Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea in Informatica

Progettazione di un sistema per patient-reported outcome utilizzabile anche su dispositivi mobili

Tesi di Laurea in Basi di dati

Relatore:
Chiar.mo Prof. Danilo Montesi

Presentata da:
Matteo Brucato

Correlatore:
Chiar.mo Prof. Fabio Vitali

Sessione II
Anno Accademico 2010-2011

A buon intenditor poche parole..
Ma a volte è necessario essere un po' più prolissi.

Indice

Elenco delle figure

Capitolo 1

Introduzione

In questo elaborato prenderemo in esame la questione della progettazione di un sistema software atto a gestire alcuni dei problemi legati alla raccolta dei dati in ambito medico. Da tempo infatti si è capita l'importanza di una speciale tecnica di raccolta dei dati clinici, nota in letteratura col nome di *patient-reported outcome*, che prevede che siano i pazienti stessi a fornire le informazioni circa l'andamento di una cura, di un test clinico o, più semplicemente, informazioni sul loro stato di salute fisica o mentale. Vedremo in questa trattazione come ciò sia possibile e, soprattutto, come le tecniche e le tecnologie informatiche possano dare un grande contributo ai problemi di questo ambito. Mostreremo non solo come sia conveniente l'uso, in campo clinico, di tecniche automatiche di raccolta dei dati, della loro manipolazione, aggregazione e condivisione, ma anche come sia possibile realizzare un sistema moderno che risolva tutti questi problemi attraverso l'utilizzo di tecnologie esistenti, tecniche di modellazione dei dati strutturati e un approccio che, mediante un processo di generalizzazione, aiuti a semplificare lo sviluppo del software stesso.

Nel primo capitolo di questa tesi ci occuperemo di definire, nel modo più scientifico possibile, i concetti chiave che verranno utilizzati in tutto il resto della trattazione. Attraverso l'analisi della letteratura scientifica esistente, tracceremo una cornice ben precisa attorno alle questioni legate alla raccolta dei dati dai pazienti. Vedremo come sia possibile e perché sia importante ricevere input non

filtrati dal giudizio di terzi e scopriremo come i questionari siano stati scelti dalla comunità medica e scientifica come miglior mezzo per la raccolta dei dati di questo tipo. Vedremo quindi come il primo approccio, quello che prevede l'uso dei questionari tramite carta e penna, presenti numerosissimi problemi e come una soluzione elettronica sia invece considerata da tutti una via più consona. Nell'ultima parte del capitolo, infine, vedremo una breve rassegna delle tecnologie per patient-reported outcome già esistenti al giorno d'oggi e cercheremo di analizzarne le qualità e i difetti.

Il secondo capitolo presenterà l'idea alla base del progetto che stiamo realizzando. Sulla scorta delle considerazioni fatte in precedenza, mostreremo le soluzioni ai problemi esposti dai sistemi già esistenti e prepareremo la strada per le successive fasi di analisi dei requisiti e di progettazione, che caratterizzeranno il corpo principale del presente elaborato. Vedremo come alla base del sistema ci sia la possibilità di renderlo fruibile attraverso i moderni computer portatili touch-screen, in quanto, grazie alla loro estrema semplicità di utilizzo, rendono il sistema usabile a una percentuale di pazienti estremamente ampia. In un'importante sezione del capitolo, introdurremo il sistema per mezzo di esempi e casi d'uso. Essi spazieranno tra tutte le principali aree funzionali del sistema evitando, però, di entrare nei dettagli di progettazione.

Uno dei due capitoli chiave di questa tesi è il terzo capitolo. Esso raccoglie, in un modo il più completo e coerente possibile, il risultato dell'analisi dei requisiti del sistema di gestione dei patient-reported outcome che vogliamo progettare. In questo capitolo vedremo come il sistema sia organizzato mediante un'architettura client-server nella quale il server centrale possa usufruire di un sistema di memorizzazione permanente dei dati e comunichi con client che presentano caratteristiche diverse, che richiedono interfacce diverse e che hanno requisiti funzionali diversi. Mostreremo come le comunicazioni tra i client e il server avvengano mediante il comune protocollo HTTP e come i dati vengano scambiati nella forma di documenti XML. In una sezione del capitolo mostreremo l'approccio più consono alla modellazione dei dati, con particolare enfasi sulla strutturazione dei questionari, delle loro domande e opzioni e sulla raccolta dei risultati delle som-

ministrazioni, sottolineando l'importanza di un approccio generale che garantisca al sistema la possibilità di memorizzare questionari di ogni tipologia. Vedremo inoltre come il sistema preveda degli specifici requisiti di usabilità, giustificati dal fatto che le interfacce utente saranno utilizzate da pazienti sui quali non si possano fare alcune assunzioni di età, patologie, educazione, competenze informatiche, ecc.

Il capitolo sui requisiti si chiude con dei cenni sulle qualità del software che intendiamo sviluppare. Tra di esse spiccano la sicurezza, l'affidabilità, la correttezza e la manutenibilità. Il sistema, infatti, è destinato a essere utilizzato in un contesto molto delicato, nel quale eventuali debolezze, vulnerabilità o errori logici possono compromettere non solo l'integrità dei dati e dei risultati clinici e scientifici ma, talvolta, anche la salute della persona. In uno scenario del genere, è bene tener presente che i pazienti si trovano spesso nella condizione più sfavorevole ed è quindi anche compito del sistema stesso quello di implementare delle misure di difesa e di prevenzione, le quali possano essere osservate anche attraverso l'attuazione di determinati protocolli per la gestione degli utenti e per lo scambio e la condivisione dei dati tra di essi.

Il quarto capitolo presenta le nostre scelte progettuali, sulla scorta di tutte le considerazioni fatte nelle fasi di analisi dei requisiti. Mostreremo innanzitutto come sia possibile rappresentare i dati dei questionari in formato XML, sia per quanto concerne la modellazione di un questionario da compilare, sia per quanto riguarda l'invio dei risultati di un questionario già compilato. Raffineremo quindi la visione architetturale del server precedentemente introdotta e mostreremo come sia possibile implementare i requisiti discussi, mediante le tecnologie web di ultima generazione. Tratteremo inoltre della rappresentazione dei dati nel modello relazionale, introducendo innanzitutto la loro strutturazione in diagramma ER e passando infine per la loro implementazione in tabelle SQL. Il sistema lato server sarà implementato secondo un'architettura MVC (Model-View-Controller) che ha l'enorme vantaggio di disaccoppiare la reale rappresentazione dei dati dalla loro rappresentazione sulle interfacce utente.

Nella seconda parte del capitolo parleremo di come sia possibile realizzare

le interfacce utente per un così ampio numero di client, molto diversi tra loro, tutti con requisiti e interfacce differenti. Infine mostreremo come sia possibile effettuare le trasformazioni del formato dei dati, sia da XML a modello relazione, sia da XML ad HTML e viceversa. Una sezione finale del capitolo indicherà, invece, tutti quei miglioramenti e perfezionamenti del sistema non attualmente inclusi nella versione corrente del progetto, ma che potranno esserlo in versioni successive, qualora risultasse necessario.

Nell'ultimissima parte di questa trattazione esporremo il lavoro effettuato per la realizzazione di un semplice prototipo di esecuzione dei questionari. L'interfaccia realizzata permette a un paziente di completare un questionario che gli sia stato somministrato da uno dei suoi dottori, per mezzo di un qualunque dispositivo mobile, come tablet e smartphone, o tramite un computer fisso. Il prototipo, inoltre, è stato utilizzato per effettuare un certo numero di test atti a misurare tre caratteristiche importanti, relative all'interfaccia di esecuzione dei questionari: l'usabilità dell'interfaccia; la sua adeguatezza all'uso in ambito clinico da parte di pazienti di ogni tipologia; e il grado di fiducia verso il sistema, giudicato soltanto in base all'interfaccia stessa. I risultati hanno mostrato che il sistema è considerato estremamente facile e intuitivo, molto adatto all'ambito clinico in cui si pone di operare e si è visto che gli utenti reputano di potersi fidare abbastanza nell'eventualità di utilizzarlo per l'invio di loro dati personali. Tali risultati sono un incentivo importante alla continuazione del lavoro finora svolto, nella speranza che il sistema possa essere usato molto presto in ambiti reali, ove i patient-reported outcome siano reputati uno strumento indispensabile per la cura dei pazienti e per la ricerca medica.

Capitolo 2

Patient-reported outcome

2.1 Introduzione ai PRO

Il termine *Patient-reported outcome* è usato in ambito medico per indicare tutte quelle informazioni cliniche e sulla qualità della vita, nonché tutti gli strumenti e le tecniche atte a recuperare tali informazioni, quando esse sono fornite direttamente dai pazienti, senza cioè l'intermediazione analitica o valutativa di terzi, siano essi medici o semplici intervistatori. É usato ormai da parecchi anni e ha soppiantato il precedente termine *Health-related quality of life* (HRQL), ovvero "Qualità della vita in relazione alla salute" in quanto, secondo la *Food and Drug Administration* (FDA), l'organismo governativo americano che si occupa della regolamentazione dei prodotti farmaceutici e alimentari, meglio riflette l'intento di raccogliere la prospettiva dei pazienti circa l'andamento di un trattamento in svariati ambiti clinici o di ricerca medica [Wiklund, 2004]. In questo capitolo illustreremo in maniera più dettagliata gli aspetti fondamentali di questo concetto in modo tale da delineare il più possibile la struttura concettuale nella quale ci muoveremo nel resto del trattato. Vedremo innanzitutto di chiarire meglio il concetto di "outcome" ricevuto direttamente dal paziente, mostreremo perché questo sia così importante da aver scatenato una vasta produzione scientifica nonché tecnologica nel corso dell'ultimo decennio e infine cercheremo di individuare in che modo le tecniche e tecnologie informatiche possano essere d'aiuto e possano

fornire nuovi strumenti di supporto concreto alla realizzazione e all'utilizzo dei patient-reported outcome.

