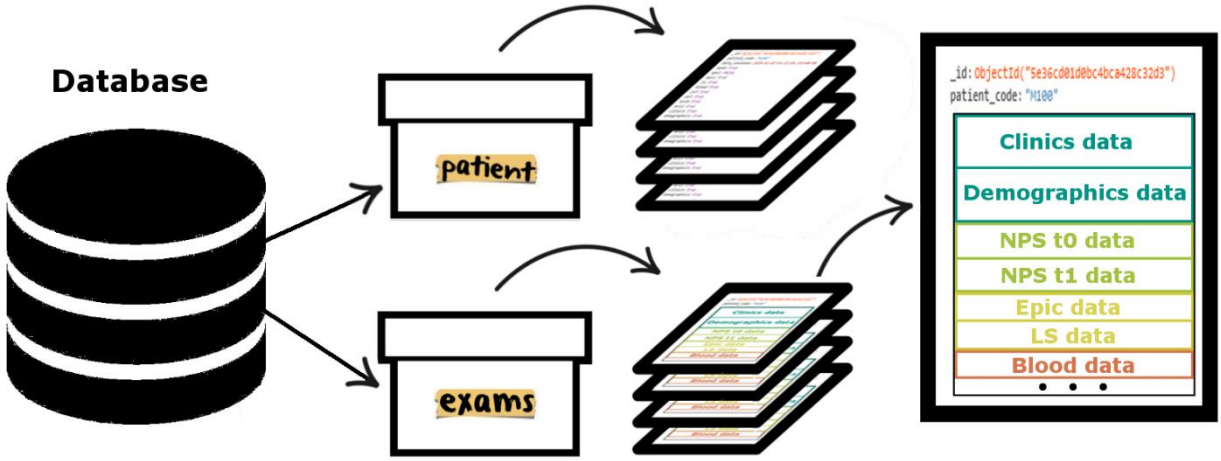


Figura 3.9: schema database alternativo 3.

3.4.4. Collezione Patient

La collezione che conterrà i documenti dei pazienti è stata progettata con l'unico dato ammesso dal gruppo dei ricercatori per quanto concerne l'identificazione del paziente, cioè il codice. Come anticipato non è possibile per questioni di privacy memorizzare informazioni personali come per esempio la residenza.

Oltre al codice, ogni paziente avrà dieci variabili booleane che attesteranno l'avvenuto inserimento di un determinato esame da lui svolto.

All'interno della figura 3.10 è possibile visualizzare uno schema della collezione.

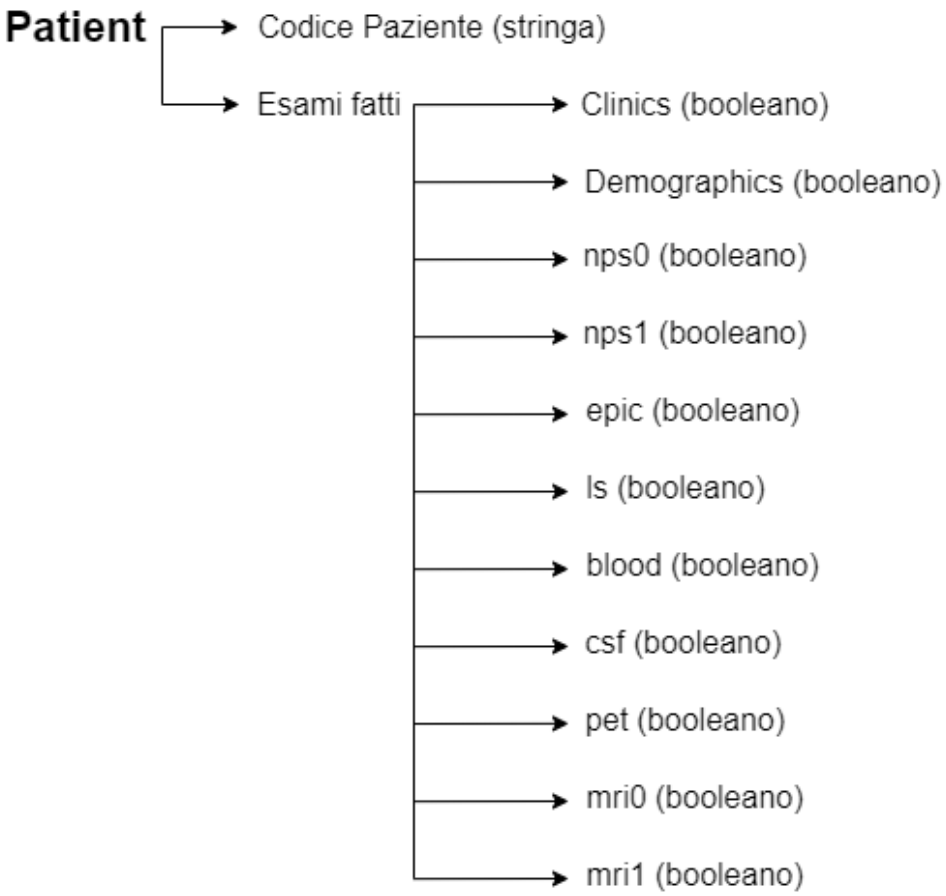


Figura 3.10: progettazione collezione patient.

3.4.5. Collezione Clinics

Per quanto riguarda la collection che conterrà i dati clinici, i dati che verranno inseriti sono visualizzabili all'interno della figura 3.11.

Per quanto riguarda l'attributo "Altre diagnosi" sarà resa disponibile una scelta multipla unita con possibilità di aggiungere altre diagnosi all'interno di un campo di testo libero.

Il campo "Terapie" sarà un campo a scelta multipla dove ognuna avrà un campo di testo libero per scrivere le annotazioni.

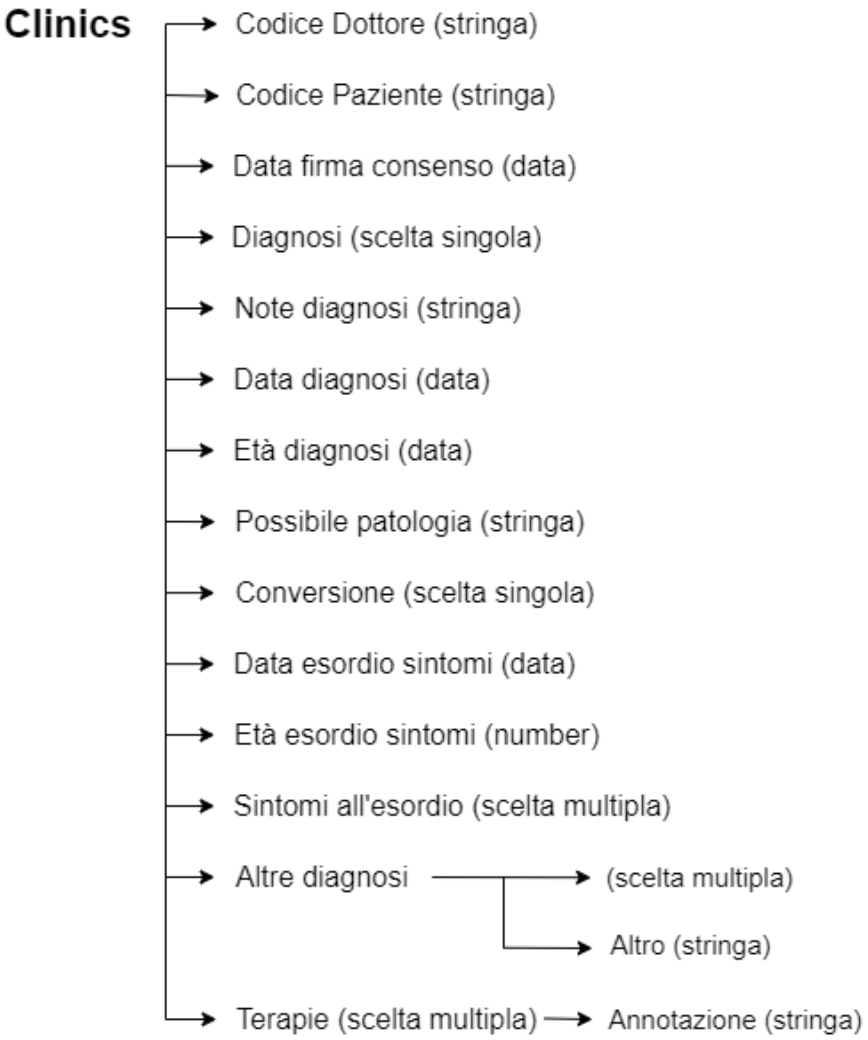


Figura 3.11: Progettazione collezione clinics.

3.4.6. Collezione Demographics

I dati demografici da poter inserire possono essere visualizzati all'interno della figura 3.12.
Il dato "Caregiver" (badante) permetterà oltre alla scelta multipla, di inserire un'ulteriore o diversa persona.

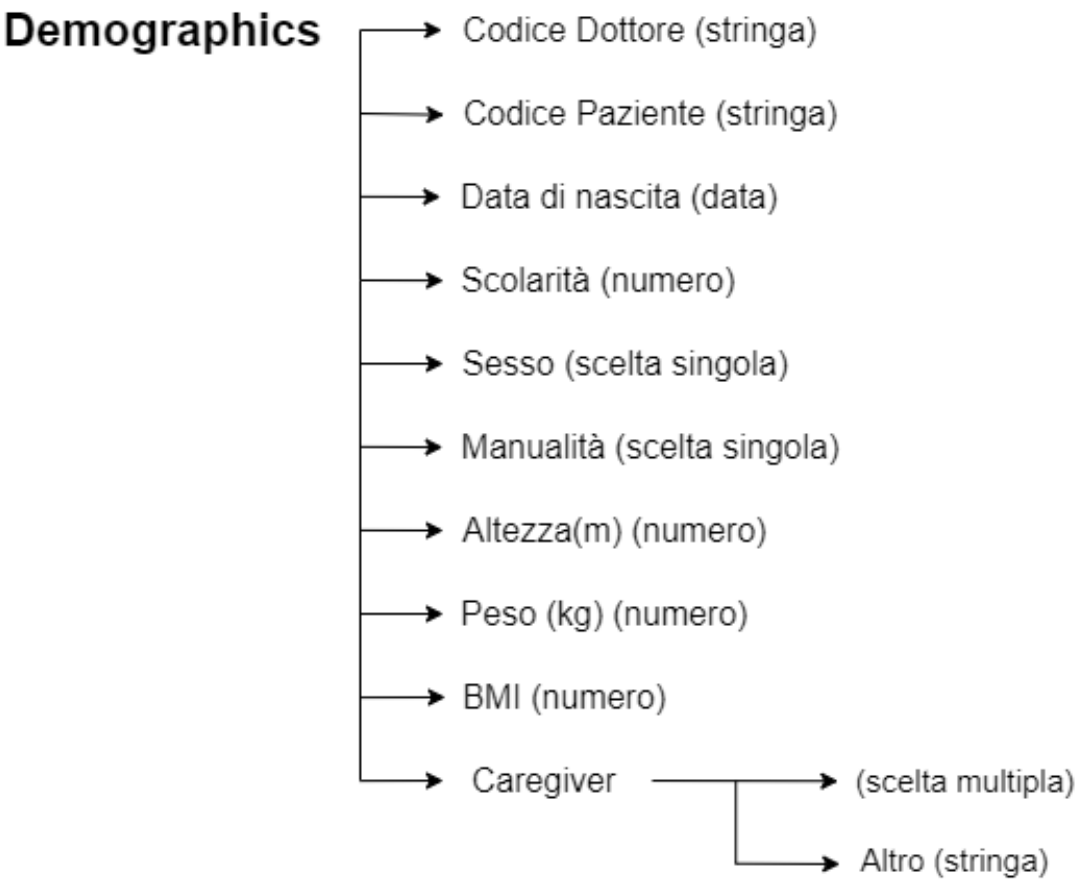


Figura 3.12: Progettazione collezione demographics.

