Università degli Studi di Milano Bicocca

Scuola di Scienze

Dipartimento di Informatica Sistemistica e Comunicazione

Corso di Laurea Triennale in Informatica

Sviluppo e implementazione di un Sistema Decisionale in ambito medico

Relatore:
Federico Cabitza

Correlatore:
Andrea Campagner

Candidata:
Matilde Ghidini
852256

ANNO ACCADEMICO 2022/2023

Ringraziamenti

Dedico questo spazio del mio elaborato a tutte le persone che hanno contribuito alla realizzazione dello stesso, e che mi sono state vicine durante questi anni.

Al prof. Cabitza e al dott. Campagner, per avermi permesso di prendere parte a questo progetto, accompagnandomi nella fase conclusiva del mio percorso accademico. Grazie per la fiducia dimostratami e per avermi supportato nel mio lavoro, pur lasciandomi la libertà di esplorare e sviluppare autonomamente il progetto.

Ai miei genitori; grazie perché avete abbastanza forza per starmi vicino, abbastanza coraggio per starmi lontano e abbastanza amore per incontrarci a metà strada.

A Sere e Dami; io vivo in due mondi: in uno sono la sorella minore, nell'altro la maggiore. Grazie per renderli entrambi piú sopportabili, per dirmi dove devo andare e farmi credere di sapere dove sto andando, per tracciare la differenza tra affetto e affetto e poi dimenticarci che esiste.

Ad Ale, Cami e Giada, perché quasi tutto quello che so sul volersi bene l'ho imparato con voi.

Ai miei amici, perché anche quando parto o sparisco, so di avere un posto in cui poter sempre tornare. Siete la mia casa, la mia famiglia, la mia Tana.

A Giulia e Clara; c'era un tempo in cui a tenerci lontane erano gli spazi tra un banco e l'altro, in cui condividevamo un libro di inglese in tre, e scambiavamo le ore di lezione con ore di vite illimitate. Forse adesso il tempo è limitato, ma non per questo meno prezioso, e forse siamo un po' meno spensierate, ma di certo non meno vicine.

A Simo e Dani, perché tra le mie più grandi fortune c'è il fatto di avere un posto speciale come Siviglia da poter chiamare casa, e due persone speciali come voi da poter chiamare amiche.

Indice

Elenco delle figure

Introduzione

Nel contesto dell'evoluzione tecnologica e dell'importanza crescente dei dati nel settore sanitario, le applicazioni web in ambito medico rappresentano una risorsa importante per migliorare la qualità delle cure mediche e ottimizzare i processi decisionali clinici. Esse, infatti, semplificano la comunicazione medico-paziente, favoriscono la comprensione dei dati e facilitano il calcolo e la predizione, anche dove entrano in gioco dati in grande quantità e di elevata complessità.

Il presente lavoro di tesi si propone di esplorare tale contensto, attraverso lo sviluppo e l'implementazione di un Sistema di Supporto Decisionale (Decision Support System o DSS), partendo da un applicativo già esistente, Epimetheus.
Il contesto è, dunque, quello medico e, più precisamente, quello chirugico-ortopedico.
L'applicazione Epimetheus fornisce predizioni sull'esito di operazioni chirurgiche ortopediche, in particolare riguardo alla ripresa del paziente, assistendo così il medico nel processo decisionale clinico. In particolare, il supporto si propone di evidenziare un aspetto molto delicato di quello che è il percorso di decision-making, ovvero quello della rappresentazione dell'incertezza.

Obbiettivo della tesi

Con questa relazione, si vuole descrivere quanto svolto durante l'esperienza di stage presso l'Università degli Studi Milano Bicocca, in collaborazione con il laboratorio MUDI, del Dipartimento di Informatica, Sistemistica e Comunicazione (DISCo).
L'obiettivo dello stage riguarda lo sviluppo e deployment dell'applicazione web denominata Epimetheus, che implementa un Decision Support System in ambito di decision-making medico, con particolare attenzione agli aspetti di sviluppo back-end e di flusso logico. L'implementazione è in linguaggio Python ed è volta all'integrazione di componenti di Machine Learning preesistenti nell'applicativo sopra descritto.

La struttura della tesi è organizzata come segue: nel primo capitolo verrà approfondito lo stato dell'arte riguardante le applicazioni dell'intelligenza artificiale e i sistemi di supporto decisionali, soffermandosi anche sul concetto di incertezza; nel secondo capitolo verrà descritta la metodologia adottata nel progetto, dall'analisi dell'applicazione pre-esistente, alla progettazione e implementazione della nuova architettura back-end, alla messa in produzione dell'applicazione; infine, verranno esposte le conclusioni e i possibili sviluppi futuri per il progetto.

1. Contesto teorico

1.1 AI e Machine Learning in ambito medico

L'Intelligenza Artificiale e il Machine Learning stanno gradualmente rafforzando il loro impatto nella vita di tutti i giorni, e prendono sempre più posizione nel campo della sanità, influenzando il processo di diagnostica, prevenzione e cura della malattia.
L'impiego nel settore potrebbe rivoluzionare il sistema sanitario, rendendolo sempre più accurato, sicuro e veloce.

Nell'articolo *"Clinical applications of artifcial intelligence and machine learning in cancer diagnosis: looking into the future"*[4], Muhammad Javed Iqbal *et al.* analizzano gli sviluppi dell'AI nell'ambito oncologico.
È noto come le tecnologie di imaging medico (MIT) assistite da Intelligenza Artificiale, abbiano un ruolo fondamentale, in quanto sono in grado di interpretare e classificare immagini mediche, organizzare i dati conseguentemente, archiviare informazioni ed eseguire data mining. Grazie a questo ampio spettro di azione, l'AI, che è già un supporto fondamentale per il settore di radiologia, può essere declinata anche in contesti oncologici.
Oltre alle tecnologie MIT, un altro impiego è quello della progettazione di algoritmi per la rilevazione precoce dei tumori, attraverso, ad esempio, l'identificazione di biomarcatori (indicatori di processi fisiologici, patologici o di risposte biologiche all'intervento terapeutico). Ciò è utile anche per sviluppare farmaci di precisione, che in oncologia sono progettati per mirare specificamente alle cellule tumorali. Risulterebbe quindi possibile suggerire terapie efficaci, considerando fattori genetici personalizzati.
Pertanto, l'Intelligenza Artificiale può essere classificata tra le terapie futuristiche più avanzate per la diagnosi, la prognosi e il trattamento preciso dei tumori.

Secondo uno studio condotto da A. B. Lisacek-Kiosoglous e*t al.*, *"Artificial intelligence in orthopaedic surgery"*[1], l'AI potrebbe giocare un ruolo fondamentale anche in ambito ortopedico, nel prossimo futuro.
Le applicazioni esistenti in chirurgia ortopedica hanno già evidenziato successi nell'indi-

viduare impianti in artroplastica e nel segnalare caratteristiche o problemi relativi ad essi, come il cattivo posizionamento o allentamento delle protesi.
Risultano, poi, di particolare importanza gli algoritmi predittivi di AI e ML, per ottenere, ad esempio, stime sulla lunghezza della convalescenza o permanenza in ospedale e sui costi. Inoltre, possono giocare un ruolo fondamentale nell'ambito di riabilitazione e trattamenti postoperatori, facendo previsioni sui risultati delle operazioni e lo stato dei pazienti.

Un'altra applicazione di AI e Machine Learning in ambito medico, riguarda i Sistemi di Supporto Decisionale, che forniscono un valido aiuto ai medici nel processo di decision-making, e che analizzeremo più approfonditamente nel prossimo paragrafo.

1.2 Clinical Decision Support System

Un Sistema di Supporto Decisionale (*Decision Support System*) è un sistema informatizzato, che ha l'obbiettivo di assistere l'utente nel processo di *decision making*, analizzando e combinando dati e informazioni specifiche in un determinato ambito e fornendo un aiuto per comprendere problemi, valutare alternative e prendere decisioni informate.

Nel caso dei *Clinical Decision Support System* (CDSS), il contesto è quello medico; questo tipo di software rappresenta quindi uno strumento utile al personale sanitario.
Le caratteristiche di un singolo paziente sono confrontate con una base di conoscenza clinica computerizzata e la situazione specifica dei pazienti o le raccomandazioni sono poi presentate al medico per una decisione.
I CDSS oggi sono principalmente usati dal medico per combinare la propria conoscenza con i suggerimenti proposti dal sistema. Tuttavia, sempre più sistemi vengono sviluppati con la capacità di evidenziare dati e osservazioni altrimenti non ottenibili o non interpretabili dagli umani.
Attualmente, i CDSS fanno spesso uso di applicazioni web o integrazioni con cartelle cliniche elettroniche e CPOE (Immissioni computerizzate di ordini medici). Possono essere amministrati da desktop, tablet e smartphone, ma anche dispositivi di monitoraggio biometrico e tecnologia *wearable*.

Il loro spettro di utilizzo è vasto e include, ad esempio, diagnostica, sistemi di allerta, prescrizioni e controllo di medicinali.

In un analisi svolta da Reed T. Sutton *et al.* sui CDSS, dal titolo *"An overview of clinical decision support systems: benefits, risks, and strategies for success"*[5], vengono identificati tre componenti principali per questi sistemi:

1. La **base di conoscenza**, costituita dalle regole implementate nel sistema o gli algoritmi utilizzati per modellare le decisioni e dai dati disponibili.

2. Il **motore di inferenza**, che combina le regole (programmate o modellate da AI) e le applica ai dati specifici del paziente, generando un output da presentare all'utente.