2.1.1 Dati e informazioni dei pazienti, dai pazienti

Non è difficile immaginare il processo mediante il quale un dottore controlli l'andamento di una certa terapia, oppure il processo mediante il quale un ricercatore controlli gli effetti a breve e lungo termine di un farmaco in via di sviluppo, che sia esso una cura a un problema fisico, psicologico o quant'altro. In ogni caso il metodo più semplice e intuitivo è un'intervista che il dottore sottopone al paziente, ovvero una serie di domande per lo più a risposta aperta, alle quali il paziente darà risposte più o meno precise, più o meno fraintendibili, più o meno utili e che il dottore cercherà al meglio di interpretare. Questa interpretazione è il fulcro chiave del problema delle interviste negli studi clinici e in generale nei vari ambiti clinici dove è richiesto un feedback da parte dei pazienti. Infatti l'interpretazione delle risposte dei pazienti, proprio in quanto tale, può non essere del tutto corretta, può risultare incompleta o addirittura sbagliata. In altre parole potremmo dire che l'utilizzo del linguaggio naturale al fine di creare valutazioni strettamente scientifiche può facilmente risultare ingannevole e fuorviante.

Proprio per questi motivi, nel corso degli anni, si è cercato di trovare delle soluzioni alternative e più scientificamente precise per la valutazione delle caratteristiche cliniche più difficili da misurare. Si è capito che ciò non può essere realizzato se non eliminando l'intermediazione (e quindi poi l'interpretazione) di alte persone nel processo di racconta delle informazioni. Le informazioni dei pazienti devono venire direttamente dai pazienti. Il processo di raccolta deve essere ben strutturato e non lasciare spazio a risposte vaghe e imprecise. In altre parole, le risposte a una domanda dovrebbero essere scelte da un insieme di risposte predefinite, oppure risultare dall'esito di misurazioni precise (come ad esempio quelle effettuate tramite strumenti di misurazione).

Da questa descrizione si evince come lo strumento che meglio si presta alla raccolta di tali informazioni sia il *questionario*. Un questionario proprio come quelli cui siamo più abituati a rispondere, fatto di semplici domande e un set ben

determinato di possibili risposte, nel quale basti semplicemente usare una matita per barrare le caselle relative alle risposte prescelte. Un questionario del genere potrebbe quindi essere compilato in maniera totalmente autonoma o, perché no, guidati da un intervistatore in quale si limiti semplicemente a registrare le scelte effettuate dal paziente. I questionari possono anche essere usati per raccogliere dati nel tempo e a scadenze ben precise, nel qual caso si parla più comunemente di *diari*. Essi sono da tantissimi anni molto usati nei trial farmaceutici e clinici in quanto forniscono importanti informazioni sull'esperienza dei pazienti circa l'utilizzo di un farmaco o di un trattamento [Stone et al., 2003]. I diari sono inoltre caratterizzati dal fatto di essere accompagnati dall'informazione circa il momento esatto in cui i dati vengono riportati dal paziente.

Per illustrare meglio questi concetti mostreremo qui di seguito un paio di esempi: in prima istanza considereremo il problema della qualità della vita in relazione alla salute (HRQL), nel quale i dati forniti dai pazienti servono a misurare caratteristiche legate al loro stato di salute (in generale molto difficili da ottenere con metodi alternativi); nel secondo esempio mostreremo una particolare scala di valutazione che si prefigge il compito, non facile, di misurare il livello di dolore dei pazienti.

HRQL

Il termine *health-related quality of life* è usato per misurare caratteristiche della vita dei pazienti strettamente legate alla qualità della vita e alla salute, unitamente ad altre caratteristiche meno legate alla salute, come per esempio il reddito o il concetto di libertà personale [Guyatt et al., 1993]. Non useremo questo testo per motivare la necessità di misurare tali caratteristiche della vita di un paziente, credendo che siano abbastanza chiare e ovvie e che il lettore possa trovare moltissimo al riguardo nella letteratura scientifica [Velikova et al., 2004, Wiklund, 2004, Taieb et al., 2011]. Un questionario potrebbe, per esempio, contenere una domanda che interroghi il paziente sul suo livello di salute generale [CDC, 2011]:

Direbbe che in generale il suo stato di salute è:

a) Eccellente
b) Molto buono
c) Buono
d) Discreto
e) Scarso

Altre domande potrebbero invece riguardare il genere di limitazioni che il paziente deve subire nella sua vita quotidiana a causa di un problema di salute, o ancora da quanto tempo egli o ella soffra di tali limitazioni, ecc [Guyatt et al., 1993].

Misurazione del dolore

Un altro tipo di questionario è stato proposto per misurare un'altra caratteristica fisica dei pazienti, oggettivamente molto difficile da analizzare: la misurazione del dolore. Il questionario sul dolore di McGill è stato proposto proprio al fine di risolvere tale problema e fornire una misurazione del dolore dei pazienti la più obiettiva possibile, sebbene la scala valuti un dato palesemente soggettivo. I dati risultati possono quindi essere trattati in maniera abbastanza corretta per calcoli statistici. Il questionario contiene tre cosiddette "misure": 1) l'indice di dolore (*pain rating index*), 2) il numero di parole scelte relative al dolore e 3) l'intensità attuale di dolore su una scala da 1 a 5 [Ronald and Melzack, 1975].

2.1.2 Vantaggi dei PRO

Questi esempi mostrano chiaramente l'importanza degli strumenti di misurazione patient-reported outcome, ovvero il più possibile non filtrati da terzi. I questionari sono la via più semplice e comoda per ottenere tali risultati: essi possono essere compilati in maniera del tutto autonoma, senza l'ausilio di intermediari. Ciò è fondamentale in quanto spesso gli osservatori esterni sovrastimano certe caratteristiche o, altre volte, ne sottostimano altre. É stato notato, ad esempio, che essi tendono a sottostimare l'impatto psicologico, così come quello del dolore, della nausea o del vomito. Possono inoltre basare il loro giudizio più su dati clinici,

come ad esempio i sintomi, quando invece, come è stato mostrato, tali valori sono poco correlati con la valutazione sulla qualità della vita di un paziente [Fayers and Machin, 2007]. Per questi e molti altri motivi i PRO sono considerati dalla comunità scientifica supplementi importanti alle più comuni tecniche di raccolta dati biochimici o psicologici [Wilson and Cleary, 1995] e strumenti unici per la raccolta di dati relativi alla salute dei pazienti e più in generale delle persone. Non è infatti detto che i PRO debbano essere somministrati soltanto a persone affette da qualche patologia: essi possono essere usati anche per valutazioni più generiche su soggetti sani (nel qual caso alcuni si riferiscono ai PRO col significato di "person"-reported outcome [Fayers and Machin, 2007]).

Un'altra ragione a favore dei PRO che reputiamo importante citare è il vantaggio concreto che essi offrono nel processo di valutazione di prodotti farmaceutici, come farmaci, macchinari per trattamenti medici o ancora prodotti biologici. I PRO infatti permettono di ricevere input dai pazienti ai quali i prodotti medici sono direttamente destinati. Nel 2006 l'FDA ha pubblicato una guida dettagliata sull'utilizzo dei patient-reported outcome come mezzo atto ad accreditare le dichiarazioni stampate nelle etichette dei prodotti medici e farmaceutici ("Patient-Reported Outcome Measures: Use in Medical Product Development to Support Labeling Claims" [Food and Drug Administration, 2006]). L'agenzia, infatti, vede nei PRO uno dei migliori strumenti atti a tal scopo e ne ha regolamentato l'utilizzo in tutte le sue sfumature: dalla creazione degli strumenti di raccolta delle informazioni, alla loro verifica e modifica e, infine, alla valutazione delle informazioni raccolte [Patrick et al., 2007].

Riassumendo, che i PRO siano considerati "patient"-reported o "person"-reported (o addirittura patient-"recorded", come è stato proposto nel caso di misurazioni di valori biochimici provenienti da strumentazione precisa e riportati direttamente dai pazienti), i questionari includono domande atte alla valutazione di svariati elementi, come sintomi, benessere, qualità della vita, percezione di un trattamento, rischio, soddisfazione circa una cura e soddisfazione circa la comunicazione col personale medico [Rothman et al., 2007]. I dati provenienti da tali questionari sono preziosi per svariati motivi: l'opinione diretta del paziente permette di

stimare eventi non osservabili quali ad esempio il dolore, la depressione, ecc.; anche per valori tipicamente più facili da osservare il paziente rimane sempre nella miglior posizione per poter giudicarli (pensiamo per esempio a informazioni circa l'obbedienza a una dieta); forniscono informazioni più affidabili rispetto all'uso di un intermediario che debba interpretare le parole del paziente; in poche parole i questionari PRO forniscono preziosissime informazioni impossibili da ottenere in altro modo [Gwaltney et al., 2008].

2.2 PRO elettronici

Finora abbiamo visto che i patient-reported outcome si ottengono tramite la semplice somministrazione di questionari. I questionari, infatti, sono caratterizzati dal fatto di essere intuitivi e solitamente abbastanza brevi: semplici collezioni di domande cui sono annesse possibili risposte. Il formato più largamente usato è stato per molti anni quello cartaceo, nel quale i questionari venivano compilati e consegnati al personale medico di presenza o inviati per posta. É intuitivo pensare (ed è stato largamente dimostrato) che tale metodo ha svariati problemi. In questa sezione mostreremo i problemi più comuni delle metodologie di raccolta dati cosiddette "paper-and-pencil" e vedremo come la soluzione più moderna e conveniente sia quella di utilizzare strumenti elettronici.

2.2.1 Soluzione carta e penna

Un esempio molto interessante di questionari su carta sono le cosiddette *scale di valutazione*, largamente utilizzate in ambito medico: dalla verifica della funzionalità alla stima di un danno in corso, alla valutazione dei rischi cui un paziente si trova a causa, ad esempio, di una grave situazione clinica. Le Figure a pagina 114 e nella pagina seguente, in Appendice A, mostrano un esempio di questionario cartaceo per la misurazione della disabilità relativa al dolore cervicale [Fairbanks et al., 1980]. L'esempio contiene le istruzioni sulla compilazione, dieci domande cui rispondere e una sezione in basso per commenti e dati del paziente. Nel questionario è possibile selezionare una sola delle possibili risposte per ogni domanda

ed è prevista la possibilità di saltarne qualcuna. Nella pagina successiva sono presenti le istruzioni per la computazione del punteggio e le regole per la sua interpretazione.