3.4.7. Collezione NPS

La collezione "nsp" si divide in due esami specifici "Test di screening" e "scale funzionali", questi comprendono varie sottocategorie ognuna delle quali contiene un punteggio grezzo "PG". Oltre a questo punteggio, ci sono delle sottocategorie che contengono un punteggio corretto "PC" e un valore booleano che risulta "TRUE" nel caso in cui il punteggio corretto sia minore o maggiore di una certa soglia specificata.
I campi "PG" e "PC" conterranno valori numerici che dovranno rispettare un range di valori.
Lo schema di questa collection è visualizzabile all'interno della figura 3.13.

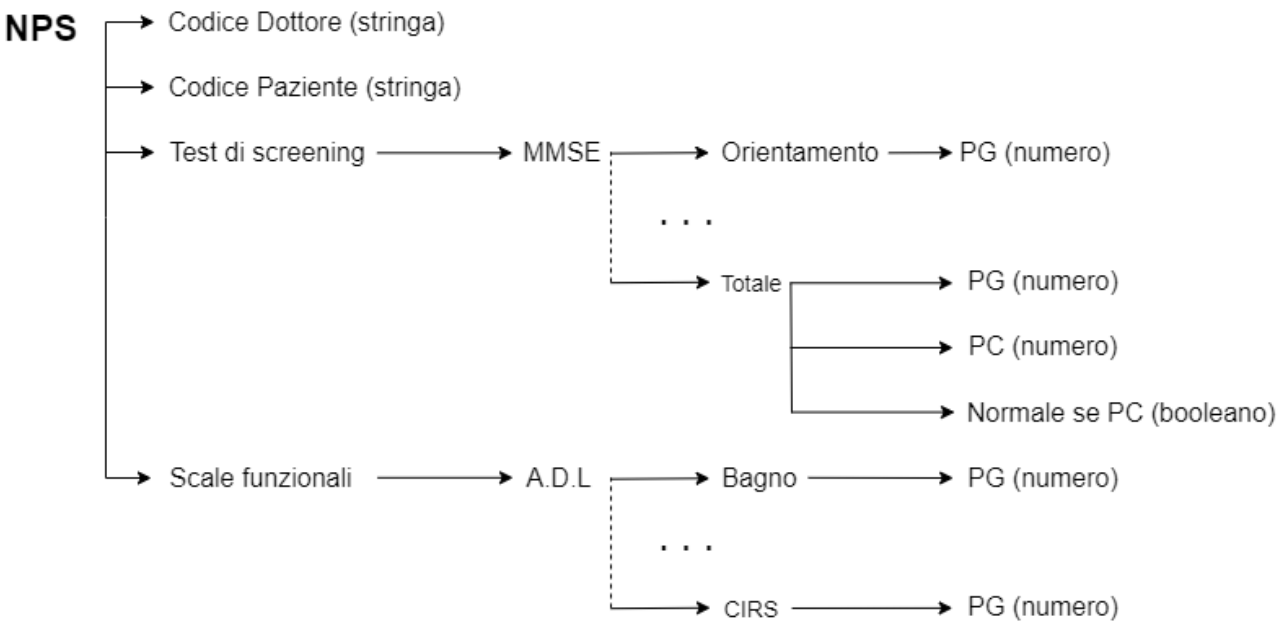


Figura 3.13: Progettazione collezione nps.

3.4.8. Collezione Epic

Per quanto riguarda il questionario "Epic", come anticipato, verranno salvati soltanto i valori delle variabili. Di conseguenza, la struttura sarà molto semplice in quanto conterrà oltre al codice del paziente e del dottore che inserirà i dati, circa seicento variabili di tipo numerico.

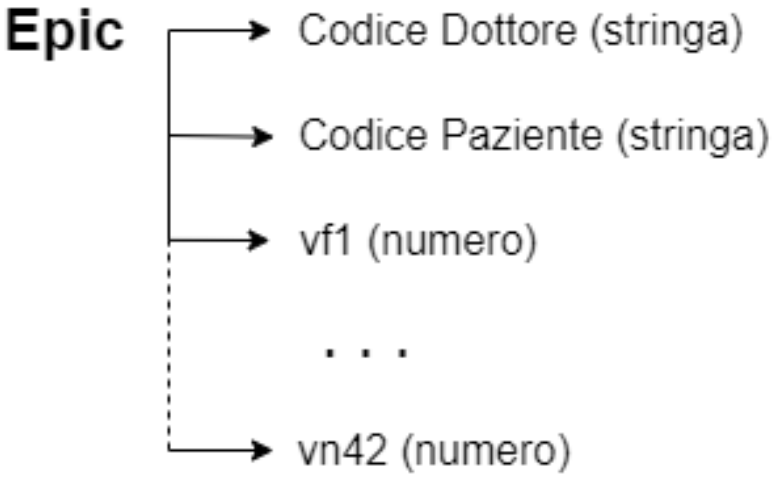


Figura 3.14: Progettazione collezione epic.

3.4.9. Collezione LS

Similmente alla collezione “Epic”, conterrà oltre al codice paziente e quello del dottore, circa trecento variabili di tipo stringa riguardati il questionario “life style”.
All’interno della figura 3.15 è possibile visualizzare la struttura della collection in questione.

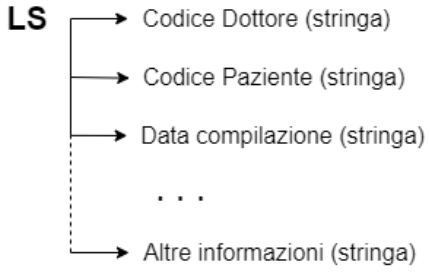


Figura 3.15: Progettazione collezione ls.

3.4.10. Collezione Blood

Per quanto riguarda la collezione “blood”, essa viene logicamente divisa in due. La prima parte riguarda i dati clinici, all’interno dei quali (figura 3.16) è possibile notare una struttura annidata. “Screening malattie neurodegenerative” conterrà tre variabili di tipo numerico e un oggetto “ricerca mutazioni genetiche” conterrà diciotto variabili di tipo stringa.
La seconda parte riguarda i dati epidemiologici, i quali sono costituiti da dodici variabili di tipo numerico.

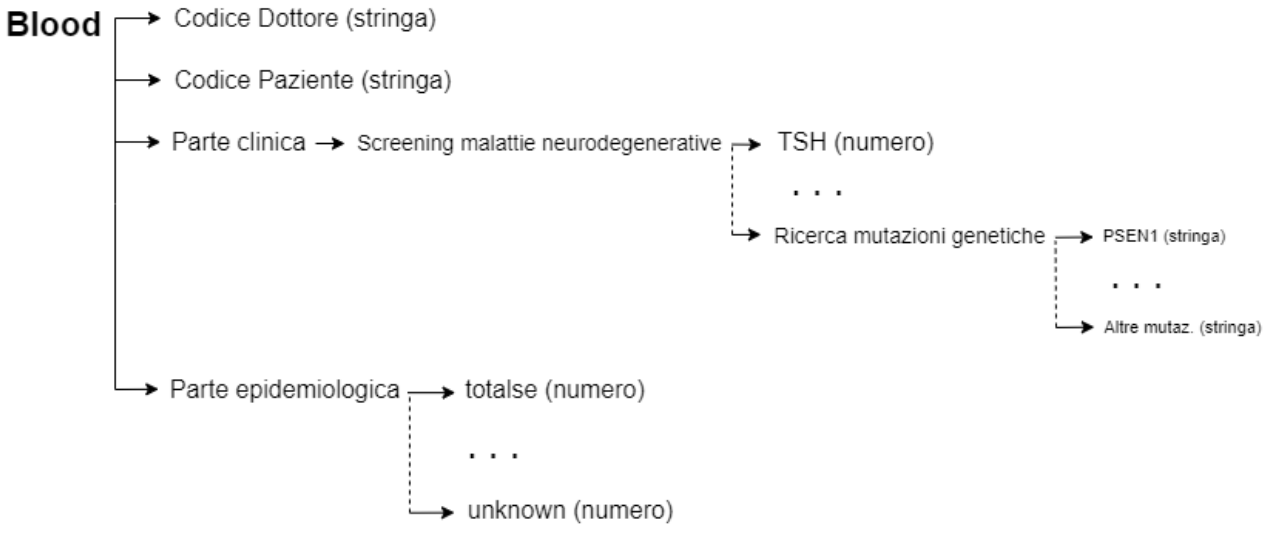


Figura 3.16: Progettazione collezione blood.

3.4.11. Collezione CSF

La collezione “csf” è strutturata in modo simile alla collection “blood”. Anche questa viene divisa logicamente in due parti. La prima parte tratta i dati clinici mentre la seconda i dati epidemiologici. La struttura di questa collection si può visualizzare all’interno della figura 3.17.

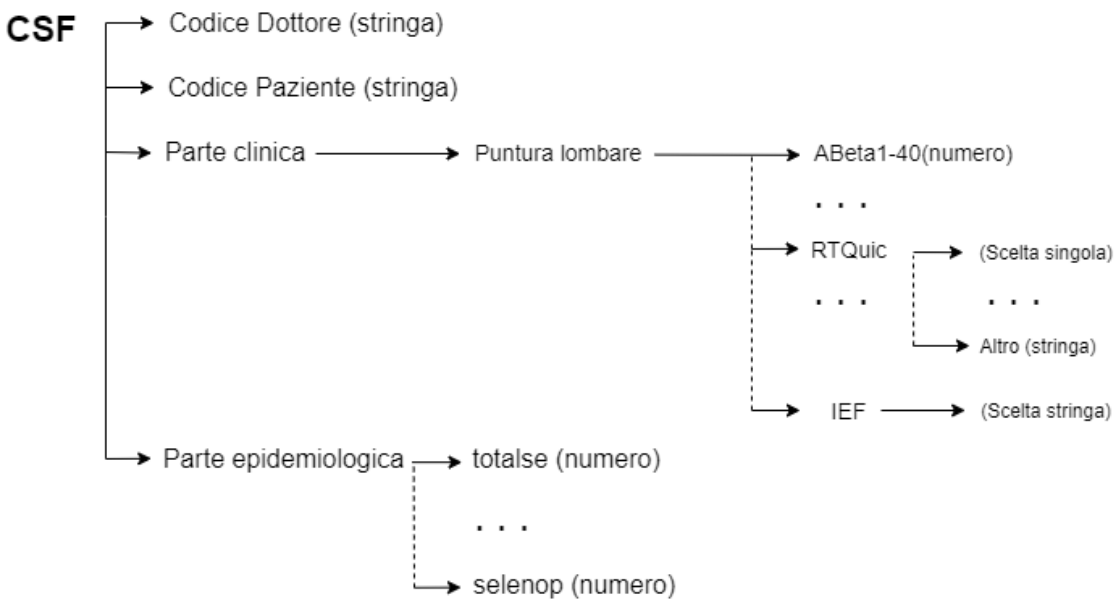


Figura 3.17: Progettazione collezione csf.