3. Un **meccanismo di comunicazione**, per mezzo del quale il risultato finale viene mostrato all'utente; può essere un sito web, un'applicazione o un'interfaccia, tramite cui l'utente interagisce con il sistema.

Sempre sulla base sullo studio specifico condotto da Reed T. Sutton *et al.*[5], si andranno ora ad esporre alcuni benefici e rischi, derivanti dall'adozione di Sistemi di Supporto Decisionale in ambito sanitario.
Il primo vantaggio evidenziato riguarda la sicurezza del paziente; l'impiego di questi supporti, infatti, determina una riduzione dell'incidenza degli eventi avversi e degli errori, per esempio, nella prescrizione di farmaci.
Dal punto di vista delle organizzazioni sanitarie, si rileva un contenimento dei costi. Questo perché vengono notevolmente ridotte le duplicazioni di test e ordini, si possono sfruttare suggerimenti di opzioni di trattamento o farmaci più economici e vengono automatizzati passaggi tediosi, riducendo il carico di lavoro del personale medico.
Il beneficio maggiore deriva sicuramente dal fatto che i CDSS rappresentano un supporto diagnostico e decisionale per il medico e per il paziente stesso. Questi sistemi sono in grado di fornire suggerimenti diagnostici basati sui dati, interpretare risultati dei test di laboratorio e visualizzare e analizzare immagini mediche. Attraverso la comprensione e lo studio dei risultati proposti, medico e paziente hanno la possibilità di confrontarsi e compiere decisioni ponderate.

Per quanto riguarda gli svantaggi, si evidenzia innanzitutto un limite pratico, ovvero la necessità di competenze informatiche; spesso i CDSS possono richiedere conoscenze tecnologiche anche elevate per essere utilizzati, e ciò potrebbe precludere a una grande quantità di utenti la possibilità di beneficiare di questi sistemi.
Il secondo rischio riguarda invece la fiducia degli utenti e dei pazienti nei confronti di questo tipo di supporti. Da una parte ci può essere una mancanza di fiducia, che porta gli utilizzatori a non tenere in considerazione le linee guida e i suggerimenti forniti dal CDSS. L'estremo opposto, altrettanto pericoloso, è la dipendenza o la fiducia eccessiva nell'accuratezza del software.

I CDSS sono sicuramente un mezzo per avere una visione oggettiva dello stato del paziente e possono fornire suggerimenti basati sull'evidenza. Tuttavia, la medicina è raramente un contesto semplice e sicuro. Per questo motivo è importante, nel *decision-making* in ambito clinico, sviluppare sistemi che siano in grado di comunicare e rappresentare l'incertezza.

1.3 Rappresentazione dell'incertezza in contesti medici

La pratica della medicina non è mai un contesto semplice e assoluto. I dati che vengono utilizzati per prendere decisioni cliniche sono spesso ambigui, contraddittori o scarsi; perciò, un buon approccio richiede equilibrio tra l'esperienza del medico, le conoscenze disponibili e le preferenze del paziente.
L'incertezza è un elemento intrinseco alla pratica medica e influenza il processo delle decisioni di tipo clinico.

Nel suo elaborato di tesi, dal titolo *"Comunicare l'incertezza in ambito medico. Data visualization di supporto alla prognosi di interventi chirurgici e al decision making"* [2], L. Frontino, sottolinea come, al giorno d'oggi, la maggior parte dei medici si sentano obbligati a fornire certezze assolute ai propri pazienti. Questo, oltre a causare problemi di comunicazione tra le due parti, contribuisce, insieme all'analfabetismo numerico, ad aggravare l'incapacità delle persone di comprendere e ragionare obbiettivamente sulle incertezze. Concentrandosi solo sulla ricerca di riscontri e risultati certi, infatti, si incorre

facilmente in illusioni di certezza e ignoranza del rischio.
Bisogna ricordare, inoltre, che tra medico e paziente è presente un divario anche di tipo sociale, e che il paziente si trova spesso in una condizione di svantaggio rispetto al linguaggio utilizzato e alle conoscenze necessarie a comprendere informazioni in contesto medico.
È quindi naturale, che si vada verso un sempre più attivo processo di collaborazione tra medico, paziente e strumenti tecnologici. Uno dei risultati di questa sinergia sono proprio i Sistemi di Supporto Decisionale, che sono in grado di costruire un ponte tra le figure di medico e paziente, favorendo una comunicazione senza fraintendimenti.

Nell'articolo *"Uncertainty in Decision-Making in Medicine: A Scoping Review and Thematic Analysis of Conceptual Models"*, di Helou M. A. *et al.*, viene analizzato il ruolo dell'incertezza in ogni punto del processo di decision-making clinico: dalla prevenzione, alla diagnosi, fino alla scelta di cure e trattamenti. Gli autori propongono degli step attraverso cui il medico si può confrontare con situazioni di incertezza:

1. Riconoscimento dell'incertezza
2. Classificazione dell'incertezza
3. Considerazione di diverse prospettive e parti interessate
4. Acquisizione di conoscenze
5. Approccio alla decisione rispetto all'incertezza

I CDSS, in particolare, possono essere un valido aiuto durante le fasi di questo processo; fornendo supporto decisionale basato sull'evidenza, essi aiutano i medici a valutare i rischi, considerare alternative e ponderare i benefici e i limiti di diverse opzioni di trattamento.
Inoltre, incorporando modelli di ragionamento basati sulla probabilità, consentono di valutare l'occorrenza di diagnosi o esiti clinici, aiutando a prendere decisioni più informate e razionali, in contesti incerti e talvolta poco trasparenti.

Per quanto riguarda l'aspetto comunicativo, una buona comunicazione dell'incertezza passa anche attraverso la visualizzazione dei risultati forniti dai CDSS. Un buon supporto

decisionale, infatti, deve avere come requisito la capacità di mostrare in modo chiaro e diretto i dati calcolati, in modo che siano facilmente leggibili sia dal medico che dal paziente.

A questo scopo è stato ideato il progetto Epimetheus, un *Clinical Decision Support System* per l'ambiente chirurgico-ortopedico, di cui si discuterà più in dettaglio nel capitolo 2.

2. Il progetto Epimetheus

Il presente capitolo descrive il lavoro svolto durante il periodo di stage, presso il laboratorio MUDI dell'Università degli Studi Milano Bicocca.
MUDI è un laboratorio informatico di ricerca, del dipartimento di Informatica, Sistemistica e Comunicazione (DISCo) ed è focalizzato su interazione uomo-macchina, sistemi di supporto decisionali, machine learning e incertezza.

Il progetto oggetto di questa relazione riguarda l'implementazione e il *deployment* di una web app, che implementa un sistema di supporto alle decisioni in ambito clinico, denominato Epimetheus.
In particolare, gli obbiettivi del progetto vertevano sullo sviluppo back-end e di flusso logico, utilizzando come strumenti il linguaggio Python e il framework Flask e integrando componenti di Machine Learning preesistenti nel sistema.

In questo capitolo, si andrà a fare un'analisi dell'applicativo esistente in tutte le sue componenti, per poi esporre le fasi di sviluppo della nuova architettura back-end e infine la fase di *deployment* dell'applicazione in ambiente di produzione.

2.1 Analisi dell'applicativo esistente

Epimetheus è un supporto digitale alla prognosi di interventi chirurgici complessi in ambito ortopedico. Il progetto è legato a un'analisi qualitativa dello stato di salute dei pazienti, da cui poi viene ricavata una presentazione dei risultati, attraverso alcune visualizzazioni dati.
Più precisamente, si tratta di un *Decision Support System*, che analizza i dati del paziente in fase preoperatoria e calcola la probabilità e il livello di miglioramento dello stato del paziente, nei sei mesi successivi all'operazione.
I dati del paziente derivano da una serie di questionari, mentre le previsioni sono effettuate tramite modelli di Machine Learning.

Attraverso le visualizzazioni il medico potrà avviare un confronto con il paziente, per decidere la prognosi.

Il sistema è stato declinato in una web app, in modo da offrire agli utenti un accesso intuitivo e conveniente tramite browser.

2.1.1 Interfaccia

L'interfaccia è composta da due pagine: nella prima pagina (Figura 2.1) viene presentato un form di compilazione, in cui il medico inserisce i dati derivanti da questionari relativi allo stato di salute del paziente. Questa pagina consiste, a sua volta, di due step: un primo step per scegliere la procedura (cioè il tipo di operazione: *spine*, *hip* o *knee*) e il secondo in cui viene effettivamente richiesta la compilazione del form.
Una volta compilati tutti i campi, l'utente, cliccando un bottone, avvia il calcolo delle previsioni e, se non ci sono errori, atterra alla pagina dei risultati (Figura 2.2), in cui è possibile consultare le visualizzazioni dei dati.

2.1.2 Questionari

I dati inseriti dal medico nel form di compilazione derivano da questionari, compilati dal paziente stesso, in relazione al proprio stato di salute.
Di seguito, verranno analizzatile alcune voci di tali autovalutazioni, per comprenderne meglio il significato ed evidenziarne il range di valori, che ciascun elemento può assumere. Per questa analisi si fa riferimento al lavoro di tesi di L. Frontino [2].

- Classe ASA: la scala ASA è un sistema di classificazione per lo stato fisico del paziente, in relazione ai rischi. I suoi valori vanno da 0 (migliore) a 4 (peggiore).