Tale questionario è tra i più semplici in assoluto, ma ciononostante introduce alcuni problemi. Per esempio, come si fa a forzare il paziente a non cerchiare più d'una risposta? Oppure, se il questionario fosse stato spedito a casa del paziente ed egli dovesse compilarlo entro una certa data, utile ai fini di una ricerca, come si potrebbe controllare che il paziente lo abbia compilato davvero nella data prevista? Inoltre, cosa succederebbe se i questionari compilati non pervenissero ai laboratori per tempo, o addirittura venissero persi durante la spedizione? Questi sono solo alcuni dei problemi che hanno convinto da anni la comunità scientifica della necessità di adottare uno strumento diverso, non più basato sul modello carta e penna ma puramente elettronico. D'altra parte è stato necessario convincersi della possibilità di convertire i già esistenti questionari su carta in formato elettronico e, soprattutto, che i due formati siano equivalenti, ovvero che stesse domande forniscano stesse risposte [Gwaltney et al., 2008, Velikova et al., 1999, Rebollo et al., 2010, Salaffi et al., 2009].

La soluzione elettronica è nota in tutto il mondo col nome di *Electronic Patient-reported outcome*, abbreviato `ePRO`. Gli `ePRO` offrono numerosissimi vantaggi che cercheremo di descrivere in modo conciso ma esauriente.

2.2.2 Vantaggi degli `ePRO`

L'esempio precedente mostra chiaramente uno dei vantaggi cardine dei questionari elettronici, ovvero che è possibile settare dei limiti e delle regole ben precise sui dati inseriti dagli utenti. É infatti possibile forzare l'inserimento di dati corretti, come ad esempio che i numeri inseriti siano in un certo range o che le date inserite siano logicamente corrette. Ma questi sono solo alcuni dei vantaggi offerti dagli `ePRO`. É possibile forzare il completamento di tutte le parti fondamentali di un questionario (aumentare cioè la cosiddetta *compliance* dei questionari [Stone et al., 2002, Stone et al., 2003]), aggiungere un'etichettatu-

ra temporale a ogni dato inserito, inviare i dati in tempo reale, somministrare i questionari a scadenze regolari e controllarne l'effettivo completamento. Gli `ePRO` sono più facili da compilare in quanto le interfacce grafiche possono semplificare di molto la comprensione delle domande mediante l'uso di icone, pulsanti, messaggi di aiuto, messaggi di errore, ecc. É possibile implementare navigazioni condizionali del tutto trasparenti all'utente, nelle quali le domande successive dipendono dalle risposte precedenti. L'uso dei patient-reported outcome elettronici semplifica moltissimo la raccolta dei dati, la loro analisi e permette l'utilizzo di tecniche automatiche per la segnalazione in tempo reale di fattori di rischio. I questionari elettronici inoltre riducono, per ovvi motivi, un utilizzo inutile di carta, con un vantaggio non indifferente sull'ambiente.

Uno dei supporti migliori per i questionari elettronici sono i computer tablet (come ad esempio l'iPad della Apple [Apple.com, 2011]) che hanno tipicamente display abbastanza grandi da risultare usabili per la stragrande maggioranza degli utenti e sono facilmente utilizzabili per mezzo degli schermi *touch-screen* per i quali, tipicamente, non è necessario un lungo periodo di training. I display touch-screen in generale e i tablet pc in particolare sono utilizzati da moltissimi anni nell'ambito dei patient-reported outcome, per l'analisi di moltissimi aspetti legati alla salute dei pazienti, al loro stato psicologico, al fattore di rischio legato al loro stato fisico e/o mentale, in svariati contesti e applicazioni (si veda [Lawrence et al., 2010] per la valutazione del rischio suicidio nell'HIV, [Salaffi et al., 2009] per la valutazione dell'artrite reumatoide, [Osterhaus et al., 2002] per l'analisi lo stato di salute e l'utilizzo di cure per pazienti con disordini dell'apparato locomotore, solo per citarne alcuni) e sono stati dimostrati in moltissimi contesti essere lo strumento preferito dai pazienti (si guardi per esempio [Salaffi et al., 2009, Eun-Hyun and Lee, 2009]), con percentuali di preferenza piuttosto alte rispetto ai test somministrati su carta o su strumenti elettronici non touch-screen, come ad esempio i classici computer con mouse e tastiera.

2.2.3 Tecnologie ePRO già esistenti

Già da parecchi anni e recentemente in maniera più massiccia sono stati proposti e usati con successo molti sistemi per la gestione dei questionari elettronici. Alcuni di essi sono stati inizialmente realizzati per l'utilizzo sui comuni computer per mezzo di mouse e tastiera, fin quando le nuove tecnologie non hanno permesso la realizzazione dei questionari compilabili per mezzo di monitor touch-screen. Un sistema molto famoso, usato in più di 450 cliniche nel mondo è fornito da *PHT* [PHT, 2011], un'azienda americana che ha sviluppato un sistema di raccolta e analisi dei dati ePRO per mezzo di appositi computer palmari. L'unico problema di tale sistema, a nostro parere, è il fatto che il servizio può essere utilizzato solamente su un ridottissimo numero di dispositivi palmari e non gode quindi della possibilità di essere usato sui cellulari, i computer o i tablet che la grande maggioranza degli utenti oggigiorno ha.

Un'altra azienda, la *Invivo Data* [Invivo Data, 2011], ha proposto una soluzione che permette l'utilizzo del loro sistema anche tramite Web. Non è purtroppo chiaro se il sistema sia utilizzabile anche mediante tablet pc o i comuni smartphone.

Ancora, esiste una soluzione, proposta da *Arrowhead Electronic Healthcare*, che a nostro avviso si avvicina di più alla soluzione ottimale, che permette di creare ed eseguire questionari su smartphone, tablet, computer e addirittura su sistemi IVR e tramite SMS [Arrowhead Electronic Healthcare, 2011]. Non è purtroppo possibile valutarne l'effettiva usabilità o efficacia in quanto i gestori del sistema non rilasciano facilmente le credenziali di accesso alle demo. Ci è quindi impossibile valutarne le caratteristiche e capire, per esempio, se sia possibile creare questionari per ogni tipo di studio oppure se esistano interfacce di utilizzo sui più comuni browser mobili o app mobili o ancora se il sistema permetta un adeguato livello di connessione tra medici e pazienti.

Un altro sistema che reputiamo utile menzionare è stato sviluppato nel Dipartimento di Scienze dell'Informazione dell'Università di Bologna da un gruppo di studenti. Il sistema si chiama *ProClara* e implementa un test per la misurazione dell'artrite reumatoide attraverso una speciale app su iPad [Migliorino, 2011]. Il

sistema implementa un'interfaccia molto intuitiva e usabile che si avvicina molto al tipo di interfaccia che reputiamo sia importante realizzare per un sistema di patient-reported outcome elettronici. Una pecca di tale sistema è il fatto che sia orientato all'utilizzo di un solo tipo di questionario, il quale è, come si dice in gergo, *hard-coded* all'interno della app stessa. In altre parole, non è possibile definire un nuovo questionario se non passando attraverso la modifica del codice della app stessa. Un altro problema di questa soluzione è che i dati dei pazienti risiedono nel dispositivo del medico e non sono quindi accessibili, a meno di copie e/o spostamenti, su dispositivi differenti.

Capitolo 3

ℙ.ROse: Un nuovo sistema per la gestione degli ePRO

Nel capitolo precedente abbiamo mostrato un quadro generale sui patient-reported outcome, sul perché sono usati in ambito medico e sul perché sono considerati essere importanti. Abbiamo quindi discusso dell'importanza di un approccio più moderno alla raccolta dei dati dei pazienti che passi attraverso l'utilizzo di sistemi informatizzati. In questo capitolo enunceremo la nostra soluzione al problema e mostreremo, mediante alcuni esempi, casi reali di utilizzo.

3.1 Patient-reported outcome Services

ℙ.ROse sta per *Patient-Reported Outcome SErvices* ed è un insieme di servizi informatizzati per la raccolta, l'analisi e la fruizione in senso lato di dati clinici provenienti in linea diretta dai pazienti. É possibile pensare a ℙ.ROse come a una *suite* di funzionalità, tutte incentrate sui patient-reported outcome: da un lato consente ai medici di definire questionari per i loro studi; da un altro lato permette, sempre ai medici, di somministrare i questionari ai loro pazienti e di raccogliere i dati delle somministrazioni in modo semplice e intuitivo; dall'altro lato permette ai singoli pazienti di ricevere i test somministrati, di eseguirli e, più in generale, di tenersi in contatto stretto coi loro medici. In questa sezione

mostreremo come il sistema che abbiamo pensato risolva in maniera ottimale molti dei problemi insiti nelle soluzioni già esistenti e forniremo quindi un quadro completo delle caratteristiche del sistema che intendiamo sviluppare.

3.1.1 Soluzioni

Il punto di forza di tale sistema risiede in un approccio generale a quello che sono i questionari online. In ℙ.ROse il concetto di questionario è generalizzato a una semplice collezione di domande, in modo da rendere possibile la creazione di questionari per ogni tipologia di ricerca o ambito clinico: dalla valutazione del dolore, alla valutazione sulla qualità della vita, alla misurazione del rischio in un determinato ambito clinico, ecc. Una delle soluzioni precedentemente discusse, ovvero il sistema denominato ProClara, implementa un solo tipo di test e l'inserimento di nuovi test prevede la loro scrittura a livello di codice. Un approccio di questo tipo non è accettabile se si vuole permettere l'utilizzo del sistema a una moltitudine sempre più grande di utenti e in contesti clinici non prestabiliti.

Un altro punto di forza di ℙ.ROse è la sua centralità nella gestione dei dati. Alcuni dei sistemi già esistenti prevedono la memorizzazione dei dati PRO su dispositivi client, ovvero sui dispositivi direttamente usati per la fruizione dei servizi stessi. Ciò ha numerosi svantaggi: 1) rende innanzitutto difficile il recupero e l'utilizzo dei dati da dispositivi diversi (oggigiorno moltissime persone utilizzano molteplici dispositivi informatici nell'arco della stessa giornata, dal computer di casa, al computer del lavoro, al cellulare passando infine al tablet); 2) presenta il rischio di perdita dei dati qualora il dispositivo incorra in gravi guasti; 3) non permette uno scambio facile di dati con altre persone e/o enti; 4) dal punto di vista della raccolta dati, inoltre, questo approccio obbliga il medico a far utilizzare il proprio dispositivo ai pazienti cui vuole somministrare i test: non è possibile, in altre parole, che ogni paziente utilizzi il proprio dispositivo elettronico. Ciò invece permetterebbe una maggiore autonomia nella somministrazione e infine semplificherebbe di gran lunga la raccolta dei dati. Per ovviare a tutti questi problemi ℙ.ROse offre un servizio in cui i dati risiedono al centro degli scambi di informazioni tra medici e pazienti, ovvero in un server centralizzato accessibile

tramite la rete. In questa prospettiva, i medici e i pazienti hanno il ruolo di clienti e il server centrale si occupa di raccogliere i dati e di fornire gli strumenti e i servizi necessari alla loro fruizione. Questo approccio, quindi, rende possibile l'utilizzo del sistema su ogni tipo di dispositivo cliente, in quanto i dati non risiedono nei singoli dispositivi ma in un server centrale. Ciò ha anche l'enorme vantaggio di permettere l'implementazione di misure di sicurezza speciali che evitino la perdita accidentale dei dati e semplifica la gestione dei dati e della loro confidenzialità, in quanto vi è un unico punto di accesso a essi.