3.4.12. Collezione PET

Per quanto riguarda la collezione “pet”, è stata progettata in modo da poter salvare (oltre al codice paziente e dottore) una stringa contenente l’indirizzo all’interno del quale si trova la directory contenente le immagini del suddetto esame.

Oltre al campo “IEF” a scelta singola, è stata inserita la possibilità di caricare file con annotazioni testuali collegate.

La struttura di questa collezione è visualizzabile all’interno della figura 3.18.

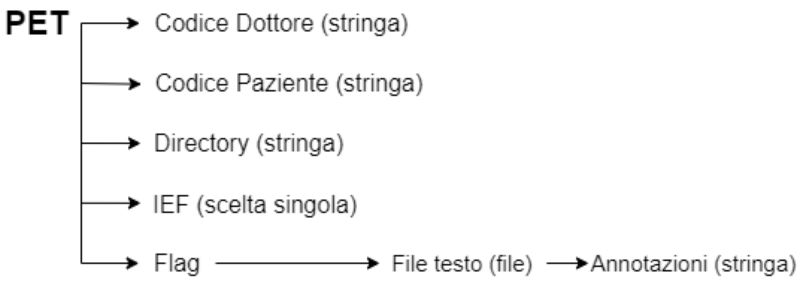


Figura 3.18: Progettazione collezione pet.

3.4.13. Collezione MRI

La collezione “mri” memorizza (oltre al codice paziente e al codice dottore) una stringa contenente l’indirizzo all’interno del quale si trova la directory contenente le immagini del suddetto esame.
Inoltre, contiene sette flag, ognuno dei quali può contenere uno o più file con un’annotazione testuale collegata.
La struttura di questa collezione è visualizzabile all’interno della figura 3.19.

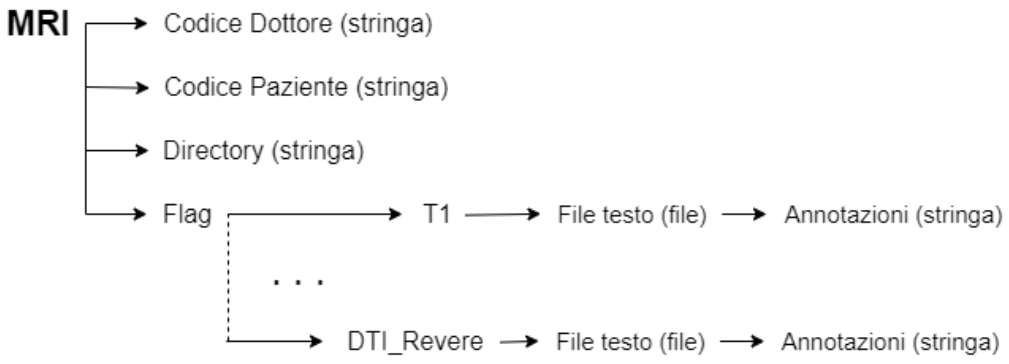


Figura 3.19: Progettazione collezione mri.

3.4.14. Schema di validazione

Successivamente alla decisione della struttura di collezioni e documenti, è stato creato uno schema di validazione per ogni documento BSON, tramite il quale era possibile eseguire un controllo di validità a tempo di inserimento e di modifica.
In un secondo momento si è presentato un problema relativo a questo aspetto. Durante un inserimento, se i dati inseriti non soddisfano i controlli del file di validazione, questi vengono automaticamente scartati senza dare possibilità di ritentare l’inserimento.
Si è deciso quindi di creare dei metodi che prendessero il posto del sistema di controllo. In pratica, ogni volta che vengono inseriti o modificati dei dati, questi verranno controllati tramite metodi costruiti sulla struttura dei file di validazione. In questo modo i documenti saranno creati correttamente evitando il problema di inserimenti con errori.

3.4.15. Riduzione spazio attributi

L'ultima operazione effettuata in termini di progettazione è stata quella di cercare di ridurre al minimo lo spazio dei nomi degli attributi in modo da velocizzare sotto tutti i punti di vista il recupero dei dati. Essendo una ricerca scientifica, non è stato possibile ridurre più di tanto i vari nomi, ma dove è stato possibile sono state utilizzate delle abbreviazioni.

Un'opzione possibile sarebbe stata quella di utilizzare all'interno del database acronimi o abbreviazioni memorizzando le traduzioni all'interno di un file. Questa soluzione è stata scartata per due semplici motivi: Il primo problema sarebbe stato l'incremento dello spazio occupato, il secondo problema sarebbe stato il processo di conversione, il quale avrebbe alleggerito il lavoro del DBMS per il recupero dei dati ma avrebbe rallentato il sistema complessivo a causa delle operazioni che il back end avrebbe dovuto fare per sostituire i termini.

3.5. Flask

Si passa ora alla tecnologia di back end utilizzata, facendo una panoramica sui motivi che hanno portato al suo utilizzo e su aspetti progettuali fondamentali della Web Application.

3.5.1. Motivi della scelta

La scelta di utilizzare Flask come framework è stata dettata dalle scelte progettuali di contorno. Una volta deciso il tipo database ed aver schematizzato tutti i passaggi da effettuare per costruire l'applicazione finale, è stato il momento di scegliere la tecnologia per la gestione della parte di back end.
Flask è risultato adatto al compito che è stato svolto per diversi motivi:

- Flask è ad oggi la tecnologia più consigliata (scritta in linguaggio Python) per essere affiancata ad un database basato sui documenti come MongoDB.

- Un aspetto fondamentale è che Flask trae vantaggio dalla semplicità di configurazione, non lasciando al tempo stesso nessun aspetto fondamentale escluso.

- La presenza massiccia di librerie e strumenti utilizzabili con un semplice "import" è stato un altro punto a favore di questa tecnologia.

Non è stato necessario sapere a priori durante la fase di progettazione tutte le librerie necessarie poiché la natura di Flask permette di importarle al bisogno garantendo un sistema molto leggero e veloce di default.

3.5.2. Struttura di Flask

Come per ogni progetto è opportuno tenere il codice organizzato. Seguendo le linee guida della documentazione di Flask e creando file secondo uno schema logico si è arrivati alla struttura finale, la quale viene rappresentata all'interno della figura 3.20.

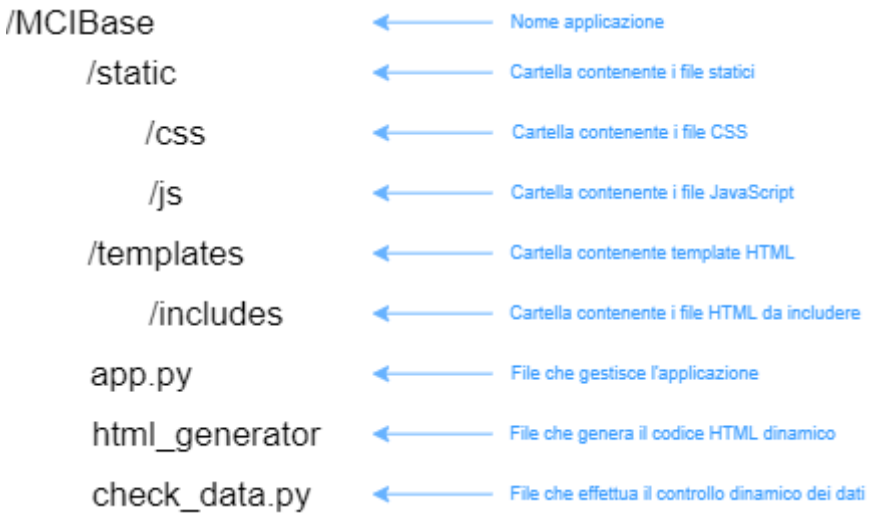


Figura 3.20: Progettazione scheletro Flask.

3.5.3. Riprogettazione dell'utilizzo dei file di validazione

I file di validazione BSON, come anticipato nel paragrafo 3.4.4, una volta progettati, sono stati utilizzati attraverso metodologie differenti:

- La prima è stata quella di creare metodi appositi che prendendo in input i campi dei file BSON sono in grado di eseguire un controllo sui dati prima di inviare questi ultimi al database. Grazie a questa tecnica, nel caso i documenti rispettino i vincoli vengono salvati all'interno del database risultando privi di errori logici. Nel caso invece di errori durante la fase di inserimento, questi non passando i vari controlli, verranno ricaricati all'interno del form eliminando soltanto i campi che non soddisfano i range o strutture imposte, permettendo all'utente di correggere soltanto i campi errati prima di ritentare l'inserimento.

- La seconda è stata quella di creare metodi che permettessero di creare dinamicamente i form di inserimento, di visualizzazione e di download.
  Questo utilizzo è stato pensato in seguito alle richieste del gruppo di ricercatori, i quali hanno espresso il desiderio di poter inserire in futuro nuovi campi all'interno dei diversi esami.
  Grazie a questa tecnica, per inserire un nuovo attributo basterà andare all'interno del file BSON relativo all'esame ed inserirlo seguendo la logica dei file di validazione, la quale è semplice ed intuitiva.
  Al caricamento successivo della pagina, sarà presente il nuovo campo all'interno del form.

4. Implementazione

4.1. Introduzione

In questo capitolo viene mostrata l'implementazione delle varie tecnologie adottate.
Viene mostrata la struttura finale del database e ne vengono illustrati tutti gli aspetti di contorno. In un secondo momento si passa a quella che è l'implementazione del micro-framework Flask, per poi concludere con una panoramica su quello che è il front-end.

4.2. Implementazione del database tramite MongoDB

4.2.1. Installazione di MongoDB

Per quanto riguarda MongoDB, si è scelto di utilizzare l'ultima versione stabile rilasciata, cioè la 4.2.0.
Per l'installazione è stato fatto riferimento alla guida ufficiale, tramite la quale in pochi minuti è possibile avere tutti gli strumenti necessari pronti all'utilizzo.
L'installazione dalla versione 4.0 permette di installare MongoDB come servizio di windows.
Un servizio di windows è un programma eseguibile che svolge compiti specifici senza richiedere l'intervento dell'utente.
In questo modo non vi è la necessità di avviare l'istanza di MongoDB manualmente ogni volta che lo si vuole utilizzare.
L'installazione fornisce anche la GUI (Graphical User Interface) ufficiale per MongoDB chiamata MongoDB Compass. Questo insieme di componenti grafici permette di visualizzare dati, lanciare query e più in generale, come si può vedere all'interno della figura 4.1 fornisce una panoramica generale sul database e su tutto quello da esso contenuto.