- Morbidity: questo indice è utilizzato per quantificare l'incidenza e la gravità delle condizioni all'interno di una comunità o di un gruppo di pazienti. I valori di questo indice vanno da 0 a 100.

- ODI: l'Oswestry Disability Index è un questionario per valutare le disabilità in persone che soffrono di lombalgia sia acuta che cronica. Questa scala può assume

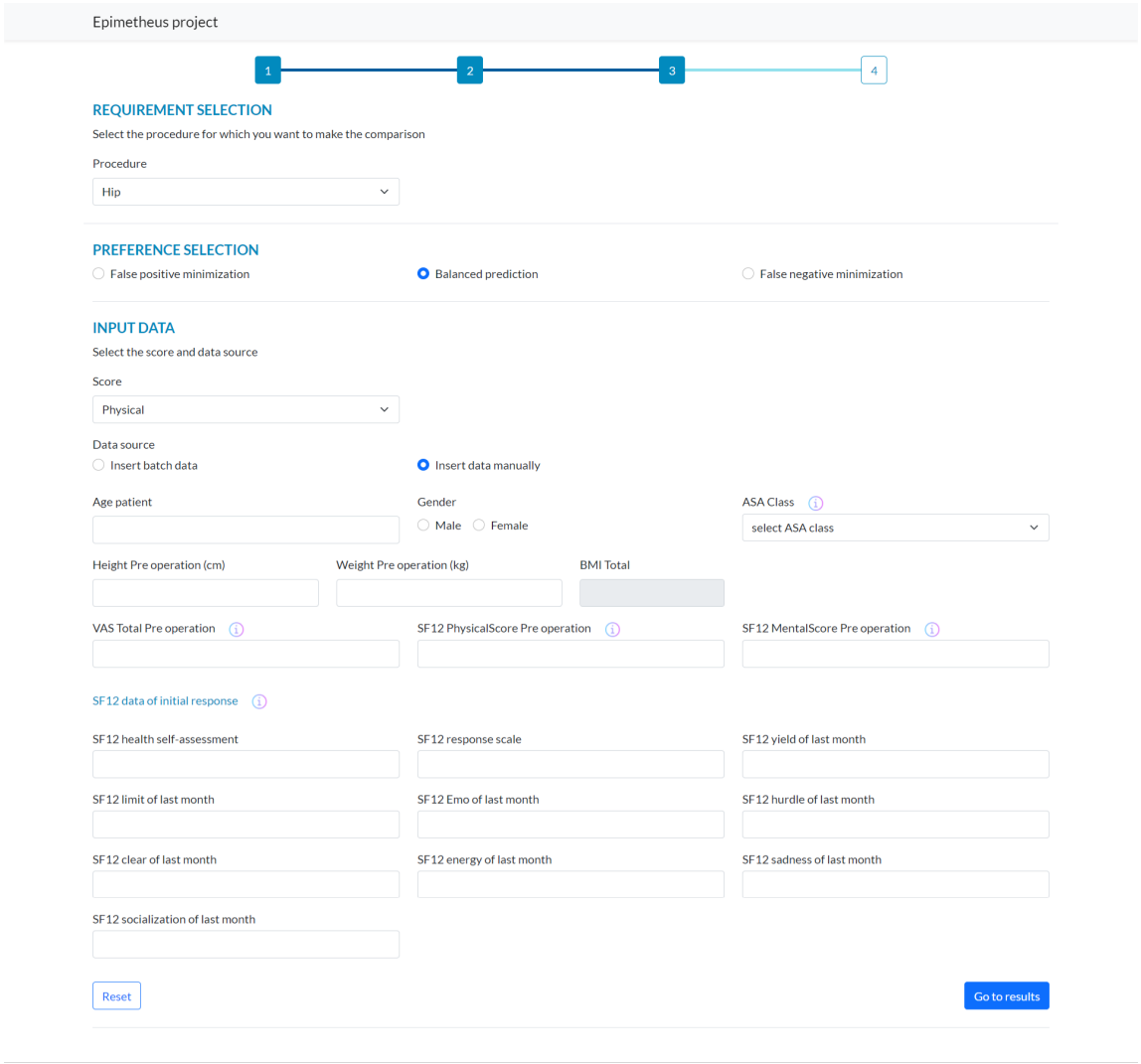

Figura 2.1: Applicazione web Epimetheus: la pagina del form

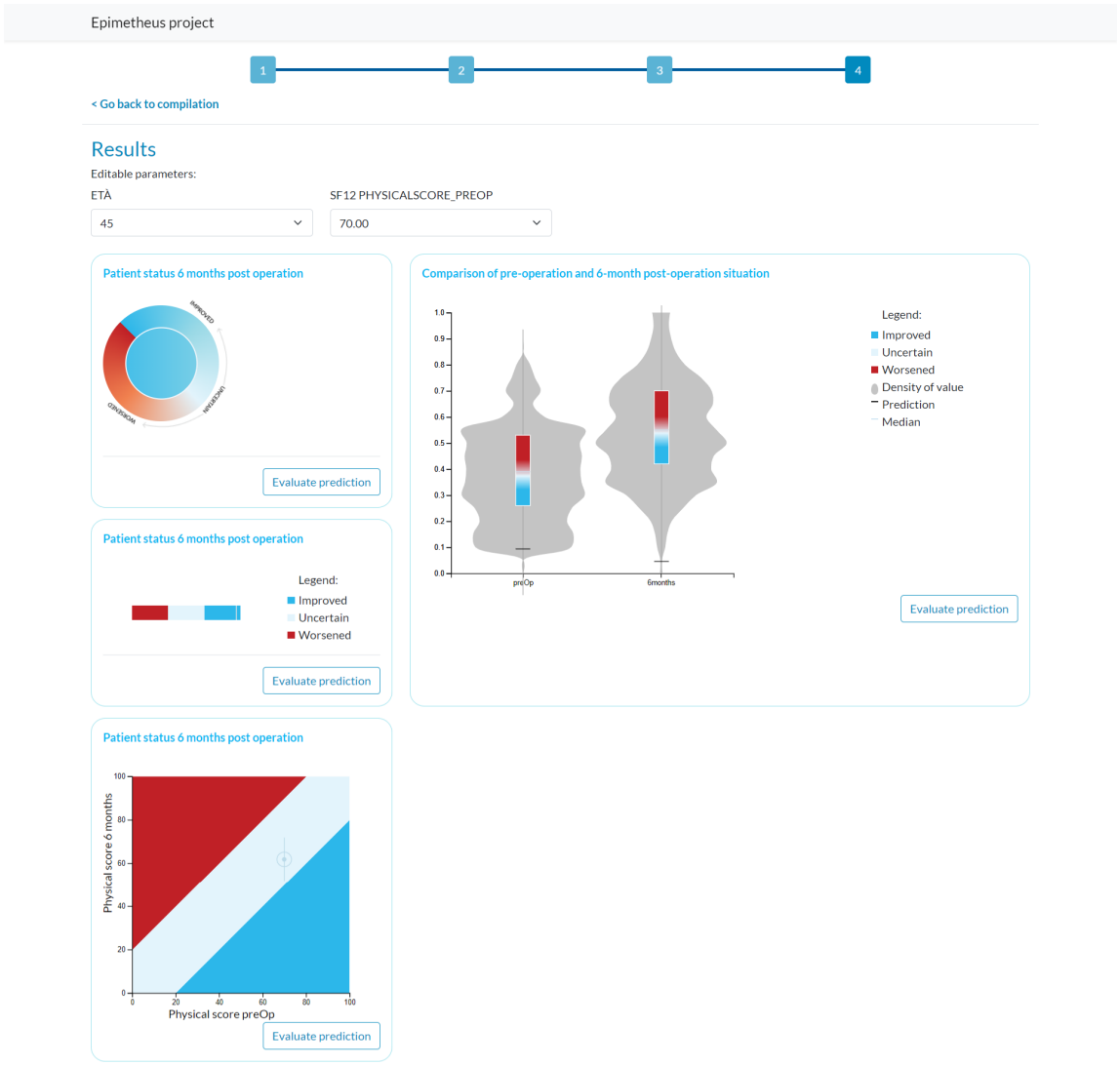

Figura 2.2: Applicazione web Epimetheus: la pagina dei risultati

valori da 0 (la migliore condizione di salute possibile) a 100 (la peggiore condizione di salute possibile)

- VAS: la Visual Analogue Scale è uno strumento di misurazione che tenta di misurare una caratteristica o atteggiamento variabile, attraverso una serie di valori difficili da misurare direttamente. L'intervallo di valori assunti da questo indice è 0-10.

- FABQ Work: il Fear-Avoidance Beliefs Questionnaire è un questionario self-report che quantica il dolore correlato a paure e credenze circa la necessità di modificare i comportamenti per evitare il dolore, e valuta l'associazione tra le credenze e le disabilità di lavoro e dell'attività fisica. Assume valori tra 0 e 66.

- SF12 e SF36: il questionario Short-Form Health Survey a 12 e 36 voci è un questionario psicometrico che permette di descrivere la salute percepita con 2 indici (fisico e mentale). Per il questionario a 36 voci, i valori assumibili stanno nell'intervallo 0-100; per quello a 12 voci, il range varia in base al campo: alcuni campi sono binari, altri hanno un intervallo di validità tra 0 e 4, altri ancora tra 0 e 5.

Ad oggi, Epimetheus permette di scegliere tra due differenti procedure: *hip/knee* (anca/-ginocchio) e *spine* (spina dorsale); ad ognuna di queste procedure è associata la rispettiva struttura del form, che si differenzia per i campi e i questionari tenuti in considerazione.