Una delle caratteristiche previste dal sistema è l'utilizzo di interfacce utente altamente usabili, in quanto reputiamo che tale approccio sia fondamentale nell'ambito clinico, soprattutto se pensiamo di voler permettere il suo utilizzo a una fetta di popolazione che sia la più larga possibile. Nel prossimo capitolo discuteremo ampiamente i problemi legati all'usabilità (cfr. sezione 4.5.2), con particolare attenzione all'usabilità delle interfacce che verranno utilizzate dai pazienti. Vedremo come sia molto importante che vengano fatte delle scelte precise durante la progettazione di interfacce usabili e che tali scelte siano in accordo col tipo di utilizzo che se ne farà.

3.2 Casi d'uso

In questa sezione introdurremo ℙ.ROse mediante l'analisi dei sui casi d'uso più tipici. Nei capitolo successivi utilizzeremo le idee qui enunciate per guidare la progettazione del sistema verso la strada più consona al suo utilizzo.

3.2.1 Utenti del sistema

Il primo passo da considerare è quello della creazione della rete sociale di pazienti e dottori che ℙ.ROse intende gestire. Nel caso più comune, un dottore che voglia iniziare a utilizzare i servizi del nostro applicativo creerà un account di accesso in maniera autonoma. La stessa cosa può farla un paziente, qualora voglia, di sua spontanea volontà, entrare a far parte della comunità di ℙ.ROse.

In un altro caso tipico, invece, è il dottore stesso che vuole inserire un proprio paziente nel sistema. In tal caso il dottore creerà l'account per il proprio paziente, il quale riceverà un'email nel suo account di posta elettronica contenente un link da cliccare per accettare l'invito. Fino a quando il paziente non avrà accettato l'invito il suo account non sarà attivo e non potrà quindi ricevere nessuna somministrazione.

3.2.2 Creazione di una libreria di questionari

L'idea alla base del sistema è quella di permettere ai medici di somministrare questionari elettronici ai propri pazienti e tenere traccia dei loro risultati. Ad ogni modo, il prerequisito affinché tutto ciò possa avvenire è l'esistenza di questionari da somministrare. I questionari vengono creati da medici a partire da test cartacei già precedentemente utilizzati in ambito medico oppure sulla base di nuovi test clinici appositamente pensati per essere eseguiti sui dispositivi elettronici. Non è compito del sistema ℙ.ROse quello di validare l'attendibilità di un certo test clinico. Si assume che i testi introdotti dai medici abbiano la loro comprovata validità scientifica. D'altronde, come è ovvio, è anche ammissibile utilizzare dei nuovi questionari ePRO non ancora testati e sfruttare ℙ.ROse per la sua validazione scientifica.

In uno scenario del genere, nel quale un dottore voglia introdurre nel sistema un nuovo questionario, si prevede che egli acceda al servizio tramite un web browser su un computer desktop o laptop[1]. Il dottore accederà al suo pannello personale e cliccherà su un apposito tasto per la creazione di nuovi questionari. Il sistema porterà l'utente verso l'interfaccia di creazione dei questionari per mezzo della quale sarà possibile decidere ogni caratteristica del questionario che si intende creare. Per esempio il medico potrà stabilire il numero di domande, il testo da mostrare a video, i range corretti per i valori numerici di input, ecc. In ogni istante potrà decidere di lanciare una demo del questionario, al fine di testare personalmente il risultato del lavoro fino a quel momento svolto.

[1]Per questa operazione si escludono tablet o cellulari per via dell'overhead operazionale che solitamente tali dispositivi richiedono durante il loro uso via touch-screen.

Una volta stabilita la struttura del test, il medico proseguirà con l'invio dello stesso al server per la sua memorizzazione permanente. Il nuovo questionario creato andrà a far parte di una "libreria" di questionari ePRO disponibili su ℙ.ROse, fruibile da ogni medico iscritto al sistema e andrà quindi ad arricchire il sistema stesso e tutti gli utenti che ne faranno parte. Qualunque medico interessato potrà quindi immediatamente somministrare lo stesso test a qualunque suo paziente.

3.2.3 Somministrazione in ospedale

Uno dei casi d'uso più importanti dei servizi ePRO in generale, riguarda la somministrazione dei test direttamente in ospedale o in clinica. Le interfacce utente per l'esecuzione dei testi sono sviluppate per essere eseguite sui comuni tablet o sui cellulari smartphone, dispositivi cioè usati principalmente mediante gli schermi touch-screen. Un medico potrebbe, ad esempio, dare un tablet a ogni suo paziente in sala d'attesa prima della visita, o addirittura durante la visita stessa, e chiedergli di compilare uno o più questionari. I risultati di tali test sarebbero resi subito disponibili al medico e visionabili in modo congiunto coi precedenti test dello stesso paziente o con test simili effettuati da altri pazienti.

Se per esempio un medico stesse controllando l'andamento della malattia di un suo paziente, potrebbe decidere di somministrargli un test ogniqualvolta il paziente venga a trovarlo nel suo studio. Ciò consentirebbe al medico di potere seguire in modo molto semplice ed effettivo l'andamento della malattia e potere quindi prendere le dovute decisioni sui trattamenti da utilizzare. Il sistema, inoltre, permetterebbe al medico di accedere a tutte le informazioni del paziente mediante un unico punto di accesso e di poter visionare in modo immediato la sua anamnesi completa.

3.2.4 Somministrazione a casa

D'altro canto è anche auspicabile che i pazienti possano compilare i questionari da casa. Tale situazione è molto utile ad esempio nel caso dei diari, per i quali spesso è necessaria una compilazione giornaliera o comunque molto ben caden-

zata. Un dottore, quindi, potrà decidere di somministrare i questionari ai propri pazienti in un qualunque momento della giornata. I pazienti utilizzeranno i loro dispositivi elettronici personali direttamente da casa, da lavoro o da qualunque altro luogo per accedere al sistema, visualizzare la lista dei test da compilare e infine completarne la compilazione. Il sistema provvederà alla raccolta dei dati e permetterà la loro fruizione in tempo reale ai medici, ponendosi al centro dello scambio dati tra medici e pazienti.

In uno scenario del genere, è auspicabile che il sistema sia in grado di lanciare degli allarmi ai medici nel caso in cui certi valori riportati dai pazienti che hanno eseguito un testo risultino fuori norma. Il sistema potrebbe essere quindi utilizzato anche come supporto alla medicina di emergenza.

3.2.5 Utilizzo di apparecchiature di misurazione

Le tecnologie esistenti oggi permettono di interfacciare un tablet, come ad esempio un iPad, con dispositivi hardware di ogni tipo. Un altro uso del nostro sistema potrebbe prevedere l'impiego dei tablet congiuntamente ad apparecchiature hardware di misurazione clinica, come ad esempio strumenti per la misurazione della glicemia o della pressione. Il paziente, invece di inserire tali valori numerici di sua mano, utilizzerebbe proprio una di queste apparecchiature. Esse, interfacciandosi direttamente con ℙ.ROse, registrerebbero automaticamente i valori misurati. Uno dei vantaggi chiave di questo sistema è, ovviamente, quello di evitare l'inserimento di dati erronei (per esempio dovuti a errore umano) e semplificare la raccolta di dati biochimici (quelli che alcuni ricercatori chiamano "patient-reported outcome"). Tali apparecchiature, inoltre, potrebbero facilmente essere usate negli ospedali o nelle cliniche direttamente dal personale medico ospedaliero, per la raccolta dei dati dei pazienti in un modo del tutto innovativo ed estremamente semplificato.

Capitolo 4

Requisiti

Nel precedente capitolo abbiamo presentato la nostra soluzione al problema di fornire degli strumenti moderni, generali, multi-piattaforma e altamente usabili per i patient-reported outcome. Il sistema $\mathbb{P}$.ROSE si prefigge il compito di fornire numerosi servizi atti alla creazione e all'utilizzo di strumenti di misurazione e valutazione dello stato di salute dei pazienti, a 360 gradi. $\mathbb{P}$.ROSE è un sistema molto ricco di funzionalità e complesso e necessita quindi di una progettazione attenta e dettagliata, soprattutto se consideriamo l'importanza che esso ricoprirà nella ricerca medica e nella qualità della vita e la salute dei pazienti che lo utilizzeranno. Il sistema verrà utilizzato da vari tipi di utenti, tra cui medici, ricercatori e pazienti e pertanto è bene porre particolare attenzione alla progettazione delle interfacce utente, ognuna pensata per rispondere alle esigenze di una determinata categoria di utenti. É inoltre necessario che il sistema risponda a ben determinate caratteristiche quali possono essere per esempio la sicurezza dei dati, la stabilità e la flessibilità del sistema stesso. Vedremo in questo capitolo di delineare meglio tutti questi concetti.

Analizzeremo qui di seguito i requisiti del sistema in tutte le sue parti e ne stileremo una accurata descrizione. Come è noto l'analisi dei requisiti è un passo fondamentale nel processo di creazione del software, poiché è il fulcro che guida le successive fasi di progettazione, implementazione e verifica. La famosa associazione di standard IEEE (*Institute of Electrical and Electronics Engineers*) ha

da tempo pubblicato un documento tecnico che fornisce ottime linee guida per la fase di specifica dei requisiti di un prodotto software [IEEE, 1998]. Nel nostro testo cercheremo di seguire alcune delle linee guida proposte dal consorzio americano, ovvero quei consigli che reputiamo più importanti, senza però rimanere obbligati nella estremamente rigida struttura logica da loro stabilita. Delineeremo il sistema ℙ.ROse secondo la sua architettura più astratta, le sue interfacce con gli utenti e gli altri sistemi software, le sue funzionalità e le qualità software che reputiamo fondamentali. Cercheremo di fornire una specifica dei requisiti che sia il più possibile non ambigua e completa. Cercheremo anche di rendere la specifica facilmente modificabile nell'eventuale (spesso non raro) caso di una necessaria revisione e/o miglioramento della stessa. Nel capitolo successivo, invece, andremo a utilizzare i concetti qui espressi per stilare una progettazione del sistema di ℙ.ROse che sia il più possibile in linea con essi. Andremo quindi a chiarire tutti quei punti che in questo capitolo lasceremo volutamente indefiniti. Prenderemo cioè le decisioni finali sulla progettazione, sulla base dei requisiti qui espressi.