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size
blood	6	763.2 B	4.6 KB	1	36.9 KB
clinics	9	443.0 B	4.0 KB	1	36.9 KB
csf	6	381.3 B	2.3 KB	1	36.9 KB
demographics	8	205.4 B	1.6 KB	1	36.9 KB
epic	5	6.9 KB	34.7 KB	1	36.9 KB
fs.chunks	16	5.7 KB	90.7 KB	2	73.7 KB
fs.files	16	177.3 B	2.8 KB	2	73.7 KB
ls	4	3.5 KB	13.8 KB	1	36.9 KB
mri0	4	269.0 B	1.1 KB	1	36.9 KB
mri1	3	308.0 B	924.0 B	1	36.9 KB
nps0	6	3.0 KB	17.8 KB	1	36.9 KB
nps1	4	2.5 KB	10.1 KB	1	36.9 KB
patient	9	153.0 B	1.4 KB	1	36.9 KB
pet	5	223.6 B	1.1 KB	1	36.9 KB
users	1	109.0 B	109.0 B	1	20.5 KB

Figura 4.1: Visualizzazione schermata MongoDB compas per il database MCIBase.

4.2.2. Creazione collezioni

In un primo momento sono state create tutte le collezioni pensate durante la fase di progettazione tramite il comando: "*db.createCollection(nomeCollezione)*" direttamente da terminale.

In un secondo momento, è risultato utile scrivere un paio di righe di codice, le quali, ogni volta che viene riavviato il server, eseguono un controllo sulla lista delle collezioni presenti all'interno del database.

```
exams = ['nps0', 'nps1', 'epic', 'ls', 'blood', 'csf', 'pet', 'mri0', 'mri1', 'clinics', 'demographics']

extra_collections = ['patient','users']
list_of_existing_collections = mongo.db.list_collection_names()
for e in exams+extra_collections:
    if not e in list_of_existing_collections:
        mongo.db.create_collection(e)
```

Figura 4.2: Righe di codice per la creazione dinamica delle collezioni.

Il frammento di codice che si può vedere nella figura 4.2 controlla se le collezioni all'interno della lista "*exams*" (lista degli esami) e quelle presenti all'interno della lista "*extra_collections*" (altre collezioni) sono tutte già presenti all'interno del database. Nel caso in cui una o più collezioni non esistano, tramite il comando "*mongo.db.create_collection(nomeCollection)"* verranno create. In questo modo, quando ci sarà la necessità di creare un esame o una collezione utile a qualsiasi altro scopo, basterà inserirla all'interno di una delle due liste.

Funzionamento frammento di codice (figura 4.1)

"*exams*" e "*extra_collections*" sono due liste di stringhe contenenti i nomi delle collezioni.
Tramite il metodo "*mongo.db.list_collection_name()*" vengono presi dal database tutti i nomi delle collezioni e vengono memorizzati all'interno della variabile "*list_of_existing_collections*".
Tramite il for, vengono scorsi gli elementi di entrambe le liste e per ognuno viene eseguito un controllo. Nel caso l'esame preso in considerazione ("*e"*) non si trovi all'interno della lista "*list_of_existing_collections*", si procede alla creazione della collezione per tale esame.

4.2.3. File di validazione

La costruzione dei file di validazione è stata basata sui dati forniti dal gruppo di ricerca.
All'interno della figura 4.3 è possibile vedere un esempio di come i dati ci sono stati recapitati.
Il caso preso in considerazione tratta i dati demografici.

	A	B	C	D	E	F	G	H
1	**Codice**		**Range**					
2	M002		M001-M200 oppure R001-R200					
3								
4								
5								
6								
7								
8								
9								
10	**Data di nascita**							
11	5/1/1965		n.d.					
12								
13								
14	**Scolarità**							
15	13		0-30					
16								
17	**Sesso**							
18	F o M			*Servirebbe una tendina dove selezionare*				
19								
20	**Manualita`**			*scelta a 3*				
21	Destrimane, mancino, ambidestro							
22								
23	Altezza (m)		1.00-2.50					
24	Peso(kg)		30-150					
25	BMI	calcolabile come peso/(altezza x altezza)						
26								
27								
28	**Caregiver o study partner**							
29	compagno/marito							
30	compagna/moglie							
31	figlio							
32	figlia							
33	amico/a							
34	altro	campo aperto						

Figura 4.3: Dati demografici da trattare.

Una volta raccolti tutti i dati da dover trattare, sono stati costruiti i file di validazione, uno per ogni collection. Nella figura 4.4 è possibile visualizzare il risultato per quanto riguarda la parte demografica.

```
1   {
2       "$jsonSchema": {
3           "bsonType": "object",
4           "title": "Demographics data",
5           "required": [
6             "patient_code",
7             "doc_code"
8           ],
9           "properties": {
10              "patient_code":{
11                "bsonType": "string"
12              },
13              "doc_code":{
14                "bsonType": "string"
15              },
16              "data_di_nascita": {
17                  "bsonType": "date"
18              },
19              "scolarità": {
20                  "bsonType": "number",
21                  "maximum": 30,
22                  "minimum": 0
23              },
24              "sesso": {
25                  "bsonType": "string",
26                  "enum": [
27                      "F",
28                      "M"
29                  ]
30              },
31              "manualita'": {
32                  "bsonType": "string",
33                  "enum": [
34                      "Destrimane",
35                      "Mancino",
36                      "Ambidestro"
37                  ]
38              },
39              "altezza(m)": {
40                  "bsonType": "number",
41                  "minimum": 1.00,
42                  "maximum": 2.50
43              },
44              "peso(kg)": {
45                  "bsonType": "number",
46                  "maximum": 150,
47                  "minimum": 30
48              },
49              "BMI": {
50                  "bsonType": "number"
51              },

52              "Caregiver_o_study_partner": {
53                  "bsonType": "object",
54                  "anyOf": [
55                      {
56                          "required": [
57                              "list"
58                          ]
59                      },
60                      {
61                          "required": [
62                              "altro"
63                          ]
64                      }
65                  ],
66                  "properties": {
67                      "list": {
68                          "bsonType": "array",
69                          "items": {
70                              "bsonType": "string",
71                              "enum": [
72                                  "compagno/marito",
73                                  "compagna/moglie",
74                                  "figlio",
75                                  "figlia",
76                                  "amico/a"
77                              ]
78                          }
79                      },
80                      "altro": {
81                          "bsonType": "string"
82                      }
83                  }
84              }
85          }
86      }
87  }
88
```

Figura 4.4: Schema di validazione dati demografici.

L’operatore “$jsonSchema” fa match con documenti che soddisfano le specifiche dello schema JSON.
Questo operatore deve avere la seguente sintassi: “*{ $jsonSchema: <JSON Schema object> }*”.
Il “*JSON Schema object*” deve seguire la sintassi: “*{ <keyword1>: <value1>, ... }*”.

Ogni documento ha le proprie caratteristiche che variano in base al tipo di dato:

- La parola chiave “*bsonType*” permette di definire quale tipo di dato il sistema dovrà aspettarsi. I tipi di dato che sono stati utilizzati sono: object, string, date, array, number, boolean.

- Tramite il bsonType “*Object*” è possibile creare documenti annidati. Un documento annidato sarà all’interno della chiave “*properties*” del documento padre, come è possibile vedere alla riga di codice 9 dell’immagine 4.4.

- La chiave “*title*” permette semplicemente di specificare un titolo per un bsonType “*object*” nel caso ci sia il bisogno.

- La chiave “*required*” permette di specificare quali dati del documento sono obbligatori.

- Le chiavi “*minimum*” e “*maximum*” permettono di definire il range di valori che il bsonType “*number*” deve rispettare.

- La chiave “*enum*” permette di specificare una lista di elementi. Di questi elementi, in fase di inserimento, ne dovrà essere selezionato uno tra quelli presenti.

- La chiave “*anyOf*” viene utilizzata per specificare che è richiesta una delle *n* opzioni contenute.

4.2.4. Unione di MongoDB a Flask

Per permettere il collegamento tra MongoDB e Flask si è reso fondamentale l'uso della libreria "pymongo", raccomandata per lavorare con il database in questione tramite il linguaggio Python.

L'installazione è resa immediata dal comando: "*python -m pip install pymongo*".

Le righe di codice essenziali per poter iniziare a lavorare con MongoDB da Flask sono essenzialmente tre.
La prima riguarda l'importazione della libreria all'interno del file principale di Flask:

"from flask_pymongo import PyMongo"

La seconda e terza riga di codice servono rispettivamente per configurare l'indirizzo del database, il quale in locale utilizza la porta 27017 e per istanziare un oggetto di tipo PyMongo.

"app.config["MONGO_URI"] = "mongodb://localhost:27017/MCIBase"
"mongo = PyMongo(app)"

Dalla riga successiva, per tutte le operazioni da effettuare sul database, viene fatto riferimento alla variabile "*mongo*".

4.2.5. Recupero dei dati

Di seguito vengono riportate alcune interrogazioni effettuate per recuperare i dati all'interno dell'applicazione, con lo scopo di illustrare l'implementazione delle query.

- *pazienti = mongo.db.patient.find({})*: assegna alla variabile "*pazienti"* la lista dei pazienti presenti all'interno della collezione.
- *mongo.db[path].find_one({'patient_code':code})*: tramite questa query è possible recuperare il singolo documento del paziente specificato (*patient_code*) nella collezione di interesse (*path*).
- *mongo.db[path].delete_one({'patient_code':code})*: questa query è utilizzata per eliminare il documento di un esame (*path*) di un utente (*patient_code*)
- *mongo.db['patient'].update_one({'patient_code':code},{"$set": {path:False} })*: una volta eliminato il documento relativo ad un esame di un paziente, si procede ad effettuare l'update

del campo booleano relativo alla presenza di quell'esame (campo booleano introdotto all'interno del capitolo sulla progettazione paragrafo 3.4.2).

4.3. Creazione del back-end tramite Flask

4.3.1. Installazione di Flask

Per quanto riguarda Flask, si è scelto di utilizzare l'ultima versione stabile rilasciata, cioè la 1.1.1.
L'installazione è stata effettuata seguendo la guida presente all'interno del sito ufficiale, la quale ha permesso di installare questo micro-framework in pochi minuti.
Di default, Flask utilizza la porta 5000 per connettersi al server.
La versione di Python utilizzata per questo progetto è la 3.5.

4.3.2. Creazione applicazione

Il primo file creato "*app.py*" viene utilizzato per quella che è la logica dell'applicazione. Può essere considerato il gestore o il "main" dell'applicazione.
La prima cosa fatta, è stata quella di importare la classe Flask. Un'istanza di questa classe sarà l'applicazione WSGI.
Oltre a questa classe, ne vengono importate molte altre, illustrate all'interno del paragrafo 4.3.3.

4.3.3. Import

Viene riportata di seguito una lista degli import effettuati, seguiti da una breve spiegazione del loro utilizzo:

- from flask_pymongo import PyMongo: classe per l'interfacciamento al database
- import render_templare: Permette di effettuare il render dei vari template.
- import request: utilizzata per costruire richieste HTTP.
- import redirect: utilizzata per effettuare redirect.
- import session: utilizzata per implementare le sessioni per gli utenti.
- import send_from_directory: utilizzato per permettere l'upload di file.
- import send_file: utilizzato per poter inviare un file da una data directory
- import url_for: utilizzato per costruire dinamicamente un URL specifico.