2.1.3 Modelli di Machine Learning e visualizzazioni

Come abbiamo visto, dopo aver compilato tutti i campi del form, il sistema Epimetheus procede al calcolo dei risultati, tramite modelli di Machine Learning, per poi visualizzarli nella pagina dedicata.
Il modello prende i dati in input, che corrispondono allo stato preoperatorio del paziente e va a prevedere quale sarà lo stato del paziente dopo sei mesi dall'operazione.
Ad ogni tipo di procedura sono associati due tipi di modelli di Machine Learning, entrambi della categoria di *Supervised Learning*, cioè che utilizzano dati etichettati; il primo è di tipo classificatorio e il secondo di tipo regressivo. Questa doppia scelta è funzionale alla rappresentazione dei dati; infatti, le due tipologie di modelli restituiscono output diversi,

che sono utilizzati per generare le visualizzazioni. Differenti diagrammi e grafici, dunque, si basano su differenti tipi di modelli di ML.
Il modello classificatorio prevede se il paziente migliorerà o peggiorerà. Il modello di regressione non solo prevede il miglioramento o peggioramento del paziente, ma anche il suo *score*.

Per quanto riguarda le visualizzazioni, nel sistema Epimetheus ne abbiamo quattro diverse:

- Uno **scatterplot** (Figura 2.3), che mostra un solo punto con un intervallo di confidenza per evidenziare il grado di incertezza. Le tre regioni definiscono se si prevede un miglioramento o peggioramento del paziente, oppure se ci si trova in una situazione di incertezza. Questo grafico viene costruito utilizzando gli output del modello classificatorio.

- Un grafico di tipo **stick bar** (Figura 2.4), creato anch'esso sulla base di modelli classificatori. La posizione della barretta verticale mostra il livello di certezza relativo al miglioramento o peggioramento del paziente.

- Un **grafico circolare** (Figura 2.5), che fa riferimento sempre ai modelli di tipo classificatorio. In questo caso, la zona centrale corrisponde alla predizione, mentre l'anello attorno funge da legenda per una corretta interpretazione.

- Un **violin plot** (Figura 2.6), che è invece costruito a partire dai modelli di tipo regressivo. In questo grafico viene mostrata la forma della distribuzione dei dati. Ad ogni livello l'ampiezza del "violino" indica se quel valore è presente nella distribuzione. Quindi, ad esempio, il valore che ha ampiezza massima corrisponde alla moda della distribuzione. Nel grafico viene inoltre mostrata la mediana. Una lineetta nera mostra il valore predetto dal modello. Anche qui è evidenziato il livello di incertezza riguardo il peggioramento o miglioramento del paziente, in base alle regioni di colore.