4.1 Requisiti generali

In questa sezione chiariremo innanzitutto il tipo di sistema che vogliamo progettare, le sue funzionalità e caratteristiche principali, i suoi vincoli e le assunzioni che è necessario fare in fase progettuale e di utilizzo. Nelle sezioni successive andremo maggiormente nel dettaglio, per ognuna delle caratteristiche ivi introdotte.

ℙ.ROse sarà un insieme di servizi software per la gestione di patient-reported outcome di ogni tipo, basato su tecnologie web di ultima generazione, centralizzato e fruibile anche su dispositivi mobili. Il sistema lo si può innanzitutto vedere come un contenitore centrale di dati medici direttamente provenienti dai pazienti, cui sia quindi associato un sistema di memorizzazione stabile e affidabile dei dati, sia esso un database, un datastore [Zahariev, 2009] o quant'altro. I dati presenti in questa parte del sistema dovranno potere essere acceduti e/o modificati da

persone autorizzate: per esempio i medici, da un lato, dovranno potere accedere ai dati dei pazienti che si trovano sotto la loro ala, mentre i pazienti, dall'altro lato, dovranno potere essere liberi di inserire i loro dati quando e come richiesto dai medici. Il database dovrà quindi essere interfacciato a una componente software, lato server, che si occupi di ricevere e gestire tutte queste richieste. Il fatto che tale componente software risieda in un solo server fisico o in un cluster di server fisici o di macchine virtuali distribuite non è, al momento, un problema fondamentale.

Il server e il database sono le componenti centrali del sistema, ove risiedono i dati e la logica di accesso a essi. Il server dovrà essere in grado di accettare connessioni HTTP e HTTPS da ogni parte del mondo e dovrà fornire dei servizi di autenticazione e di invio e ricezione sicura dei dati. Ad esempio il server fornirà ai pazienti accesso ai questionari `PRO` che i medici hanno loro somministrato, insieme ad altre informazioni di corredo sempre provenienti dai dottori, come ad esempio email private, allarmi, ecc. D'altra parte il sistema permetterà l'invio al server dei questionari compilati dai pazienti e il loro salvataggio permanente e fornirà modi di accesso a tali dati al personale autorizzato.

Ai dottori dovrà esser permesso di tenere traccia costante dei propri pazienti e dei risultati che essi inviano al server centrale. Dovranno essere implementati diversi modi di visualizzazione dei dati provenienti dai pazienti, come per esempio grafici e statistiche e dovranno essere presenti metodi di comunicazione coi pazienti in cura. Inoltre i medici e ricercatori dovranno poter creare nuovi strumenti di misurazione, come questionari e diari e poterli somministrare ai loro pazienti sia manualmente che in maniera automatica.

Al fine di completare il quadro generale qui proposto, è necessario sottolineare che il sistema dovrà implementare delle politiche ben precise di accesso e protezione dei dati. In particolare sarà necessario impedire l'accesso ai dati dei pazienti a medici coi quali i pazienti stessi non hanno stipulato un accordo diretto. Ad esempio, se un paziente avrà accettato di partecipare a un trial clinico con un medico, i suoi dati `PRO` saranno messi a disposizione soltanto di quel medico o del gruppo di medici che lavoreranno al trial. Non dovrà essere in nessun mo-

do permesso l'accesso ai dati di un paziente da parte di altri utenti se non per esplicita richiesta del paziente e ciò implica, tra le altre cose, che il sistema non debba permettere il libero scambio di dati dei pazienti tra i dottori. Ciò a tutela dei pazienti e delle loro informazioni.

Il sistema, nel suo complesso, sarà un organismo autosufficiente, ma dovrà essere ugualmente prevista la possibilità di interfacciarlo ad altri sistemi che trattino dati medici. Ovvero la progettazione dovrà lasciar spazio a possibili futuri cambiamenti nelle interfacce software, che possano renderlo in grado di ricevere e inviare dati da e verso altri sistemi che gestiscano dati clinici generali e/o dati PRO.

Un'altra caratteristica che la progettazione del sistema dovrà prevedere è il possibile utilizzo del software in unione a speciali apparecchiature hardware per la raccolta di dati biochimici, quali ad esempio macchine per la misurazione della glicemia o della pressione. É in fatti in continuo aumento il numero di dispositivi medici direttamente collegabili a smartphone o tablet che permettono di effettuare misurazioni cliniche col proprio cellulare o computer touch-screen. Il sistema ℙ.ROse dovrà quindi essere progettato per facilitarne l'aggiornamento e l'interfacciamento con tali dispositivi quando questi entreranno più massivamente nel mercato. Ciò renderà quindi possibile la raccolta dati cosiddetti patient-*recorded* outcome (cfr. pagina 11 del presente testo).

4.2 Requisiti di sistema

Il tipo di scenario in cui ℙ.ROse si propone di operare necessita di una certa flessibilità nella specifica dei requisiti di sistema. In particolare, poiché l'obiettivo principale è appunto quello di rendere il sistema utilizzabile da diversi supporti e soprattutto dai dispositivi mobili, è necessario prevedere tale eventualità si dalle fasi embrionali di progettazione. Si specifica quindi che i sistemi che utilizzeranno ℙ.ROse siano computer classici (desktop e laptop) nonché dispositivi mobili come smartphone e tablet pc. Sarà auspicabile rendere in futuro il sistema utilizzabile anche attraverso linea telefonica e sistema di messaggistica SMS.

Nel caso di utilizzo del software su computer "tradizionali" (come ad esempio un computer desktop), si prevede la fruizione dei servizi esclusivamente online, via web browser, facendo in modo che il software non sia (ove possibile) dipendente da nessun browser in particolare e soprattutto che abbia lo stesso comportamento su ogni tipo di browser. Questo deve esser vero sia per quanto riguarda la grafica dell'interfaccia, che deve essere appunto omogenea tra un browser e un'altro, sia per quanto riguarda l'esecuzione di codice lato client, ovvero senza che vi sia differenza di funzionalità o di risultato delle funzionalità.

Nel caso invece di utilizzo del software su supporti mobili, si prevede un doppio utilizzo. Da un lato l'accesso via web browser, con le stesse restrizioni già espresse precedentemente ma con un'interfaccia grafica adatta alle dimensioni del supporto; dall'altro lato deve essere possibile una fruizione delle funzionalità del software per mezzo di apposite *app*, ovvero applicazioni che girano nei più comuni sistemi operativi per dispositivi mobili. In particolare si prevede l'utilizzo di ℙ.ROse su dispositivi con sistema operativo iOS e Android, al fine di coprire un ampio bacino di utenza. Sintetizziamo in questa frase quanto stabilito in questa sezione:

> *Il sistema sarà utilizzabile su diverse piattaforme hardware e software. Da un lato avrà un'interfaccia verso i più comuni web browser desktop e mobili, dall'altro delle interfacce in stile app per smartphone e tablet.*

4.3 Architettura del sistema e interfacce

Il sistema ℙ.ROse è una tipica applicazione web client-server su protocollo HTTP che fa uso di svariate tecnologie per la fruizione dei dati, l'esecuzione delle richieste dei client e la presentazione finale dei risultati di tali richieste. Possiamo individuare tre componenti principali nel sistema: da un lato vi sono i client che richiedono risorse al sistema; da un altro lato vi è una componente server centrale che si occupa di ricevere le richieste dai vari client e produrre i risultati; infine vi è una componente di memorizzazione persistente dei dati che comunica

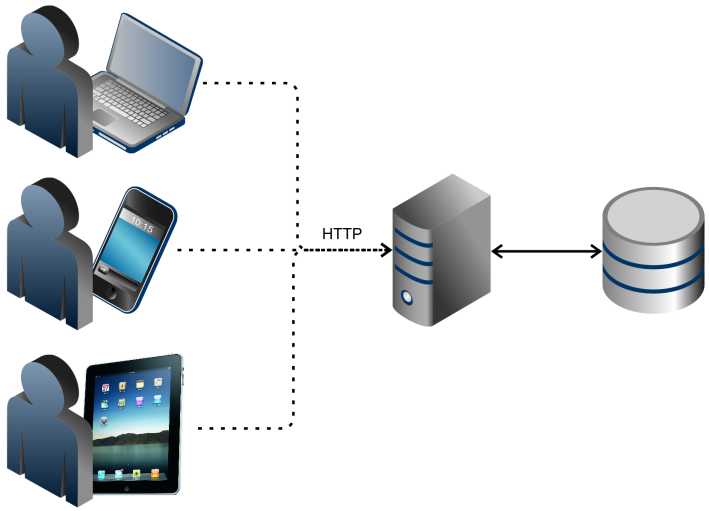

Figura 4.1: Architettura generale del sistema.

soltanto con il server e gli fornisce primitive di accesso ai dati. La Figura 4.1[1] illustra l'architettura appena descritta a parole. Ultima parte del sistema sono le interfacce utente prodotte dai client a seguito delle interrogazioni al server. Esse sono diverse a seconda del tipo di client che l'utente sta utilizzando in un dato momento.

4.3.1 Interfacce software e di comunicazione

Una prima interfaccia di comunicazione connette il server al database mediante l'uso di comuni primitive di interrogazione (per esempio mediante query in SQL, se si trattasse di un database relazionale). L'unica entità del sistema che può accedere al database è il server il quale può modificarne lo stato aggiungendo, modificando e rimuovendo dati in esso.

Possiamo ulteriormente raffinare la visione del sistema lato client aggiungendo una serie di dettagli fondamentali sulle interfacce software e di comunicazione tra le componenti. Per quanto riguarda i client, il sistema può essere acceduto attraverso almeno due tipi di dispositivi: i classici computer desktop/laptop e i dispositivi mobili (come smartphone e tablet). Gli ultimi, in particolare, forni-

[1]Per realizzare questa e la Figura 4.3 è stato utilizzato *Gliffy* [Gliffy.com, 2011].