- from flask_wtf.csrf import CSRFProtect, CSRFError: Classi utilizzate per l'abilitazione della protezione CSRF e per la personalizzazione dell'errore CSRF.
- import datetime: modulo utilizzato per importare il tipo di dato "data".
- import bcrypt: modulo utilizzato per importare la funzione di hashing delle password
- import json: import necessario per poter lavorare con dati JSON.
- import csv: import necessario per poter leggere e scrivere file CSV.

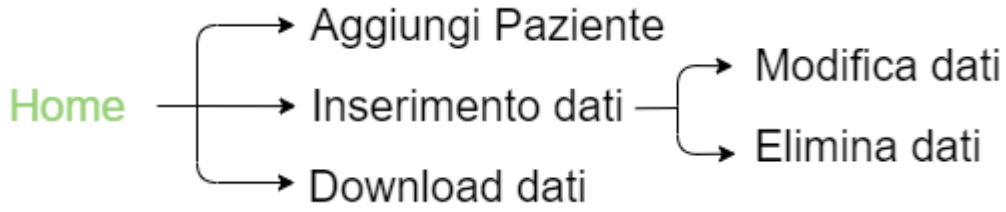
Altri import di file interni utilizzati sono:

- from html_generator import get_infos_from_json, combine
- from check_data import convert_data,search_type_and_other_in_json, convert_bsontype
- from html_view import create_html_view

4.3.4. Workflow e metodi fondamentali

In questo capitolo, dato l'ingombro del codice, è stato ideato un semplice schema, che permetterà di capire la divisione dei vari metodi e soprattutto in quale parte del progetto sono utilizzati.

4.3.5. Funzione per la gestione della pagina principale

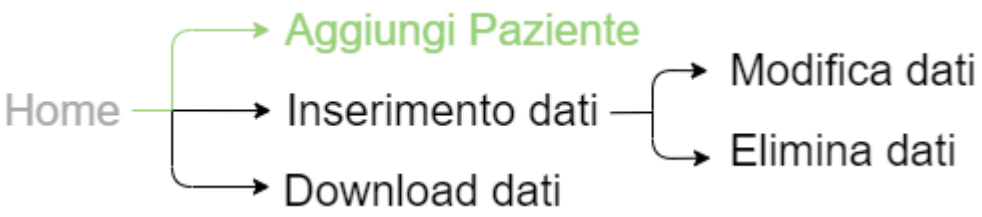


La funzione *index()* (figura 4.5) controlla quale ordinamento è stato scelto dall'utente ed effettua di conseguenza una query per recuperare la lista di pazienti ordinati.
Questa funzione Inoltre, nel caso sia presente, mostra nella home page un messaggio di errore o di conferma.

```python
@app.route('/',  methods=['GET','POST'])
def index():

    original_order = ''
    if request.method == 'POST' and 'ordinamento' in request.form.keys():
        order = request.form['ordinamento'].lower()
        original_order = request.form['ordinamento'].lower()
        order = order.replace(' t0','0')
        order = order.replace(' t1','1')
        order = order.replace(' ','_')


        if order in ['nps','mri','codice(m/r)']:
            if order == 'nps':
                pazienti = mongo.db.patient.find({}).sort([("nps0", pymongo.DESCENDING), ("nps1", pymongo.DESCENDING)])
            elif order == 'mri':
                pazienti = mongo.db.patient.find({}).sort([("mri0", pymongo.DESCENDING), ("mri1", pymongo.DESCENDING)])
            else:
                pazienti = mongo.db.patient.find({}).sort("patient_code", pymongo.ASCENDING)

        else:
            pazienti = mongo.db.patient.find({}).sort(order, pymongo.DESCENDING)
    else:
        pazienti = mongo.db.patient.find({}).sort("data_consenso", pymongo.DESCENDING)

    #se c'è un messaggio da mostrare
    if 'msg_code' in request.args.keys() and not request.args['msg_code'] == None:
        msg_code = request.args['msg_code']
        message = request.args['message']

    else:
        msg_code = None
        message = None

    return render_template('index.html', pazienti=pazienti, msg_code = msg_code, message = message, order = original_order )
```

Figura 4.5: Funzione per la gestione della pagina principale.

4.3.6. Funzione per l'aggiunta di un paziente



La funzione *new_patient()* (figura 4.6) viene eseguita quando un Ricercatore inserisce un nuovo paziente.

Viene per prima cosa resa la prima lettera maiuscola e si controlla che il campo sia stato riempito.

Si procede a controllare la lunghezza, la prima lettera e che gli ultimi tre caratteri siamo numeri e non lettere.

L'ultimo controllo effettuato, accerta che il paziente che si sta cercando di inserire, non esista già.

Nel caso tutti i passaggi vengano passati, il nuovo paziente verrà inserito e sarà mostrato un messaggio di conferma.

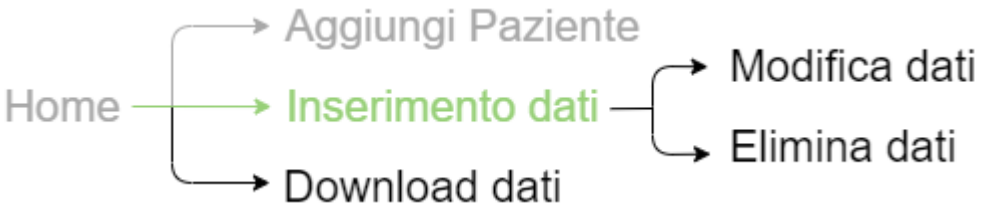
Nel caso contrario in cui uno dei controlli restituisce esito negativo, viene mostrato un messaggio di

errore.

```
@app.route('/new_patient',  methods=['GET','POST'])
def new_patient():

    if request.method == 'POST':
        code = request.form['code'].upper()

        #controllo che non sia nullo
        if not code.replace(' ','') == '':

            #controllo lunghezza
            if len(code) == 4:

                #controllo che la prima lettera sia effettivamente una r o una m
                if str(code[0]) in ['M','R']:
                    for i in code[1:]:
                        # controllo che le altre 3 lettere siano dei numeri, in caso contrario restituisce errore
                        if not i.isdigit():
                            return redirect(url_for('index',msg_code='0', message="Codice inserito non valido."))

                    # controllo che il paziente non esista già
                    if mongo.db.patient.find({'patient_code':code}).count() == 0:
                        doc = {}
                        doc['patient_code'] = code
                        doc['data_consenso'] = datetime.datetime.now()
                        for e in exams:
                            doc[e] = False
                        mongo.db.patient.insert(doc)

                        return redirect(url_for('index',msg_code='1', message="Codice inserito."))

                    else:
                        return redirect(url_for('index',msg_code='0', message="Codice inserito già esistente."))

                else:
                    return redirect(url_for('index',msg_code='0', message="Codice inserito non valido."))
            else:
                return redirect(url_for('index',msg_code='0', message="Codice inserito non valido."))
        else:
            return redirect(url_for('index',msg_code='0', message="Codice inserito non valido."))

    return redirect(url_for('index',msg_code='0', message="Non inserito, qualcosa è andato storto."))
```

Figura 4.6: Funzione per l'aggiunta di un paziente.

4.3.7. Funzione per l'inserimento dei dati



La funzione *check_data(),* viene utilizzata per l'inserimento di un file all'interno del database.
Viene creato per prima cosa un dizionario temporaneo chiamato *document*, il quale verrà salvato alla fine del procedimento sul database come documento.
Qualsiasi documento verrà inserito, verranno memorizzati sempre il codice del paziente e il codice del dottore.
Tramite la funzione *convert_data()*, vengono presi i dati presenti nel form di inserimento (contenuti nella variabile *"result")* e formattati dentro la variabile "*document*".
Successivamente viene controllata l'esistenza del file (stesso paziente e stessa collection). Il caso di corrispondenza, indica che ci si trova nel caso di modifica del documento, di conseguenza, il documento vecchio viene eliminato dal database per poi procedere con l'inserimento del nuovo.
Dopodiché segue l'aggiornamento della variabile booleana del paziente, che attesta la presenza di tale esame.
Infine, viene generato il codice HTML per poter mostrare la pagina dei dati inseriti al ricercatore.

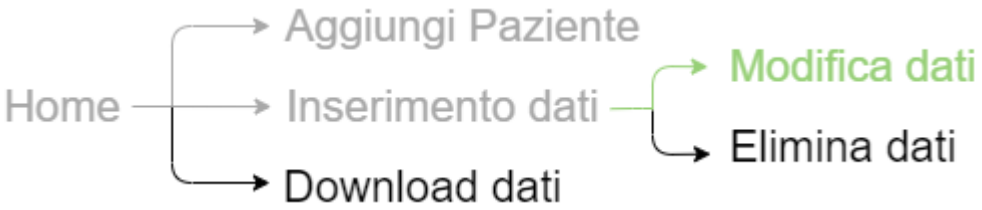
```python
@app.route('/check_data/<path:path>', methods=['GET','POST'])
def check_data(path = None):
    # prendo codice associato ai dati inseriti tramite form
    doc_code = 'AAAAA'

    # controllo che il link sia corretto
    if request.method == 'POST' and path in exams:
        result = request.form
        files = request.files

        #codice paziente associato all'esame in questione
        code = request.form['code']
        # doc_code = request.form['doc_code']

        #caso nuovo esame
        #documento che verra inserito nel db
        document = {}
        document['patient_code'] = code
        document['doc_code'] = doc_code

        #controllo sesso, solo nel caso di nps
        s = None
        if 'nps' in path:
            s = mongo.db['demographics'].find_one({'patient_code': code})['sesso']

        files_dict = {}
        #controllo presenza di file
        for f in files:
            file = files[f]
            n = int(f[-1])
            f = f[0:len(f)-1]

            if n == 0 and f+'$$changed_file' in result:
                rcf = result.getlist(f+'$$changed_file')
                doc = mongo.db[path].find_one({'patient_code':code})
                for k in f.split('$$'):
                    doc = doc.get(k)
                tmp_doc = []
                for i in range(len(rcf)):
                    if rcf[i] in doc:
                        tmp_doc.append(rcf[i])
                    else:
                        #elimino il file non confermato, ovvero rcf[i]
                        file_id = mongo.db.fs.files.find_one({"filename":rcf[i]})['_id']
                        gridfs.GridFS(mongo).delete(file_id)
                        mongo.db['fs.files'].delete_one({"file_name":rcf[i]})

                        pass
                doc = tmp_doc

                if f in files_dict:
                    for d in doc:
                        files_dict[f].append(d)
                else:
                    files_dict[f] = doc
```