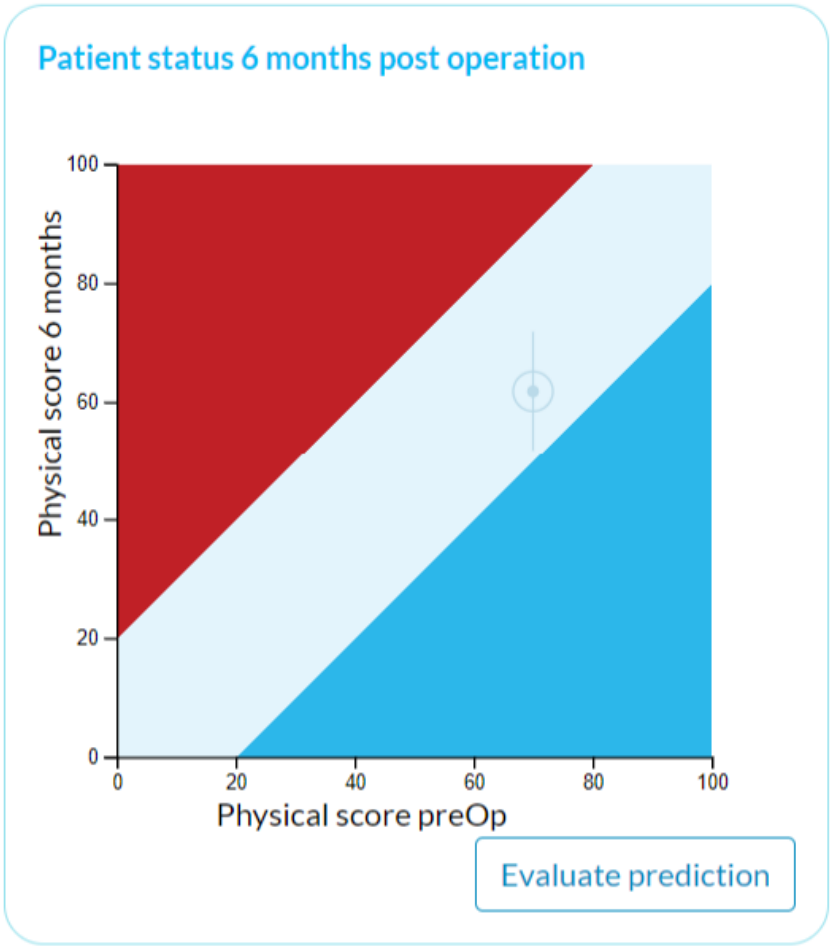

Figura 2.3: Diagramma di tipo scatterplot

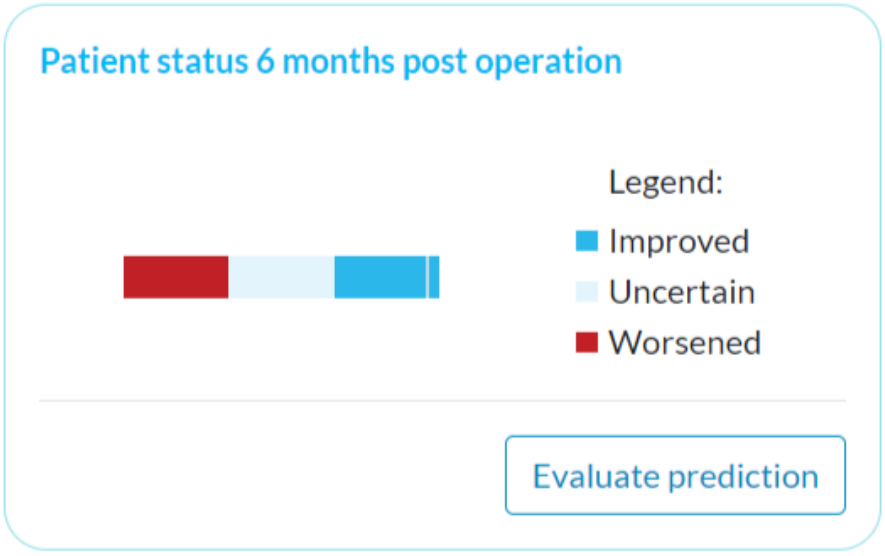

Figura 2.4: Diagramma stickbar

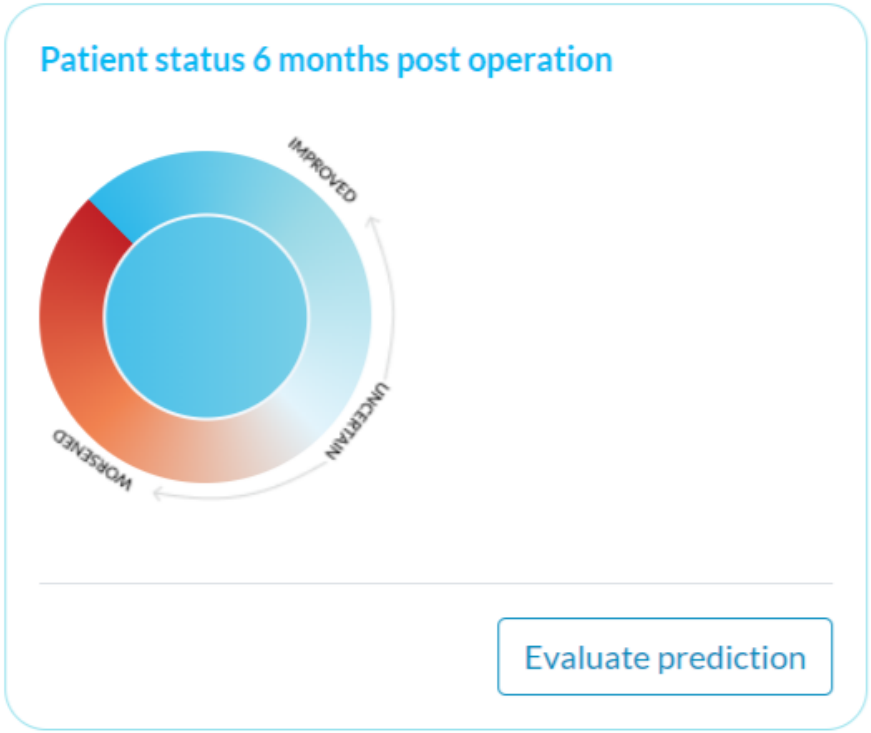

Figura 2.5: Diagramma circolare

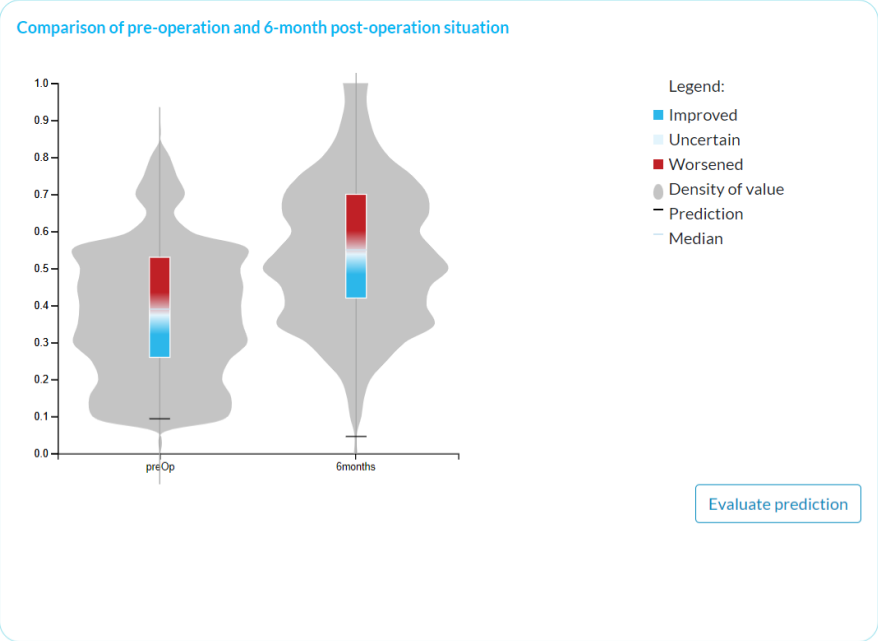

Figura 2.6: Diagramma di tipo violin plot

2.1.4 Tecnologie utilizzate

Per l'implementazione dell'applicativo sono stati utilizzati il linguaggio Python e il framework Flask.

Flask è una libreria Python usata come framework WSGI, che permette una facile e rapida implementazione di applicazioni web e API.

Elenchiamo di seguito alcuni dei motivi che rendono l'utilizzo di Flask una scelta popolare e potente per lo sviluppo di web app in Python:

- Semplicità: è progettato per essere semplice da imparare e utilizzare, con una struttura minimale e una sintassi chiara.
- Flessibilità: non impone alcuna struttura rigida alle applicazioni. L'essere estremamente flessibile consente agli sviluppatori di organizzare il codice come preferiscono e di utilizzare solo i componenti di cui hanno bisogno.
- Estensioni: offre una vasta gamma di estensioni che consentono di aggiungere funzionalità avanzate alle applicazioni con facilità.

- Leggerezza: è leggero e non ha molte dipendenze esterne; è lo sviluppatore a poter scegliere quali strumenti e librerie utilizzare.
- Documentazione: ha una buona documentazione che fornisce guide dettagliate e esempi pratici per aiutare gli sviluppatori.
- Comunità attiva: negli ultimi anni Flask è uno dei framework più popolari per lo sviluppo web, dunque è estremamente facile trovare supporto e risorse online per risolvere i problemi e migliorare le proprie abilità di sviluppo.

Per quanto riguarda la logica e i modelli di ML, alcune delle librerie utilizzate sono pandas, numpy, scipy, xgboost, scikit-learn.

La parte di UI e front-end, invece, è stata sviluppata in HTML, CSS e Javascript, con l'integrazione della libreria D3.js.

2.1.5 Punti deboli dell'applicativo

Lo studio appena descritto ci conduce ad evidenziare alcuni punti deboli dell'applicativo, che andremo di seguito ad analizzare.

La prima criticità è rappresentata dalla struttura del progetto e, in particolare, dalla sua organizzazione monolitica. La semplicità e la flessibilità del framework Flask consentono, infatti, di servire sia il front-end che il back-end dell'applicazione, tramite un unico blocco di codice. Ciò ha il vantaggio di accelerare il processo di sviluppo, oltre che di ridurre la complessità dell'architettura e rendere più semplice e diretta l'interazione tra le due parti di front-end e back-end.

Tuttavia, questo approccio può rappresentare una debolezza e comporta diversi svantaggi, soprattutto in vista di sviluppi futuri. In applicazioni di grandi dimensioni, infatti, mantenere il front-end e il back-end in un unico blocco di codice può portare a un aumento della complessità del codice. Questo potrebbe rendere più difficile la comprensione del codice da parte di nuovi sviluppatori e portare a una maggiore probabilità di errori.

Un altro fattore da considerare è la scalabilità: servire il front-end e il back-end dalla stessa applicazione può limitare la scalabilità della stessa, specialmente se il carico di lavoro aumenta notevolmente.

La mancata separazione tra FE e BE porta anche ad avere gran parte della logica applicativa gestita lato front-end; si può notare come molte funzioni di controllo del flusso logico o di validazione dei dati siano implementate tramite Javascript.
Un altro limite riguarda invece il form e la sua compilazione da parte dell'utente. Le variabili o elementi che si riferiscono ai campi del form sono impostati all'interno del codice Javascript, in maniera statica. Ciò è sconsigliabile dal punto di vista della manutenibilità, perché ogni volta che è necessario apportare modifiche o aggiornamenti ai campi del form, si deve modificare direttamente il codice Javascript. Tale procedimento può essere laborioso e aumentare il rischio di errori, soprattutto se ci sono molte dipendenze nel codice. In secondo luogo, se gli elementi del form sono codificati staticamente, può diventare difficile gestire e scalare l'applicazione in futuro.
Infine, dopo aver testato l'applicazione, è emerso che la maggior parte dei campi del form non hanno controlli che verifichino la correttezza dei dati inseriti dall'utente. Questo requisito è particolarmente importante, poiché le predizioni dei modelli, e quindi le visualizzazioni dei risultati, dipendono dai dati inseriti.

È dunque dall'osservazione e valutazione di questi punti critici, che nasce la necessità di esaminare una nuova architettura per il progetto Epimetheus, che verrà discussa nel successivo paragrafo.

2.2 Progettazione dell'architettura back-end

Tenendo in considerazione l'analisi svolta e le debolezze evidenziate, in questo paragrafo si andrà a delineare il progetto di una nuova architettura per Epimetheus.

Il requisito fondamentale e di maggiore rilevanza è la separazione tra la parte server back-end e la parte front-end.
La separazione delle responsabilità è una *best practice* di progettazione, che facilita la manutenzione, il testing e lo sviluppo parallelo.
Si tratta, quindi, di organizzare un'architettura su due livelli (*two tiers architecture*); questo tipo di pattern separa la UI dal server back-end funzionalmente e fisicamente e costruisce un ponte di comunicazione attraverso un API esposta dal server.

Questo design ha diversi vantaggi:

- UI e back-end possono essere sviluppati e testati come entità separate.
- L'applicazione avrà una maggiore scalabilità.
- Si ottiene una maggiore modularità, che facilita lo sviluppo per modifiche future.

Per quanto riguarda il lato front-end, il cui progetto e sviluppo non sono oggetto di questa relazione, esso sarà costituito da un'applicazione React. Al front-end sono affidate la gestione delle interfacce e le interazioni con l'utente e la "traduzione" delle predizioni in grafici facilmente comprensibili e interpretabili.
Il front-end comunicherà con il server back-end tramite specifici endpoint; inoltre verrà utilizzato il protocollo CORS (*Cross-Origin Resource Sharing*), per permettere l'interazione attraverso diversi domini.

Per il server back-end, si è scelto di mantenere l'utilizzo del framework Flask.
Questa parte si occuperà di mantenere la maggior parte della logica dell'applicazione. In particolare, sono affidate al back-end le seguenti operazioni:

- Selezione e invio al front-end della struttura del form selezionato dall'utente.
- Controllo e validazione dei dati del form.
- Calcolo delle predizioni tramite modelli di ML preesistenti.

Al fine di rispettare tali requisiti, si è deciso di implementare una RESTful API costituita da particolari endpoint, cioè *routes* speciali, in grado di inviare e ricevere dati da/al front-end.
In particolare, sono stati creati tre API endpoint: `get_form` e `predict_hip_and_knee` e `predict_spine`, che verranno analizzati nel paragrafo successivo.

Infine, si è pensato all'introduzione di una nuova classe Python, denominata `forms.py`, contenente le strutture di tutte le tipologie di form, sotto forma di dizionari chiave-valore. Svolgerà la funzione di database, per mantenere tutti i form esistenti.

Nella figura 2.7 sono mostrate le interazioni tra i diversi componenti.