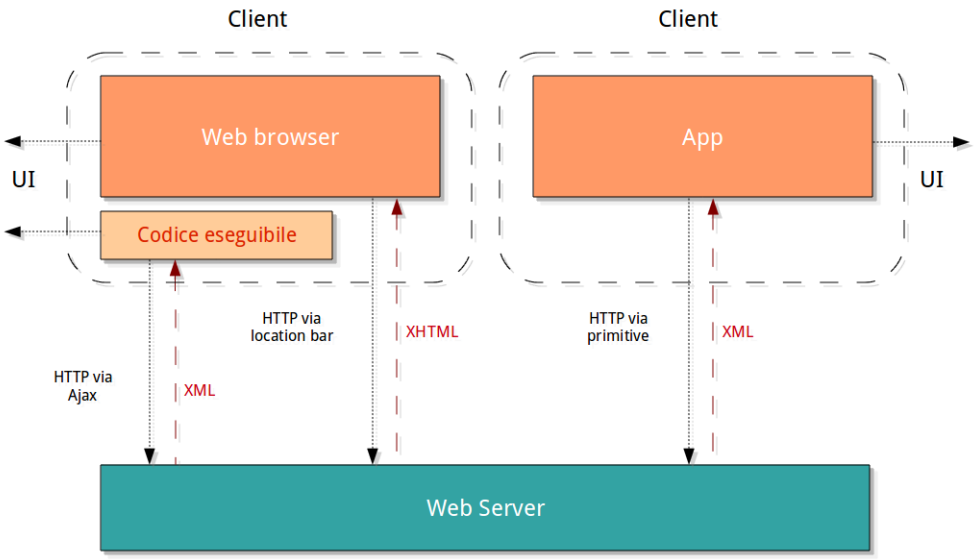

Figura 4.2: Schema di interazione client-server.

scono a loro volta due modi distinti di accesso al sistema: uno via browser e uno via app (applicazione nativa di un sistema operativo mobile).

Ogni client ha una componente eseguibile che può, tra le altre cose, comunicare col server. In Figura 4.2 è mostrato un maggior dettaglio sulla comunicazione client-server. Il client che gira su un web browser può comunicare col server in due modi distinti: 1) tramite location bar del browser (o in modo equivalente tramite reindirizzamento diretto via link); 2) tramite chiamate asincrone Ajax (ovvero tramite la primitiva `XMLHTTPRequest()`) [Garrett, 2005]. Il client che gira su una app invece userà delle primitive per chiamate HTTP fornite tipicamente dal linguaggio di programmazione usato. Come si evince in figura i diversi metodi di comunicazioni si appoggiano tutti sul protocollo di comunicazione HTTP (o HTTPS, sua estensione "sicura").

Lo scenario qui descritto prevede quindi la realizzazione di una componente server capace di comunicare con diversi tipi di client. Per ovviare alla complessità di tale situazione è necessario stabilire un'unica interfaccia di comunicazione e scambio dati tra il server e i diversi client. Si predispone quindi che il server e i client si scambino i dati tramite XML, il quale permette alle due componenti

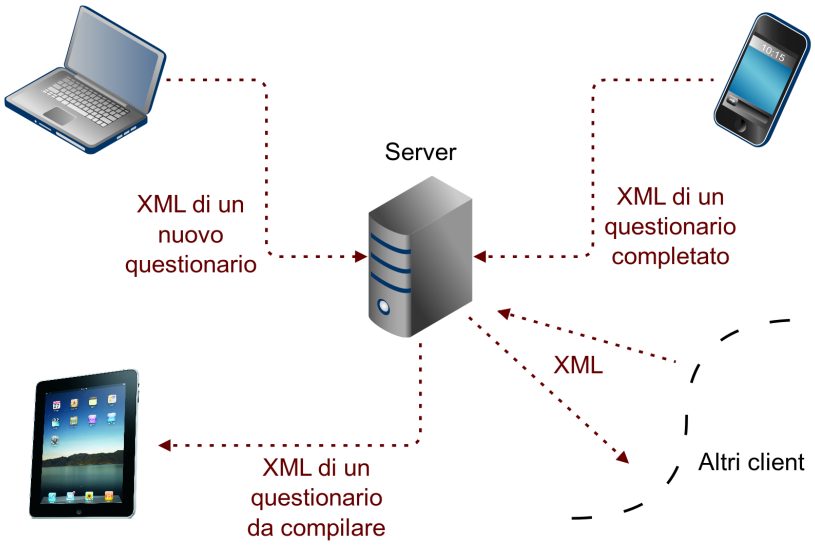

Figura 4.3: Schema di scambio dati tra client e server.

client e server di comunicare facilmente in maniera distaccata dalla logica che ne governa il loro funzionamento interno.

Questo approccio ha un enorme vantaggio: permette l'integrazione di client `ePRO` già esistenti con ℙ.`ROSE`. Ad esempio un client specializzato in un unico tipo di questionario (come è il caso di ProClara, di cui abbiamo precedentemente parlato) potrebbe comunicare col server mediante la sua interfaccia XML e utilizzarne il database per il salvataggio permanente dei dati. Altri client già esistenti potrebbero essere facilmente messi in comunicazione con ℙ.`ROSE` semplicemente aggiungendo dei moduli di conversione dei dati dai loro formati interni al formato XML di ℙ.`ROSE`. In altre parole questo approccio permette a client esterni di sfruttare la memorizzazione centralizzata dei dati e la rete sociale di dottori e pazienti forniti da ℙ.`ROSE`. In tale scenario, uno sviluppatore esterno potrebbe concentrarsi soltanto sulla creazione di interfacce client per la raccolta di dati `PRO` oppure su interfacce per la definizione di nuovi tipi di questionario, i quali inviino al nostro server una rappresentazione in XML dei dati raccolti sul client.

Il server quindi deve essere in grado di ricevere interrogazioni via HTTP e infine di ricevere e inviare dati da e verso i clienti attraverso XML. La Figura 4.3 mostra tale tipologia di interfaccia tra server e client, la quale si presta ad almeno

due utilizzi tipici del sistema ℙ.ROse:

Compilazione di un questionario

Nel primo caso il paziente vuole compilare un questionario: attraverso l'interfaccia utente il paziente decide quale questionario eseguire e il cliente richiede a suo nome tale questionario al server, il quale glielo fornisce in versione XML; il client utilizza tale XML per la realizzazione dell'interfaccia utente relativa alla compilazione del questionario stesso, raccoglie quindi i valori scelti dell'utente e li invia al server in formato XML[2] (un formato XML potenzialmente differente dal precedente).

Creazione di un nuovo questionario

Il secondo caso riguarda un dottore che vuole creare un nuovo tipo di questionario: egli utilizza l'interfaccia fornita dal client per la creazione del nuovo questionario; il client infine impacchetta il questionario in un XML e lo invia al server il quale lo legge e lo persiste nel database.

Tale approccio ricorda molto quello noto col nome di Model Driven Architecture (MDA) [Soley and the OMG Staff Strategy Group, 2000]. ℙ.ROse fa infatti uso di XML al fine di creare una rappresentazione dei questionari (ovvero un suo modello) che l'applicazione possa gestire in modo automatico nell'ambiente distribuito della rete, astraendo dalle differenze insite in ogni istanza del modello stesso. I vantaggi di tale approccio sono ampiamente discussi nella tesi di laurea di [Aquino, 2011] e sono evidenti nel nostro contesto, in cui la rappresentazione dei questionari mediante modelli semplifica di molto la progettazione e rende possibile l'interfacciamento del server da una moltitudine di client diversi.

Quanto detto finora può essere riassunto in questo modo:

Esiste un'unica componente server e un'unica componente di memorizzazione permanente dei dati ed esiste un'interfaccia di comunica-

[2]Tipicamente il metodo HTTP utilizzato per inviare dati dai client ai server è il POST. Il questionario compilato potrebbe quindi essere trasmesso al server in formato XML all'interno del *body* del metodo POST. In linea di principio però, nulla vieta di utilizzare il POST in modo classico, ovvero sostituendo l'XML con il formato in coppie *key-value* comunemente usato per l'invio dei form nel Web.

zione tra di esse. I vari client utilizzano HTTP/S come metodo di interrogazione del server e XML come formato di scambio dati col server.

L'interfaccia XML del sistema permette ai clienti di: 1) ricevere un questionario da compilare; 2) inviare un questionario compilato; 3) inviare la definizione di un nuovo questionario.

4.4 Requisiti delle basi di dati

Cominciamo la nostra analisi dei requisiti dalla componente di modellazione, salvataggio e recupero dei dati del sistema. Tale componente è di vitale importanza soprattutto per quanto concerne la modellazione dei metodi di misura dei dati `PRO`. Il database centrale deve essere in grado di memorizzare correttamente e coerentemente questionari di ogni tipo e con diverse caratteristiche. Alcuni questionari conterranno domande atte alla misurazione di un certo stato di salute, altri chiederanno dei dati precisi provenienti da strumenti di misurazione; altri ancora potrebbero voler misurare combinazioni diverse di caratteristiche cliniche dei pazienti. É importante capire che la generalità e flessibilità della rappresentazione in forma di dati dei questionari è per questo progetto di vitale importanza.

Un'altra cosa molto importante da sottolineare è l'uso che si farà di tali dati. Infatti un `PRO` tipicamente è atto a misurare una certa caratteristica clinica, come per esempio la salute generale del paziente, la sua qualità della vita, il dolore, lo stato d'ansia, la funzionalità, il fattore di rischio, ecc. Un `PRO`, quindi, deve produrre un certo *outcome*, un risultato, che dal punto di vista informatico non è altro che un numero. Il paziente riempie il questionario, effettuando delle scelte e fornendo dati e il sistema calcola il punteggio finale per la caratteristica che il questionario dato si prefigge di misurare. Tutte queste considerazioni dovranno guidare le scelte progettuali delle basi di dati. Cerchiamo adesso di raffinare ulteriormente questi concetti.

4.4.1 Rappresentazione dei questionari e dei loro risultati

L'obiettivo di questa sezione è di fornire una cornice per la progettazione del database per gli `ePRO` e infine di facilitarne l'uso e l'interazione col sistema. Da un punto di vista astratto un generico questionario è semplicemente un insieme di domande con annesse possibili risposte. Astraendo dai dettagli presentazionali, a ogni domanda l'utente può: a) effettuare una scelta di uno o più elementi da un dato insieme; oppure b) inserire manualmente una risposta[3]. Qualunque altra caratterizzazione dei tipi di domanda è puramente presentazionale (ad esempio se usare menu a scomparsa, bottoni, immagini o altro). Dal punto di vista informatico la scelta effettuata dall'utente può sempre esser vista come un numero tra 0 e $n-1$ se n è il numero delle scelte possibili oppure, nel caso di scelta multipla, come un sottoinsieme, possibilmente vuoto, $I \subseteq \{0, 1, \ldots, n-1\}$ delle possibile risposte.