Figura 4.7: Funzione per l'inserimento dei dati (parte 1).

```
        #controllo file vuoto
        if not file.filename == '':

            #elaborazione nuovo nome
            namefile = ''
            tmp_name = code + '$$' + f
            tmp_name = tmp_name.replace('Flag$$','')

            for sf in tmp_name.split('$$')[0:len(tmp_name.split('$$'))-1]:
                if namefile == '':
                    namefile += sf.upper()
                else:
                    namefile += '_'+sf

            #controllo duplicazione nome
            # namefile += '_n'+n+'.'+file.filename.split('.')[-1]

            while True:
                if f in files_dict.keys():
                    if namefile+'_n'+str(n)+'.'+file.filename.split('.')[-1] in files_dict[f]:
                        n+=1
                    else:
                        break
                else:
                    break

            namefile += '_n'+str(n)+'.'+file.filename.split('.')[-1]

            #salvataggio file in db
            mongo.save_file(namefile, file)

            #immissione file in documento mongo per esame
            if f in files_dict:
                files_dict[f].append(namefile)
            else:
                files_dict[f] = [namefile]

        if f in files_dict.keys() and files_dict[f] == []:
            del files_dict[f]

    #conversione dati form in dizionario da inserire nel db -> document
    convert_data(result, document, path, s, extra_dict = files_dict)
    #se siamo in modifica -> elimino -> reinserisco
    if mongo.db['patient'].find_one({'patient_code':code})[path]:
        mongo.db[path].delete_one({'patient_code':code})

    #inserimento document nella collection corrispndete[tmp = nome collection in cui inserire]
    mongo.db[path].insert(document,check_keys=False)
    #aggiorno patient e esami fatti
    patient = mongo.db['patient'].find_one({'patient_code':code})
    patient[path] = True
    if path == 'clinics':
        patient['data_consenso'] = datetime.datetime.strptime(result['Data_di_firma_consenso'], '%Y-%m-%d')
    mongo.db['patient'].save(patient)

    #genero html per mostrare i dati inseriti
    skeleton = []
    flat_dictionary = {}
    create_html_view(skeleton, document, flat_dictionary)

    return render_template('view.html', code=code, path=path, doc = skeleton, flat_dictionary=json.dumps\
        (flat_dictionary, default=str),msg_code='1', message='Esame compilato.')

return redirect(url_for('index',msg_code='2', message='Errore di comunicazione.'))
```

Figura 4.8: Funzione per l'inserimento dei dati (parte 2).

4.3.7. Funzione per la modifica dei dati



Tramite questa la funzione modify() (figura 4.9) , vengono ricaricati i dati all'interno del form di inserimento.

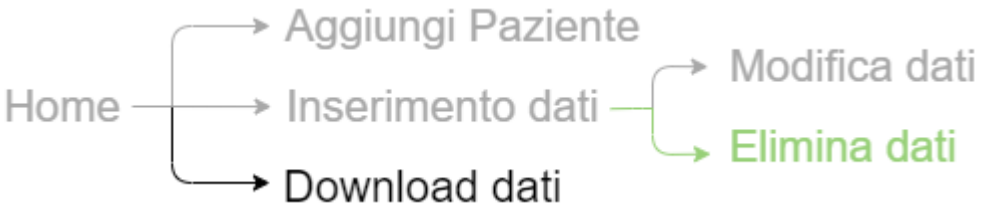
Dalla pagina di visualizzazione vengono presi i dati contenuti nel documento precedentemente inserito. Tramite la funzione combine, successivamente, va ad inserire i dati dentro il form creato nello stesso modo di quello creato per l'inserimento tramite la riga di codice:

"*skeleton = get_infos_from_json(path, code = code, use = "upload")*"

```
@app.route('/modify/<path:path>',  methods=['GET','POST'])
def modify(path = None):
    # controllo che il link sia corretto
    if request.method == 'POST' and path in exams:
        #codice paziente associato all'esame in questione
        code = request.form['code']

        flat_dictionary = request.args.get('flat_dictionary')

        flat_dictionary = json.loads(flat_dictionary)

        # controllo che il link sia corretto
        if path in exams:

            skeleton = get_infos_from_json(path, code = code, use = "upload")
            combine(skeleton, flat_dictionary)

            return render_template('upload.html', skeleton = skeleton, path=path, code = code, msg_code = None)

    return render_template('404.html', msg_code = None)
```

Figura 4.9: Funzione per la modifica dei dati di un esame.

4.3.8. Funzione per l'eliminazione dei dati

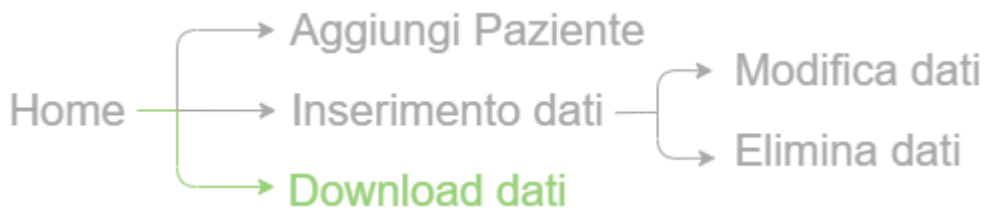


La funzione *delete()* (figura 4.9) permette di eliminare un documento precedentemente inserito. Per prima cosa attraverso la riga di codice "*mongo.db[path].delete_one({'patient_code':code})*" viene eliminato il documento in questione, successivamente, tramite "*mongo.db['patient'].update_one({'patient_code':code},{"$set": {path:False} })*" viene settato a "false" la variabile booleana che attesta la presenza o l'assenza dell'esame.

```
@app.route('/delete',  methods=['GET','POST'])
def delete():

    if request.method == 'POST':
        path = request.args.get('path')
        code = request.args.get('code')

        #eliminazione
        if mongo.db[path].delete_one({'patient_code':code}):
            mongo.db['patient'].update_one({'patient_code':code},{"$set": {path:False} })

        return redirect(url_for('index',msg_code='1', message= path.capitalize()+" di "+code+" eliminato."))
    else:
        return redirect(url_for('index',msg_code='0', message="Errore, nulla è stato eliminato."))
```

Figura 4.10: Funzione per l'eliminazione dei dati di un esame.

4.3.9. Funzione per il download dei dati



La funzione *download()* (figura 4.11), se richiamata senza argomento, rimanda alla pagina "*download.html"*.

Se richiamata dalla funzione *select* invece, essa genera un file CSV.

Più nello specifico, vengono creati due dizionari, "*riga"* e "*dizionario"* . "*riga"* contiene una lista di parole chiave, cioè i nomi degli attributi (titolo delle colonne) mentre "*dizionario*" conterrà i dati. Come prima colonna si avrà il codice del paziente mentre in tutte quelle successive saranno contenuti i vari dati .

Una volta creati i due dizionari, viene aperto un file CSV in scrittura. Successivamente tramite la funzione *send_file()* verrà aperta all'utente una finestra di dialogo tramite la quale potrà scegliere dove salvare tale file.

```python
@app.route('/download/<path:path>', methods=['POST','GET'])
def download(path = None):
    if request.method == 'POST':
        dati_richiesti = request.form.getlist('download_list')

        riga = {}
        dizionario = {}

        # #lista di pazienti e dizionario chiave(codice paziente) valore(dati esame path del relativo codice->paziente)
        pazienti = mongo.db.patient.find({})
        for p in pazienti:
            code = p['patient_code']
            dizionario[code] = mongo.db[path].find_one({'patient_code':code})

        if path[-1].isdigit():
            path=path[:len(path)-1]
        with open('Json_file/'+path+'.json', 'r', encoding='utf-8') as f:     #apro il file json specificato in path
            jfile = json.load(f)
            jfile = jfile['$jsonSchema']['properties']

        aggiunta = []
        #per ogni chiave della lista
        for l in dati_richiesti:
            #salvo bsonType
            bsonType = []
            search_type_and_other_in_json(path, l.split('$$'), bsonType)

            #controllo ogni paziente
            for p in dizionario:
                if not p in riga:
                    riga[p] = {}
                #controllo tutte le componenti della chiave
                riga[p][l] = dizionario[p]

                for s in l.split('$$'):
                    #controllo che ci sia la chiave parziale
                    if riga[p][l] == None:
                        riga[p][l] = None
                        break
                    else:
                        if s in riga[p][l]:
                            riga[p][l] = riga[p][l][s]
                        else:
                            riga[p][l] = None
                convert_to_csv_list(riga,p,l,bsonType[-1],jfile, dati_richiesti, aggiunta)

        fieldnames = ['patient_code']+dati_richiesti+aggiunta
        with open('CSV_download/'+path.upper()+'_Data.csv', mode='w', newline='') as csv_file:
            writer = csv.DictWriter(csv_file, fieldnames=fieldnames)

            writer.writeheader()

            for p in riga:
                riga[p]['patient_code'] = p
                writer.writerow(riga[p])
        return send_file('CSV_download/'+path.upper()+'_Data.csv', as_attachment=True)
        # return send_from_directory(directory='CSV_download', filename=path.upper()+'_Data.csv')
    else:
        return render_template('download.html', msg_code=None)
```

Figura 4.11: Funzione per il download dei dati di un esame.

4.4. Interfaccia grafica

4.4.1. Home

Come è stato richiesto inizialmente dal gruppo di ricerca, è stata creata una pagina iniziale che permette di visualizzare una lista di pazienti con i relativi esami.

All'interno dell'immagine 4.12 è possibile vedere la struttura della pagina principale con soltanto i pazienti, e nessun esame inserito, come si può notare dalla scala di grigi con cui sono rappresentati gli esami.

Mano a mano che vengono aggiunti i dati di un esame, la "label" relativa ad esso assume una colorazione diversa per rendere evidente l'avvenuto inserimento come mostrato nella figura 4.13.

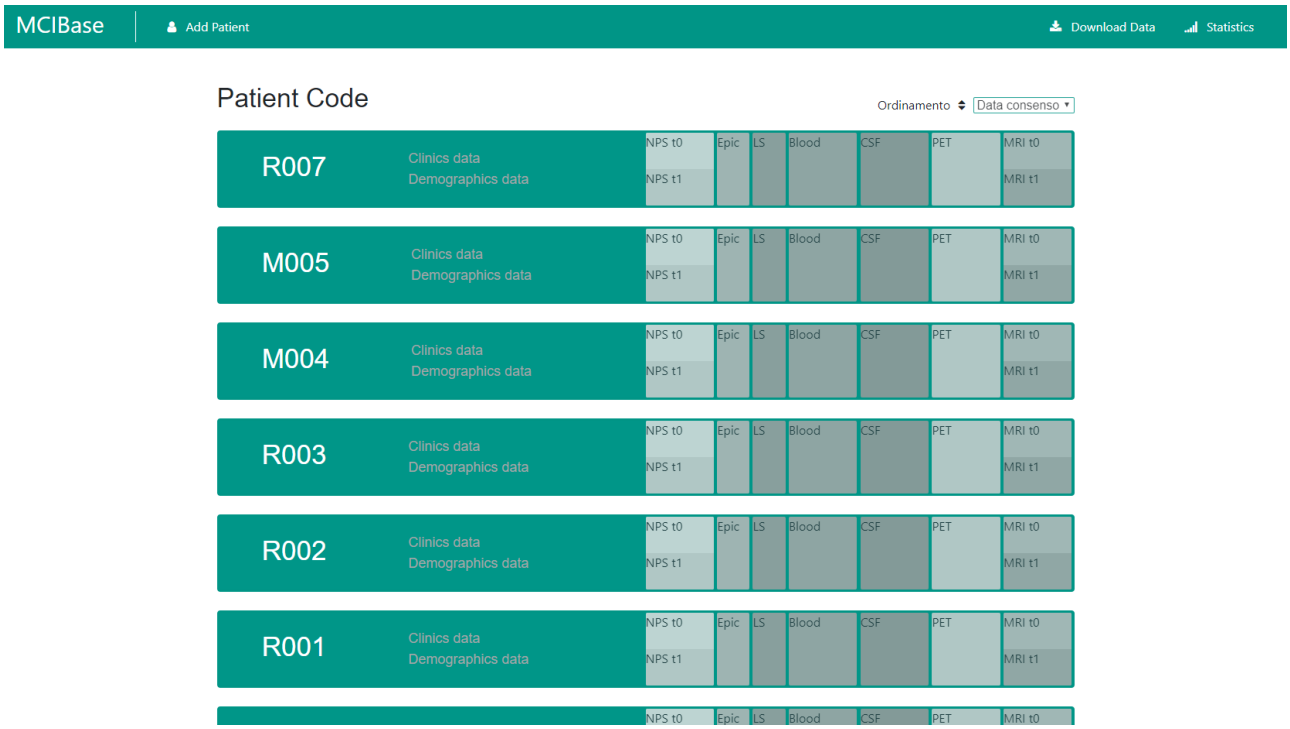


Figura 4.12: Pagina iniziale senza nessun esame inserito.