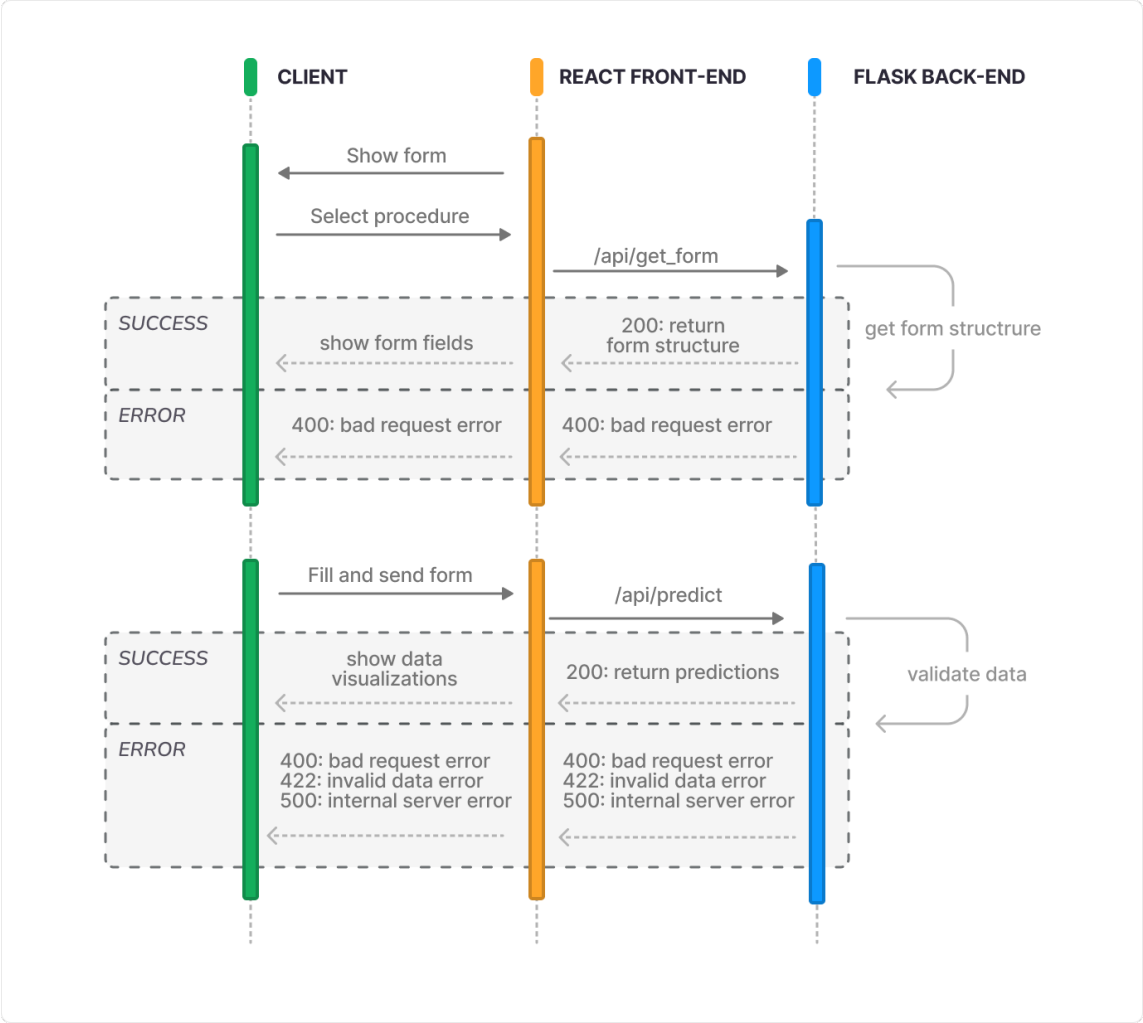

Figura 2.7: Schema delle interazioni tra front-end React e back-end Flask

2.3 Implementazione degli endpoint

Per quanto riguarda la comunicazione con il front-end, facendo riferimento al *flow* dell'applicazione, si è pensato a tre API endpoint: `get_form`, `predict_hip_and_knee` e `predict_spine`, che verranno descritti di seguito.

2.3.1 Endpoint get_form

Questo endpoint si occupa di inviare al front-end la struttura del form da presentare all'utente, in base alla scelta della procedura effettuata dallo user stesso.
Il server riceve dal front-end un file JSON con l'informazione del tipo di form richiesto; la risposta sarà un JSON contenente la struttura del form, recuperata dalla classe `forms.py`.

- Endpoint URL: `/api/get_form`
- Metodi HTTP: `POST`
- Formato della richiesta: JSON
- Parametri della richiesta:
  - selectedFormType(int): 0-1 (0 == hip/knee, 1 == spine)
- Esempio di richiesta:

```
{
"selectedFormType": 0
}
```

- Formato della risposta: JSON (ritorna un JSON con la struttura del form)
- Gestione degli errori:
  - 400 Bad Request: richiesta invalida, errore nella richiesta (JSON)

2.3.2 Endpoint predict_hip_knee

Questo endpoint si occupa di calcolare le predizioni per il modello della procedura *hip/knee*.
Il server Flask riceve dal front-end React i dati del form, sotto forma di file JSON; successivamente, tramite funzioni di validazione, controlla che i dati di ogni campo siano validi e coerenti con gli intervalli di valori, analizzati nel paragrafo precedente.
In caso di errori, ritorna un codice di errore. Se tutti i dati sono validi, utilizzando i modelli di ML opportuni, vengono calcolate le predizioni e inviate come JSON al front-end.

- Endpoint URL: `/api/predict_hip_knee`
- Metodi HTTP: `POST`
- Fromato della richiesta: JSON
- Parametri della richiesta:
  - sesso (int): 0-1
  - anni_ricovero (int): 0-120
  - classe_asa (int): 0-4
  - VAS_Total_PreOp (int): 0-10
  - physicalScore (int): 0-100
  - mentalScore (int): 0-100
  - bmi_altezza_preOp (int): 60-250
  - bmi_peso_preOp (int): 10-500
  - SF12_autovalsalute_risp_0 (int): 0-4
  - SF12_scale_risp_0 (int): 0-10
  - SF12_ultimomeseresa_risp_0 (int): 0-1
  - SF12_ultimomeselimite_risp_0 (int): 0-1
  - SF12_ultimomeseemo_risp_0 (int): 0-1

- SF12_ultimomeseostacolo_risp_0 (int): 0-4
- SF12_ultimomesesereno_risp_0 (int): 0-5
- SF12_ultimomeseenergia_risp_0 (int): 0-5
- SF12_ultimomesetriste_risp_0 (int): 0-5
- SF12_ultimomesesociale_risp_0 (int): 0-4
- zona_operazione (int): 0-2

- Esempio di richiesta:

```
{
    "sesso": 1,
    "anni_ricovero": 63,
    "classe_asa": 1,
    "VAS_Total_PreOp": 1,
    "physicalScore": 1,
    "mentalScore": 1,
    "bmi_altezza_preOp": 156,
    "bmi_peso_preOp": 84,
    "SF12_autovalsalute_risp_0": 1,
    "SF12_scale_risp_0": 1,
    "SF12_ultimomeseresa_risp_0": 1,
    "SF12_ultimomeselimite_risp_0": 1,
    "SF12_ultimomeseemo_risp_0": 1,
    "SF12_ultimomeseostacolo_risp_0": 1,
    "SF12_ultimomesesereno_risp_0": 1,
    "SF12_ultimomeseenergia_risp_0": 1,
    "SF12_ultimomesetriste_risp_0": 1,
    "SF12_ultimomesesociale_risp_0": 1,
    "zona_operazione": 0
}
```

- Formato della risposta: JSON (ritorna un JSON con le predizioni)
- Gestione degli errori:
  - 400 Bad Request: richiesta invalida, errore nella richiesta (JSON)
  - 422 Unprocessable Entity: uno o più parametri invalidi, errori nella validazione dei dati del form
  - 500 Internal Server Error: errore durante il calcolo delle predizioni.

2.3.3 Endpoint predict_spine

Analogamente al `predict_hip_knee`, questo endpoint si occupa di calcolare le predizioni, stavolta per la procedura *spine*.

- Endpoint URL: `/api/predict_spine`
- Metodi HTTP: `POST`
- Formato della richiesta: JSON
- Parametri della richiesta:
  - nome_operazione (int): 0-4
  - sesso (int): 0-1
  - anni_ricovero (int): 0-120
  - ODI_Total_PreOp (int): 0-100
  - VAS_Back_PreOp (int): 0-10
  - VAS_Leg_PreOp (int): 0-10
  - SF36_GeneralHealth_PreOp (int): 0-100
  - SF36_PhysicalFunctioning_PreOp (int): 0-100
  - SF36_RoleLimitPhysical_PreOp (int): 0-100

- SF36_RoleLimitEmotional_PreOp (int): 0-100
- SF36_SocialFunctioning_PreOp (int): 0-100
- SF36_Pain_PreOp (int): 0-100
- SF36_EnergyFatigue_PreOp (int): 0-100
- SF36_EmotionalWellBeing_PreOp (int): 0-100
- SF36_MentalScore_PreOp (int): 0-100
- SF36_PhysicalScore_PreOp (int): 0-100
- FABQ_Work_PreOp (int): 0-66
- classe_asa_1 (int): 0-4
- MORBIDITY (int): 0-100

- Esempio di richiesta:


```
{
    "nome_operazione": 1,
    "sesso": 0,
    "anni_ricovero": 4,
    "ODI_Total_PreOp": 3,
    "Vas_Back_PreOp": 3,
    "Vas_Leg_PreOp":3,
    "SF36_GeneralHealth_PreOp": 3,
    "SF36_PhysicalFunctioning_PreOp": 3,
    "SF36_RoleLimitPhysical_PreOp": 5,
    "SF36_RoleLimitEmotional_PreOp": 4,
    "SF36_SocialFunctioning_PreOp": 4,
    "SF36_Pain_PreOp": 4,
    "SF36_EnergyFatigue_PreOp": 4,
    "SF36_EmotionalWellBeing_PreOp": 4,
    "SF36_MentalScore_PreOp": 4,
```

```
    "SF36_PhysicalScore_PreOp": 4,
    "FABQ_Work_PreOp": 4,
    "classe_asa_1": 4,
    "MORBIDITY": 4
}
```

- Formato della risposta: JSON (ritorna un JSON con le predizioni)
- Gestione degli errori:
  - 400 Bad Request: richiesta invalida, errore nella richiesta (JSON)
  - 422 Unprocessable Entity: uno o più parametri invalidi, errori nella validazione dei dati del form
  - 500 Internal Server Error

2.4 Deployment dell'applicazione in produzione

Andremo ora a descrivere il processo di deployment dell'applicazione Epimetheus.
Rendere l'applicazione accessibile agli utenti permette di testare il sistema in un ambiente reale e raccogliere *feedback* dagli utenti.
Inoltre, contribuisce a promuovere l'accettazione di questo tipo di supporti: dimostrandone l'utilità e l'affidabilità nel contesto clinico, gli operatori sanitari potrebbero essere più propensi ad utilizzarli e ad integrarli nella loro pratica.