Un `ePRO` deve inoltre misurare almeno una caratteristica. Deve cioè produrre uno o più *outcome* che, come abbiamo già detto, possono esser visti come semplici numeri[4]. Per esempio un questionario potrebbe misurare allo stesso tempo sia la qualità della vita del paziente che il suo stato di ansia. Tali questionari sono solitamente detti *multi-costrutto* e i loro outcome sono due o più numeri, uno per ogni caratteristica misurata. Il modello di rappresentazione dei dati deve quindi tenere in considerazione tale eventualità, prevedendo, in fase di definizione di un questionario, la scelta del numero di outcome del questionario. L'outcome verrà computato dal sistema alla fine della compilazione del questionario, sulla scorta delle risposte fornite dall'utente e in base a un algoritmo di calcolo apposito, uno diverso per ogni outcome calcolato. Può essere conveniente vedere un questionario

[3]Il secondo approccio deve esser reso disponibile per ragioni di completezza, ma lo sconsigliamo caldamente qualora non strettamente necessario, in quanto richiede logica ulteriore per i controlli, può facilmente portare a errore e non crediamo sia realmente indispensabile nella maggior parte dei casi. Ciononostante è il modello corretto qualora si utilizzassero strumenti di misurazione clinici, come misuratore di pressione, glicemia, ecc.

[4]Ovviamente è possibile prevedere outcome che restituiscano un punteggio in forma letterale, un punteggio sotto forma di stringhe o altro. Ciò non toglie che vi è un insito ordine in ogni possibile set di outcome e quindi una corrispondenza biunivoca tra n possibili outcome e i primi n numeri naturali.

come una funzione che prende in input delle scelte e produce in output uno o più outcome.

É importantissimo, infine, che ogni immissione dei dati da parte del paziente venga segnata con un *timestamp*, ovvero una marcatura oraria. Ciò rende un questionario in linea con i requisiti tipici dei diari `PRO`.

4.4.2 Rappresentazione dei dati utente

Per quanto concerne invece il sistema in generale, si richiede che il database possa memorizzare i dati degli account dei pazienti e dei dottori, tra cui informazioni di autenticazione al sistema, dati personali, ecc. Inoltre è necessario memorizzare i legami tra i pazienti e i dottori: un dottore può avere più pazienti sotto la propria ala, ai quali può somministrare `ePRO` e dei quali può controllarne l'andamento; un paziente può d'altronde essere seguito da più di un dottore. I legami così formati creano una cosiddetta rete sociale che, se aumentata anche a connessione dottore-dottore, permetterebbe la collaborazione tra medici attraverso il sistema.

Concludiamo questa parte dicendo che non vi sono particolari requisiti per quanto concerne le tecnologie da usare. É possibile usare database relazionali, così come database a oggetti, o ancora altro. É possibile usare un solo database fisico o delle basi di dati distribuite. Ciò che importa garantire sono: la sicurezza dei dati, l'affidabilità del sistema di gestione dei dati e l'accesso veloce e concorrente. Molti di questi requisiti sono discussi in sezione 4.7. Riassumiamo schematicamente quanto detto in questa sezione:

Un questionario `ePRO`:

É una lista ordinata di domande ed è atto a misurare una o più caratteristiche del paziente.

Una domanda può essere di due tipi:

1) a risposta chiusa: si sceglie un sottoinsieme, possibilmente vuoto, tra l'insieme delle possibili risposte;

2) a risposta aperta: l'utente inserisce manualmente un valore.

Una caratteristica misurata da un questionario ePRO:
Può essere vista come un numero calcolato sulla base delle scelte effettuate dal paziente nell'intero questionario.

Il database deve permettere la memorizzazione di:

- Questionari ePRO e loro risultati
- Dati degli utenti
- Relazioni tra gli utenti

4.5 Requisiti delle interfacce utente

Parte fondamentale del servizio offerto da $\mathbb{P}$.ROSE sono le interfacce utente. Discuteremo adesso i loro requisiti in quanto moltissimi dei requisiti funzionali di tutto il sistema sono collegati a essi o ne dipendono pesantemente.

Abbiamo già visto che il sistema sarà acceduto da tre tipi diversi di client, ovvero web browser desktop/laptop, web browser mobili e app. Ognuno di essi fornirà all'utente una diversa interfaccia, pensata apposta per le sue caratteristiche, tra cui dimensione dello schermo e metodo di navigazione tra pagine. Ad esempio un computer desktop non ha tipicamente problemi di dimensione dello schermo e viene navigato attraverso mouse e tastiera. D'altro canto un browser su cellulare risente delle dimensioni ridotte del monitor e del fatto che la navigazione avviene attraverso il touch-screen. Devono quindi essere sviluppate tre diverse interfacce utente:

UI-1 **Per browser desktop/laptop**: un'interfaccia dove lo spazio non è tipicamente un problema e la fruizione avviene tramite mouse e tastiera;

UI-2 **Per browser mobile**: pensata per schermi piccoli con navigazione via touch-screen;

UI-3 **Per app**: interfaccia simile alla precedente ma con un più ristretto insieme di funzionalità, dovuto al fatto che una app è fisicamente residente sul file system dell'utente e non dovrebbe quindi superare certe dimensioni.

4.5.1 Caratteristiche delle interfacce utente

In questa trattazione considereremo una UI come formata da varie sotto-interfacce, ovvero delle *view* per ognuna delle funzionalità del client. Nel seguito indicheremo quali sono le view principali e in quali interfacce, tra quelle sopra enunciate, devono essere rese disponibili.

Esecuzione di un questionario `ePRO`

La view per l'esecuzione di un questionario permette a un paziente di scorrere il questionario e rispondere alle varie domande. É un'interfaccia fondamentale, forse la più importante di tutto il sistema e presenta numerosissimi requisiti di usabilità (cfr. *Requisiti di usabilità* più avanti in questa stessa sezione). É necessario implementare *diramazioni logiche* nell'esecuzione di un questionario, ovvero diverse strade selezionate sulla base delle precedenti risposte fornite dall'utente.
◇ *Disponibile su tutte e tre le interfacce UI-1, UI-2 e UI-3.*

Creazione di un nuovo questionario `ePRO`

É una view utilizzabile solo dai medici per la definizione di nuovi strumenti di misurazione di dati `PRO`. Presenta anch'essa requisiti di usabilità, ma molto diversi da quelli del punto precedente. Deve permettere a un medico di inserire domande nel questionario e di decidere quanti sono gli outcome da misurare. Inoltre deve fornire un modo per la definizione degli algoritmi per il calcolo automatico degli outcome del questionario.
◇ *Disponibile soltanto sull'interfaccia UI-1.*

Somministrazione di un questionario ePRO

L'interfaccia deve permettere a un medico di poter somministrare un questionario già creato a un paziente o a un gruppo di pazienti e di programmare, se necessario, somministrazioni differite automatiche e cadenzate (per esempio "venerdì prossimo alle 15:30" o "ogni lunedì alle 14:00").
◇ *Disponibile sull'interfaccia UI-1 e in misura limitata sull'interfaccia UI-2 e UI-3.*

Visualizzazione risultati ePRO

Anch'essa utilizzabile solo da personale medico, mostra i risultati delle somministrazioni in varie forme, come ad esempio grafici e tabelle. Tale interfaccia deve permettere la visione di informazioni aggiuntive sugli studi effettuati, come ad esempio la percentuale di pazienti che hanno completato i questionari loro somministrati e i relativi risultati. Questa view può essere a sua volta suddivisa in altre sotto-view:

1. **Visione dei risultati di un singolo questionario** eseguito da un singolo paziente. In questa view è possibile visionare tutte le risposte date da un paziente in un certo questionario.

2. **Visione aggregata dei risultati di più somministrazioni**. Tale view permette di accedere alle statistiche sulle misurazioni effettuate in termini di percentuali di completamento dei questionari, medie, minimi, massimi dei risultati, ecc.

3. **Visione dei risultati di un singolo paziente** sulla scorta di tutti i questionari da lui effettuati, in tutti gli studi cui il dottore ha accesso[5].

◇ *Disponibile sull'interfaccia UI-1 e in misura limitata sull'interfaccia UI-2.*

[5]Possiamo dire che un dottore ha accesso a un certo studio se l'ha creato lui stesso o se è collegato in un qualche modo al dottore che l'ha creato.

4.5.2 Requisiti di usabilità

Al fine di determinare quali fossero i requisiti fondamentali di usabilità delle interfacce utente abbiamo utilizzato l'articolo *Electronic diaries and questionnaires: Designing user interfaces that are easy for all patients to use* [Palmblad and Tiplady, 2004] poiché raccoglie in modo ben organizzato un insieme di caratteristiche che i sistemi `ePRO` devono avere affinché possano raccogliere i dati dei pazienti nel modo più efficace possibile. Molti dei requisiti qui presentati sono stati tratti da questo articolo.

Considereremo in maniera diversa i problemi riguardanti le interfacce per i pazienti da quelli riguardanti le interfacce per i medici, in quanto i requisiti delle due classi di utenti sono parecchio differenti (cfr. 4.6.4 *Classi di utenti*). Il sistema dovrà essere usato da un vasto numero di pazienti, di numero presumibilmente maggiore rispetto al numero degli utenti medici. Inoltre i pazienti verranno selezionati con criteri clinici e non con criteri tecnologici, per tanto è necessario realizzare un sistema in grado di essere usato da qualsiasi tipologia di paziente. Se così non fosse, ovvero se il sistema non permettesse l'uso dei servizi a certe categorie di pazienti, i risultati di alcuni studi clinici potrebbero risentirne seriamente.

Principi base Gli autori dell'articolo menzionano tre principi da tenere a mente durante la progettazione di un sistema `ePRO`, ovvero il sistema deve essere: a) *scientifico*, evitando errori nella selezione della popolazione studiata e nelle risposte fornite; b) *funzionale*, permettendo rilevazioni da una vasta gamma di pazienti tra cui, per esempio, i più anziani, persone con gravi disabilità o i neofiti dei computer; e infine c) *etico*, assicurandosi che le procedure di rilevazione siano semplici e veloci. Terremo anche noi a mente questi principi nel resto di questa trattazione.