Figura 4.13: Pagina iniziale con diversi esami inseriti.

In seguito alla richiesta della possibilità di ordinare i pazienti secondo diversi criteri, è stata implementata un "tendina" di tipo "drop-down" in modo da consentire un diverso ordinamento tramite un semplice click.

Si può visualizzare il risultato all'interno dell'immagine 4.14.

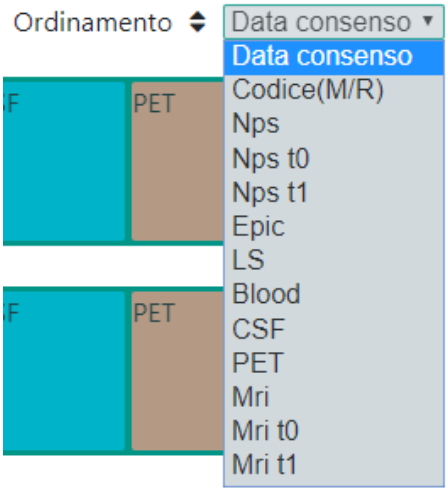


Figura 4.14: Tendina per l'ordinamento dei pazienti.

4.4.2. Aggiunta Paziente

Per quanto riguarda l'aggiunta di un paziente, si è deciso di tenere il più semplice ed intuitiva possibile l'interfaccia relativa.

Premendo il bottone "Add patient" presente all'interno della *navbar* (visibile nella figura 4.13), apparirà un elemento contenente un semplicissimo form di immissione, il quale richiede soltanto il codice del paziente. A questo form, come spiegato nel paragrafo 4.3.4 (Funzione per l'aggiunta di un paziente) sono stati applicati dei controlli per impedire l'inserimento di un'espressione non valida.

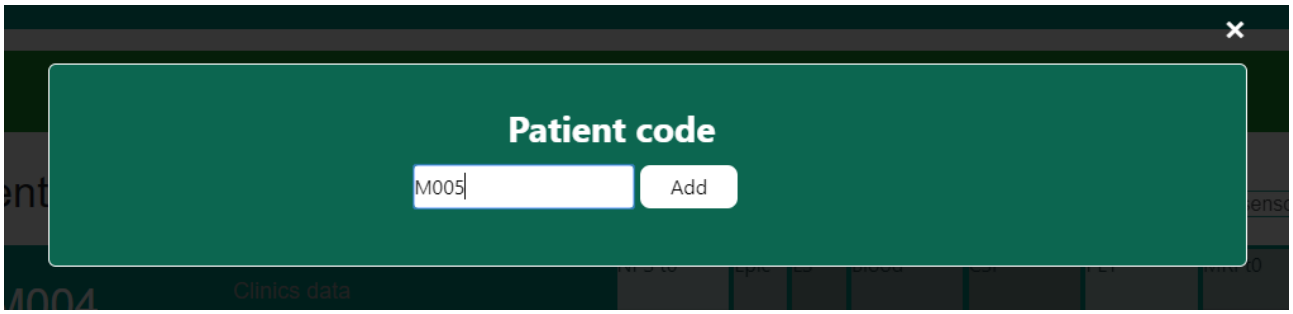


Figura 4.15: Form aggiunta paziente.

4.4.3. Inserimento, modifica ed eliminazione dati

All'interno della figura 4.16 è possibile vedere il form di inserimento dei dati demografici.

I form sono costruiti tutti allo stesso modo partendo dai file di validazione, che, tramite le funzioni presenti all'interno del file *html_generator.py*, vengono presi come input per poter creare dinamicamente il codice html di questi form.

Grazie a questa tecnica, nel caso si decida di aggiungere o eliminare un campo dal form, è sufficiente aggiornare il file JSON relativo all'esame in questione e ricaricare la pagina per apportare le modifiche.

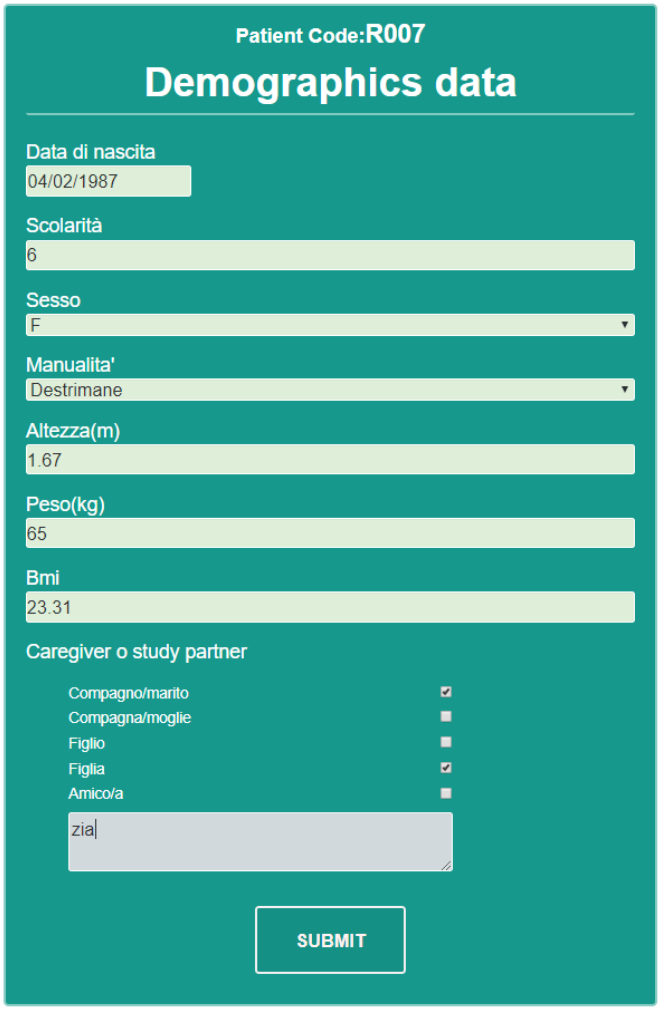


Figura 4.16: Form inserimento dati.

In seguito all'immissione dei dati, nel caso questi soddisfino i requisiti della funzione *check_data*, verranno inseriti all'interno del database e si verrà reindirizzati all'interno di una pagina dove sarà possibile visualizzare come sono stati effettivamente inseriti (figura 4.2) e un messaggio di conferma (visualizzabile all'interno della figura 4.17).

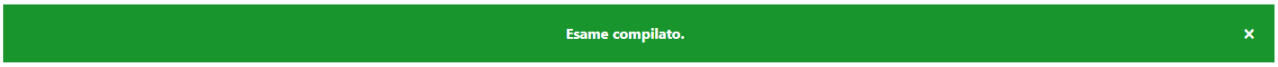


Figura 4.17: Messaggio di conferma.

Nel caso in cui i dati inseriti non passino il controllo, verrà mostrato un messaggio d'errore relativo al problema che ha impedito l'inserimento. (esempio di messaggio d'errore mostrato in figura 4.18).

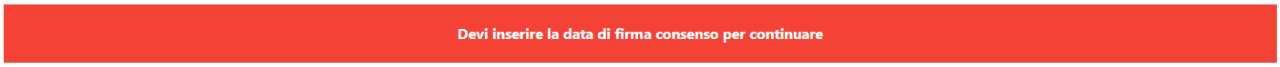


Figura 4.18: Messaggio d'errore.

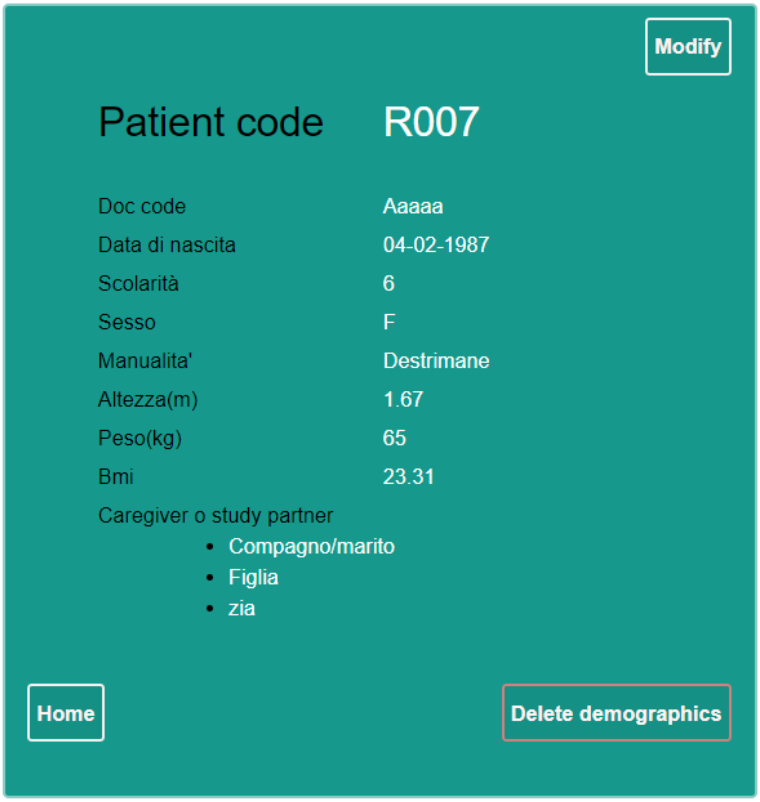


Figura 4.19: Visualizzazione dati inseriti.

Successivamente all'inserimento dei dati, all'interno della pagina che mostra l'avvenuto inserimento (figura 4.19) sarà possibile modificare o eliminare tale documento.

Il documento memorizzato all'interno del database è riportato all'interno della figura 4.20.


```
_id: ObjectId("5e7a4aa9e699c19bd22e7410")
patient_code: "R007"
doc_code: "AAAAA"
data_di_nascita: 1987-02-04T00:00:00.000+00:00
scolarità: 6
sesso: "F"
manualita': "Destrimane"
altezza(m): 1.67
peso(kg): 65
BMI: 23.31
Caregiver_o_study_partner: Object
  list: Array
    0: "compagno/marito"
    1: "figlia"
  altro: "zia"
```

4.4.4. Download dati

Tramite il bottone “Download Data” presente nella *navbar* si viene reindirizzati all’interno della pagina illustrata nella figura 4.20, la quale permette di scegliere le informazioni da scaricare.
Essendo stato espressamente chiesto di poter scaricare i dati divisi per esame, si è deciso di ricreare la struttura degli esami di un singolo paziente, ingrandendola e integrando i dati clinici e demografici al di sotto di quest’ultima.