2.4.1 Server Apache come metodo di deployment

Durante la fase di valutazione per selezionare il metodo più appropriato di mettere l'applicazione in produzione, la prima scelta è stata quella di utilizzare un servizio di porting online, quale *Pythonanywhere* o simili. Questa soluzione ha il grande beneficio di essere molto conveniente e user-friendly per il deployment di applicazioni Python.

Occorre, però, considerare che la maggior parte dei servizi di porting online hanno limitazioni di spazio, specialmente per i loro piani gratuiti. Questi limiti possono riguardare diversi aspetti, tra cui lo spazio su disco disponibile per l'archiviazione dei file dell'applicazione; questo rappresenta un problema se sono coinvolti grandi quantità di dati o risorse, come può essere il caso di un sistema che implementa modelli di Machine Learning.
Anche il traffico mensile consentito è spesso ridotto per le versioni gratuite. Dunque, se l'applicazione ha un traffico elevato, si rischia di superare facilmente questi limiti, causando la sospensione temporanea del servizio.
Pertanto, per evitare che lo sviluppo presente e futuro di Epimetheus fosse soggetto a queste limitazioni, si è deciso di considerare un metodo alternativo, e rendere l'applicazione disponibile tramite Server Apache, configurato su una macchina ad indirizzo IP pubblico.

Questa scelta comporta un lavoro di tipo sistemistico ed è più dispendiosa in termini di tempo e lavoro, ma, allo stesso tempo, apporta numerosi benefici.
Innanzitutto, utilizzando Apache su un server dedicato, si ha controllo completo sull'ambiente di hosting, compresa la configurazione del server, l'accesso ai file di log e la gestione delle risorse. Apache offre, inoltre, una vasta gamma di opzioni di configurazione, consentendo di ottimizzare il server per le esigenze specifiche dell'applicazione.
In secondo luogo, utilizzando Apache, si ha la flessibilità di integrare l'applicazione Flask con altri servizi e componenti del server, come moduli aggiuntivi, servizi di autenticazione e autorizzazione, che potrebbero risultare utili per i futuri sviluppi dell'applicativo.
Apache, infine, è noto per la sua capacità di gestire un grande volume di traffico e di garantire prestazioni elevate, anche in presenza di picchi di utilizzo.

Per quanto riguarda l'architettura dell'applicazione, Epimetheus andrà ad interagire con il server Apache come mostrato nella figura 2.8. Esso fungerà da intermediario tra il client (browser web) e il front-end dell'applicazione, che a sua volta comunica con il back-end, tramite richieste HTTP, agli API endpoint; il server gestirà poi le risposte, restituendo l'output al client.

È quindi necessario configurare Apache sia per quanto riguarda la parte back-end, sia per il front-end React.
Nella prossima sezione, in particolare, ci soffermeremo sul processo per integrare il server

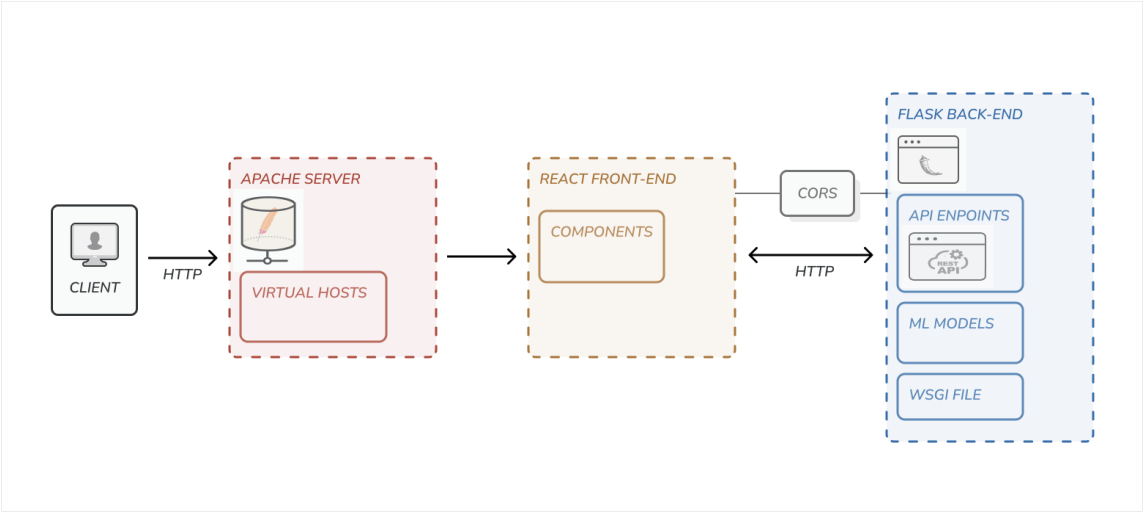

Figura 2.8: Diagramma dei componenti del sistema Epimetheus

Flask con Apache.

2.4.2 Integrazione di Flask con Apache

Verranno ora presentate le configurazioni necessarie per far funzionare correttamente il back-end Flask su Apache.

Abbiamo visto come il framework Flask sia molto popolare nello sviluppo di web app, essendo molto semplice ed agile. Tuttavia, non è così agile per quanto riguarda il deployment in produzione.

Per questo motivo, per l'implementazione del progetto, si è scelto di affidarsi all'utilizzo del modulo `mod_wsgi`.

`mod_wsgi` è un modulo Server Apache HTTP, ideato da Graham Dumpleton, che offre un'interfaccia WSGU per ospitare applicazioni web in Python, tramite server Apache.

Di seguito, elenchiamo i passi principali per la configurazione del server:

1. Eseguire il server Apache come Windows Service
2. Installare `mod_wsgi`
3. Usare `mod_wsgi` nel server Apache
4. Configurare un VirtualHost per l'app Flask

5. Creare un WSGI Script all'interno del progetto dell'applicazione.

Questi passaggi permettono di implementare il back-end Flask dell'applicazione su un server Apache.
È possibile consultare l'Appendice 2 (2.4.2), per una visione più tecnica e specifica di tale procedimento.

Conclusioni

Il risultato di questo progetto di tesi è stato lo sviluppo e l'implementazione del sistema Epimetheus, un *Clinical Decision Support System* nel contesto della prognosi d'interventi chirurgici complessi, con l'obiettivo di visualizzare l'incertezza dei dati, attraverso predizioni, calcolate con modelli di Machine Learning.

Lo studio degli aspetti teorici ha evidenziato l'importanza e l'utilità di questo tipo di strumenti in ambito medico.

Dall'analisi condotta sull'applicativo, inoltre, è emersa la necessità di pensare ad una nuova architettura per il progetto, che prevedesse la separazione tra front-end e back-end; il server back-end, che implementa una API RESTful, comunica con il front-end tramite specifici endpoint. Questa ristrutturazione ha reso il sistema meno rigido e più scalabile, e quindi pronto ad eventuali ampliamenti.

Tra gli sviluppi futuri, potrebbero essere integrati nuovi modelli di ML, e quindi nuove procedure su cui poter ottenere risultati. In questo senso, sarebbe utile lavorare alla creazione di un vero e proprio database per mantenere le strutture dei form delle diverse procedure.

Un altro aspetto utile rispetto alla creazione di basi di dati legate al sistema, sarebbe la possibilità di mantenere eventuali dati riguardanti gli utenti e gli utilizzi del sistema, al fine di poter creare statistiche.

Infine, il deployment dell'applicazione su un server pubblico consente l'utilizzo di Epimetheus a utenti esterni, permettendo l'avvio di una nuova fase di testing.

In questo contesto, una volta che la parte di front-end sarà pronta per essere messa in produzione, sarà possibile completare il deployment, integrando React con il server Apache in modo analogo a quanto è stato fatto con il back-end Flask.

Testare l'applicazione in un ambiente reale e ricevere *feedback* dagli utenti evidenzierà i punti forti e le criticità dell'applicativo, favorendo uno sviluppo continuo ed iterativo del sistema.

L'utilizzo da parte del personale medico, poi, contribuirà a promuovere l'accettazione di

questo tipo di supporti: dimostrarne l'utilità e l'affidabilità nel contesto clinico, incoraggerebbe gli operatori sanitari ad utilizzarla e a integrarla nella loro pratica.

Appendice 1

Avvio dell'applicazione Epimetheus in locale

In questa appendice si andrà ad illustrare il procedimento per avviare l'applicazione Epimetheus in ambiente locale.

Prerequisiti

1. Python 3.10 (È importante accertarsi che la versione di Python installata sia la 3.10; versioni precedenti o successive potrebbero generare errori con le dipendenze e le librerie necessarie.)
2. pip
3. Node.js

Avvio del server Flask

Installare tutte le dipendenze, posizionandosi nella cartella `<epimetheusHome>/backend` (per `<epimetheusHome>` si intende la posizione in cui si trova il progetto) ed eseguendo il comando:

```
pip install -r requirements.txt
```

Dopo la configurazione generale del progetto, posizionarsi nella cartella `<epimetheusHome>/backend` ed eseguire il comando

```
python flask_app.py
```

Se non ci sono errori il server Flask sarà in esecuzione su `http://localhost:5000`

Avvio del front-end React

Per avviare il front-end React, posizionarsi nella cartella `<epimetheusHome>/frontend` ed eseguire il comando

```
npm install
```

Questo comando installerà tutte le dipendenze e andrà rieseguito ogniqualvolta si apportano modifiche al file `packages.json`.