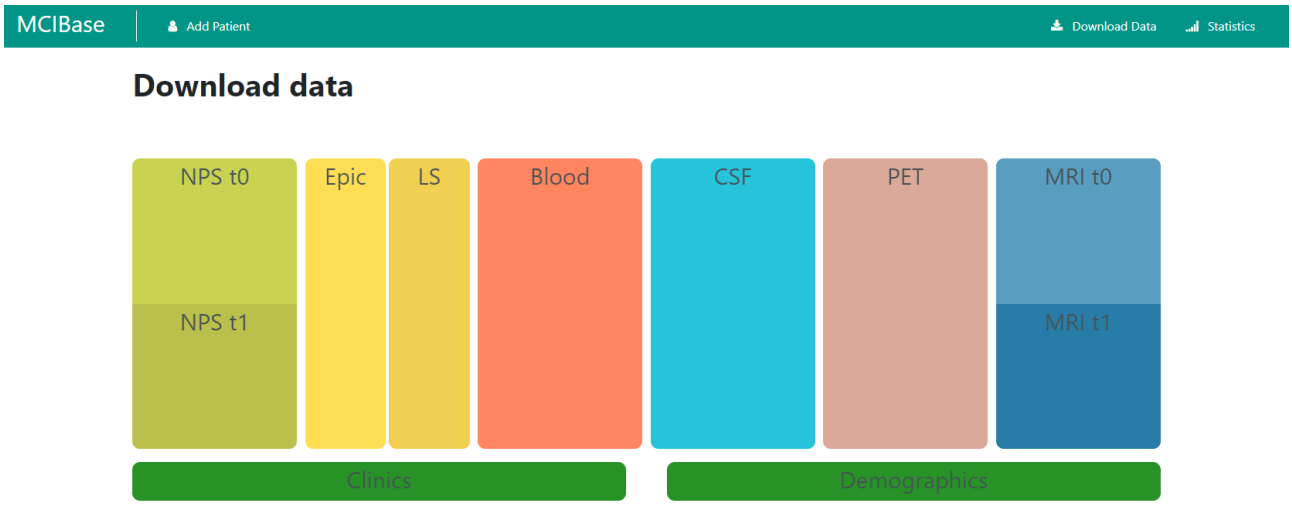


Figura 4.21: Tabella esami download.

Di seguito (figura 4.22) viene riportata la struttura della pagina di download di un singolo esame.
Per rendere più facile selezionare e deselezionare tutte le caselle è stata aggiunta (come proposto in fase di progettazione) una checkbox chiamata “Select all”.
Il ricercatore, grazie a questa interfaccia, è libero di scegliere quali dati scaricare dell’esame in questione.

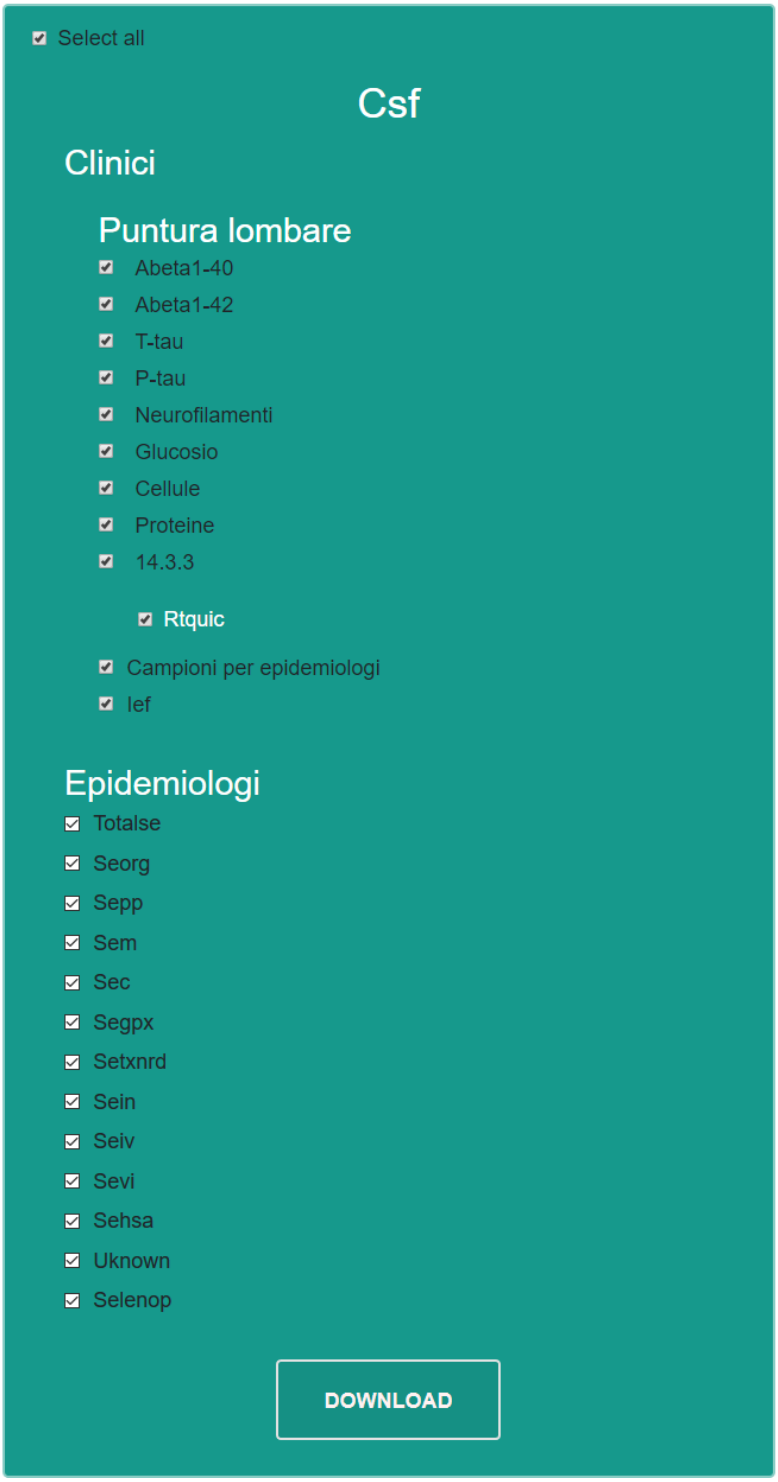


Figura 4.22: Form Download dati.

5. Conclusioni e Sviluppi futuri

L'obiettivo iniziale prefissato è stato raggiunto.
È stato progettato e realizzato un sistema che soddisfa a pieno il compito per il quale è stato ideato.
Grazie a questo sistema, è ora possibile inserire, visualizzare, scaricare e soprattutto condividere con tutti i ricercatori i dati dell'intera ricerca, facilitando operazioni prima lente o impraticabili.
Inoltre, la Web Application creata, essendo stata pensata molto guardando al futuro, è in grado di gestire situazioni molto diverse tra loro.

Il gruppo di ricerca è rimasto molto colpito dalla semplicità delle operazioni, ha apprezzato molto la struttura nel suo insieme e anche tutte le piccole funzionalità. Una volta presentato a loro il progetto, l'overview generale sul lavoro ultimato, ha fatto sì che venissero loro in mente diverse funzionalità aggiuntive che potrebbero essere implementate in futuro.

Alcuni sviluppi futuri che sono stati pensati possono includere:

- Creazione di una piattaforma per l'elaborazione dei dati direttamente online tramite l'utilizzo di librerie Python.
  L'implementazione di una pagina dedicata all'elaborazione online dei dati permetterebbe ai ricercatori di non dover scaricare i dati e importarli all'interno di applicazioni terze, risparmiando loro tempo.
- Creazione di una pagina per ricerche personalizzate (punto collegato concettualmente al precedente).
  Una pagina per le ricerche personalizzate, permetterebbe (qualora ve ne fosse bisogno) di creare maschere dinamiche per il recupero dei dati.
- Creazione di grafici per la visualizzazione dei dati sotto aspetti diversi utilizzando librerie grafiche.
  Una volta identificati i dati statistici per loro più rilevanti, è possibile creare una pagina contenente tutti i diversi grafici necessari utilizzando una libreria grafica (per esempio: Plotly, libreria scritta in Python).

- Espansione del progetto a livello regionale e nazionale.

  Questo punto non è per ora prevedibile, ma essendo una probabilità presa in considerazione, il sistema è stato pensato per poter affrontare al meglio questa espansione.
- Implementare la possibilità di Inserimento di dati da file CSV.

  Potrebbe risultare utile, qualora le varie parti (in questo momento Modena e Reggio Emilia) decidessero di adottare uno schema identico. In questo modo sarebbe possibile creare un'opzione di upload dei dati direttamente da file.
- Creare un collegamento diretto con la biobanca, dove verranno conservati i campioni biologici dei pazienti.

  Questo ultimo punto è da valutare bene per quanto riguarda l'aspetto della privacy dei dati prima di poter essere sviluppato.
- Ricreare quelli che sono i questionari cartacei, all'interno della Web Application in modo da permettere la compilazione direttamente online.

  Questa operazione richiederebbe diverse ore di lavoro per essere implementata ma potrebbe semplificare quello che è il lavoro degli Epidemiologi.
- Possibilità di fornire un accesso ai pazienti coinvolti nello studio che permetta loro di compilare autonomamente i questionati "Epic" ed "LS".

  Ciò permetterebbe agli epidemiologi di delegare la compilazione ai pazienti in modo da poter risparmiare tempo.

Bibliografia

[1] “Web Application - Oracle.” [Online]. Available: https://it.wikipedia.org/wiki/Applicazione_web

[2] “database relazionale - Oracle.” [Online]. Available: https://www.oracle.com/it/database/what-is-a-relational-database/

[3] “dato strutturato - wikipedia.” [Online]. Available: https://it.wikipedia.org/wiki/Dato_semistrutturato

[4] “NoSQL - wikipedia.” [Online]. Available: https://it.wikipedia.org/wiki/NoSQL

[5] “Base di dati a grafo.” [Online]. Available: https://it.wikipedia.org/wiki/Base_di_dati_a_grafo

[6] “Network Database.” [Online]. Available: https://it.wikipedia.org/wiki/Modello_reticolare

[7] “Hierchical Database.” [Online]. Available: https://it.wikipedia.org/wiki/Modello_gerarchico

[8] “Base di dati a oggetti.” [Online]. Available: https://it.wikipedia.org/wiki/Base_di_dati_a_oggetti

[9] “Document Database.” [Online]. Available: https://aws.amazon.com/it/nosql/document/

[10] “Information Retrieval.” [Online]. Available: https://it.wikipedia.org/wiki/Information_retrieval

[11] “Map Reduce.” [Online]. Available: https://it.wikipedia.org/wiki/MapReduce

[12] “SQL-Comparison.” [Online]. Available: https://docs.mongodb.com/manual/reference/sql-comparison/