Per avviare l'applicazione React eseguire il comando

```
npm start
```

L'output sarà visibile all'url `http://localhost3000`.

Comunicazione tra front-end e back-end

Affinché il front-end React e il server Flask possano comunicare, è necessario configurare un proxy nel file `packages.json`, perciò è importante assicurarsi che in tale file sia presente la linea:

```
"proxy":"http://localhost:5000/"
```

In questo modo, quando si avvia l'applicazione React, tutte le richieste HTTP inviate dal front-end React saranno reindirizzate al server back-end Flask. Quindi, quando si fa una richiesta dal front-end e si indirizza la richiesta a una url relativa (ad esempio `/api/...`), il proxy in `packages.json` instraderà questa richiesta al server Flask in esecuzione su `localhost:5000`.

Appendice 2

Configurazione di un server per ospitare una applicazione Flask

Di seguito verrà illustrato il processo distribuzione di un'applicazione Flask su una macchina Windows, utilizzando un server Apache. In particolare, saranno presentati i passi per la configurazione del server, i principali problemi riscontrabili e le possibili soluzioni. Per la stesura di questa appendice è stato utilizzato come riferimento l'articolo di Rizal Maulana, *"Flask App Deployment in Windows (Apache-Server, mod_wsgi)"*[3].

Prerequisiti

1. Visual C++ Redistributable for Visual Studio
2. Visual C++ Build Tools (https://visualstudio.microsoft.com/visual-cpp-build-tools/)
3. python3 (Installato per tutti gli utenti, perciò fare attenzione che la directory di installazione non sia sotto `C:/Users`)
4. python3-pip
5. XAMPP Apache

Eseguire Apache come Windows Service

Posizionarsi nella directory `<Apache Home>/bin`. Per `<Apache Home>` si intende la posizione in cui è stato installato Apache.

```
cd C:/XAMPP/Apache/bin
```

Digitare il seguente comando:

```
httpd.exe -k install
```

Dopo aver installato Apache come Windows Service, è possibile utilizzare altri comandi come start, stop, restart, uninstall per gestire il server. La sintassi è la seguente:

```
httpd.exe -k [command]
```

Installare mod_wsgi

A questo punto è necessario installare mod-wsgi in Python, tramite pip. Prima di di questa operazione, però, è consigliabile configurare una variabile di ambiente:

```
set MOD_WSGI_APACHE_ROOTDIR=<Apache Home>

pip install mod-wsgi
```

In questa fase di installazione è possibile che si riscontrino degli errori. In tal caso, la prima possibilità è che non sia stata configurata correttamente la variabile di ambiente e bisogna quindi procedere come descritto sopra; se gli errori dovessero persistere, potrebbe essere necessario installare una diversa versione di `mod\_wsgi`. Si può tentare con diverse versioni fino a trovare quella adatta alla propria versione di Python.

Utilizzare mod_wsgi in Apache Server

Dopo aver installato `mod_wsgi` in Python, eseguire il seguente comando per ottenere la configurazione di`mod_wsgi`:

```
mod_wsgi-express module-config
```

Dovrebbe stampare un output simile a:

```
LoadFile \C:/Python38/python38.dll"
LoadModule wsgi_module \C:/python38/lib/site-
packages/mod\_wsgi/server/mod_wsgi.cp38-
win_amd64.pyd"
WSGIPythonHome \C:/Python38"
```

Copiare l'output ottenuto e incollarlo nel file di configurazione di Apache (`<Apache Home>/conf/httpd.conf`), prima delle dichiarazioni di tipo `LoadModule`. Dopo questa operazione, far ripartire il server, gestendo eventuali errori.
A questo punto è possibile aggiungere delle porte, in caso non si volesse utilizzare la porta di default di Apache, che è la porta 80. Per aggiungere una porta è sufficiente aggiungere il comando `Listen <port>` (sostituire `<port>` con il numero di porta desiderato) nel file `httpd.conf`.

Configurare un Virtual Host per una Flask App

Terminata la configurazione principale nel file `httpd.conf`, possiamo aprire il file di configurazione dei Virtual Host, situato in `<ApacheHome>/conf/extra/httpd-vhosts.conf`. Copiare nel file il seguente codice:

```
<VirtualHost *:5000>
ServerAdmin admin-name-here
ServerName server-name-here (e.g:localhost:5000)
WSGIScriptAlias / \F:/myapp/app/index/web.wsgi"
DocumentRoot \F:/myapp"
<Directory \F:/myapp/app/index">
Require all granted
Options Indexes FollowSymLinks Includes ExecCGI
</Directory>
AddHandler wsgi-script .wsgi //This is to ensure that wsgi script
    would be handled by apache handler.
ErrorLog \F:/myapp/logs/error.log"
CustomLog \F:/myapp/logs/access.log" common
</VirtualHost>
```

Analizziamo i campi più importanti di questa configuarazione:

1. Server Name: dominio dell'app se esiste, o semplicemente localhost

2. WSGIScriptAlias: ha due campi, il primo una url che indica il dominio, il secondo la directory in cui è posizionato il file wsgi (all'interno del progetto dell'app). Questo campo, quindi, dice al server Apache quale script deve eseguire quando c'è una richiesta per quel dominio.

3. DocumentRoot: directory con la root del progetto dell'applicazione

4. <Directory>: per ragioni di sicurezza dovrebbe garantire l'accesso allo script wsgi ma non all'intero progetto.

5. ErrorLog e CustomLog: posizione dei file di log; se l'applicazione non prevede file di log, può essere utile crearli manualmente.

Le directory possono essere scritte come stringhe, ma in caso di errori si consiglia di provare a scriverle semplicemente senza virgolette.
A questo punto è necessario riaprire il file `httpd.conf` e cercare la dicitura `httpd-vhosts.conf`; assicurarsi che sia presente (non commentata) la seguente linea:

```
# Virtual hosts
Include conf/extra/httpd-vhosts.conf
```

Per verificare che tutto sia stato configurato correttamente, è sufficiente fare un restart del server. Per risolvere eventuali errori controllare che tutte le informazioni della configurazione del Virtual Host coincidano e siano coerenti con il progetto dell'applicazione. Tra queste si intende: directory e url, porte, indirizzo IP del server, nomi dei file. Ogni volta che si modifica la struttura del progetto, è importante che i cambiamenti siano riflessi anche nella configurazione del Virtual Host. Si possono ottenere informazioni specifiche riguardo agli errori nel file `error.log` del Server Apache.

Creare uno Script WSGI

All'interno del progetto dell'applicazione, creare manualmente lo script wsgi. L'estensione del file può essere `.wsgi` o `.py` e si può utilizzare un nome esplicativo come `web.wsgi` o `wsgi.py`. All'interno dello script, copiare il seguente codice Python:

```
import sys
sys.path.insert(0, 'F:/myapp')
from your-app-script import app as application
```

La directory `F:/myapp` deve ovviamente puntare alla root del progetto.

`your-app-script` è il file Python in cui troviamo il `main` e dovrebbe essere simile a:

```
from flask import Flask

app = Flask(__name__)

@app.route(\/")
def hello():
return \Hello World"

if __name__ == \__main__":
app.run()
```

A questo punto la configurazione dell'app Flask, utilizzando un server XAMPP Apache su Windows, è completata.

Quando il server è in esecuzione, per eseguire l'app Flask è sufficiente collegarsi all'indirizzo dell'applicazione.

Se dovessero presentarsi problemi con la visibilità del server, vale a dire il server non è accessibile dall'esterno, si può procedere controllando le impostazioni del Firewall Windows. Deve essere presente una regola per il Server Apache che accetti (in entrata) il traffico proveniente dalla porta 80 (o la porta configurata per il server Apache). Inoltre, se c'è una regola che blocca il traffico in uscita per quella porta, essa deve essere eliminata.

Bibliografia

[1] A. Fontalis et al. A. B. Lisacek-Kiosoglous A. S. Powling. «Artificial intelligence in orthopaedic surgery: EXPLORING ITS APPLICATIONS, LIMITATIONS, AND FUTURE DIRECTION». In: (2023). URL: Bone%20Joint%20Res%202023;12(7):447%E2%80%93454..

[2] Loredana Frontino. *Comunicare l'incertezza in ambito medico. Data visualization di supporto alla prognosi di interventi chirurgici e al decision making.* 2023.

[3] Rizal Maulana. «Serve Flask App With XAMPP Apache on Windows». In: (2020). URL: https://medium.com/swlh/serve-flask-app-with-xampp-apache-on-windows-4debb4eb0b91.

[4] Haleema Sadia et al. Muhammad Javed Iqbal Zeeshan Javed. «Clinical applications of artifcial intelligence and machine learning in cancer diagnosis: looking into the future». In: (2021). URL: https://cancerci.biomedcentral.com/articles/10.1186/s12935-021-01981-1.

[5] Daniel C. Baumgart et al. Reed T. Sutton David Pincock. «An overview of clinical decision support systems: benefits, risks, and strategies for success». In: (2020). URL: www.nature.com/npjdigitalmed.