Sistema alla portata di tutti Spesso i trial clinici includono ampi e svariati campioni di popolazione e ciò può, ovviamente, includere pazienti di ogni età. Inoltre capita spesso che i pazienti più anziani non abbiano molta dimestichezza

con i dispositivi tecnologici e che quindi partano svantaggiati rispetto ai pazienti più giovani o con esperienza sull'uso dei computer e dei cellulari. É importantissimo che il sistema che vogliamo sviluppare sia, per così dire, alla portata di tutti i pazienti. Nell'atto di progettare le interfacce per i pazienti sarebbe meglio avere in mente, quindi, la categoria di pazienti che possa avere più problemi durante l'uso del sistema.

Training breve Un altro aspetto importante da considerare è il fatto che spessissimo in ambito clinico le possibilità di fornire un supporto guidato all'uso del software sia molto limitata, sia per motivi di tempo che di denaro. Un sistema che si dica usabile deve permettere anche a chi è più digiuno di esperienze pregresse nell'uso di apparecchiature informatiche di imparare a usarlo in breve tempo e senza particolari sforzi. Si stima, quindi, che un periodo massimo di 5 minuti debba essere sufficiente, nella maggior parte dei casi, a rendere i pazienti a proprio agio nell'uso del sistema.

No valori di default Un'ulteriore principio importantissimo concerne l'utilizzo o meno di valori di default nelle interfacce. Di solito essi vengono usati nei sistemi informatizzati per facilitare l'utilizzo delle interfacce. Nel caso degli `ePRO`, invece, l'utilizzo di valori di default potrebbe portare all'invio di dati non corretti. Si pensi, ad esempio, cosa accadrebbe se si utilizzassero come default valori inseriti in precedenti misurazioni. L'utente potrebbe dimenticare di cambiarli semplicemente perché il sistema permette di passare alle domande successive senza alcun vincolo di sorta. Per questo motivo è bene che i dati di input siano sempre il risultato di una qualche azione del paziente e mai provenienti da valori di default.

No menu a scomparsa o aiuti opzionali Spesso al fine di condensare molta informazione dentro un'unica interfaccia vengono usati menu a scomparsa e aiuti opzionali. In un sistema `ePRO` è necessario invece che tutte le opzioni disponibili siano visibili immediatamente al paziente, altrimenti alcune di esse potrebbero venire casualmente ignorate dall'utente, semplicemente perché non subito disponibili. Ogni informazione di aiuto è inutile se non immediatamente visibile sul-

lo schermo, sia per quanto riguarda informazioni relative alle domande, sia per quanto riguarda le opzioni di scelta. Un'altra cosa da evitare sono le barre di scorrimento laterali, in quanto spezzano il contenuto e non sono usabili da utenti poco esperti.

Necessità di testare il sistema su un campione valido Non è possibile assumere che il sistema sia adeguato a ogni tipo di paziente senza effettuare dei test. É quindi di vitale importanza effettuare dei test iniziali e ricevere il feedback da parte di un variegato campione di pazienti. Tuttavia non è necessario testare ogni singola funzionalità del sistema: basta concentrarsi principalmente sull'esecuzione dei questionari e i problemi a essa legate.

Grandezza dei display e dei font Alcuni dei requisiti finora espressi sono adeguatamente soddisfatti dai dispositivi touch-screen, come ad esempio i tablet o gli smartphone. I dispositivi touch-screen si sono dimostrati particolarmente adatti all'uso estensivo nei trial clinici, dove spesso hanno riscontrato i gusti e le preferenze dei pazienti anche quando questi erano totalmente digiuni di esperienza nell'uso dei computer [Allenby et al., 2002, Eun-Hyun and Lee, 2009, Salaffi et al., 2009]. I tablet, inoltre, hanno il vantaggio di avere schermi particolarmente grandi e di essere molto leggeri e sono quindi, a nostro parare, una delle scelte ottimali per l'uso degli `ePRO`. D'altro canto reputiamo sia necessario garantire l'utilizzo a una più vasta utenza e per ottenere ciò è necessario permettere l'accesso al sistema mediante smartphone touch-screen i cui display difficilmente superano i 4 pollici. Per permettere la fruizione del sistema su tali supporti è necessario utilizzare una grandezza dei font di almeno 12 punti. Dal momento che il sistema sarà disponibile in più lingue, è bene anche ricordare che alcune lingue sono più verbose di altre e sarebbe quindi buona norma lasciare circa un 40% di spazio libero in più quando si creano i questionari.

Colori e simboli speciali Circa il 10% della popolazione maschile soffre di daltonismo. É quindi importante evitare di utilizzare i colori, specie i rossi e i verdi, per enfatizzare informazioni importanti e utili ai fini della compilazione dei

Figura 4.4: Esempio di input numerico in forma di selettore e slider.

questionari. Simboli speciali come il check-mark (✓) dovrebbero essere evitati in quanto hanno significati diversi in alcune culture.

Domande a scelta singola e multipla É molto comune nell'ambito del Web utilizzare i *radio button* per i form a scelta singola. Nel caso degli `ePRO`, invece, è bene evitare il loro utilizzo in quanto non vi è abbastanza informazione visiva sul fatto che i cerchietti sono parte del controllo e non dell'interfaccia grafica. É meglio utilizzare al loro posto dei bottoni con una grafica tipica degli oggetti cliccabili che siano anche più facili da premere dei classici cerchietti rotondi. Nel caso di un form a scelta multipla è possibile utilizzare un'interfaccia che mostri tutte le possibili scelte in forma di bottoni, a cui aggiungere un ultimo bottone con la dicitura "Nessuno dei precedenti". Ogni pressione su bottoni già selezionati li deseleziona e la pressione dell'ultimo bottone li deseleziona tutti (lui compreso).

Domande a input numerico Quando si parla di input numerici bisogna fare una distinzione: da un lato vi sono gli input numerici guidati dall'interfaccia, in cui è possibile scegliere un numero tra quelli presentati a video per mezzo di appositi gadget visuali; dall'altro vi sono gli input numerici liberi in cui l'utente è libero di comporre il numero cifra per cifra, per mezzo di un tastierino numerico (o la tastiera del computer). Il primo dei due può essere implementato mediante selettore e/o *slider*, come mostrato in Figura 4.4 (esempio preso dalla libreria di funzioni Javascript *JQuery Mobile* [Jquerymobile.com, 2011]). Per quanto concerne il secondo tipo di input, quello libero, è bene utilizzare un tastierino numerico che contenga, oltre ai numeri da 0 a 9, il tasto "Clr" per la cancellazione del numero finora inserito e il tasto "Del" per la cancellazione dell'ultima cifra inserita. Inoltre, se il numero atteso è un numero decimale bisogna includere anche il tasto che inserisce la virgola (o il punto, a seconda dell'uso nella lingua dell'utente). Ad ogni modo è bene impostare un'interfaccia grande e che si adatti

Figura 4.5: Esempio di input numerico in forma di scala analogica visuale.

bene allo schermo, evitando in ogni modo di inserire tasti che l'utente non debba potere usare nell'input. Nonostante i due tipi di input siano entrambi disponibili, è buona norma limitare l'utilizzo di quelli numerici liberi ai soli casi in cui essi siano realmente indispensabili. Se per esempio l'input numerico che ci si aspetta è di tipo intero e si trova in un range ben preciso è meglio utilizzare la prima forma di input, per mezzo di selettore e slider.

Una leggera variante dello slider che è conveniente menzionare è la cosiddetta *scala analogica visuale* [Hopper et al., 1996], ovvero uno slider i cui estremi sono etichettati con due stringhe di testo, come mostrato in Figura 4.5. Il paziente lo usa selezionando un punto interno all'intervallo, che meglio approssimi quanto si senta vicino all'uno o all'altro estremo.

Messaggi É bene inviare messaggi di successo o di errore all'utente soltanto quando realmente necessario. Consideriamo necessarie situazioni come quelle in cui l'utente non ha inserito un valore corretto (per esempio un numero fuori range, una email non valida, ecc.) oppure una spedizione dei dati è avvenuta con successo nel caso in cui c'erano buone ragioni per credere che ciò potesse non accadere (ad esempio se si stanno spedendo dati attraverso una rete). Molti degli errori di input possono essere evitati utilizzando domande a scelta multipla. Ad ogni modo, quando è necessario inviare un messaggio all'utente è bene includere informazioni quali *il motivo* del messaggio e *cosa* l'utente dovrebbe fare per andare avanti, evitando di includere dati tecnici relativi agli eventuali errori del sistema.

Navigazione La navigazione tra le domande di un questionario deve avvenire una domanda alla volta, una schermata a domanda. Come abbiamo già detto ogni schermata deve contenere tutte le informazioni necessarie, subito visibili, comprese tutte le opzioni disponibili per la domanda e tutte le informazioni di

aiuto. Per passare alla domanda successiva l'utente deve effettuare una scelta o inserire un input. Per garantire ciò, l'interfaccia deve nascondere il pulsante da utilizzare per muoversi alla domanda successiva finché non sia stato ricevuto l'input dall'utente e finché tale input non sia stato verificato (ad esempio che un numero sia entro il range stabilito). Nei rari casi in cui sia permesso saltare una domanda, tale pulsante può esser reso immediatamente disponibile. L'interfaccia, d'altronde, deve permettere all'utente di tornare indietro sui suoi passi e rivisitare le domande a cui ha già dato risposta e, possibilmente, modificarle.

É molto importante permettere la definizione di questionari in cui possano avvenire diramazioni logiche, ovvero in cui siano presenti domande a doppia scelta la cui domanda successiva è calcolata in fase di esecuzione, in base alla scelta effettuata dal paziente. É importante però che l'utente sia tenuto all'oscuro di tali biforcazioni e che veda soltanto la sequenza lineare delle domande. É anche utile mostrare all'utente un indicatore di stato di avanzamento dell'intero questionario. Ciò è utile specialmente nei questionari più lunghi.

Inoltre è consigliato inserire un riepilogo finale, compatto e minimale, che mostri tutte le scelte fatte e gli input forniti nel questionario. Infine deve essere mostrato uno speciale pulsante per l'invio dei dati al server o il salvataggio degli stessi nella memoria locale del client.

Creazione dei questionari I dottori che avranno a disposizione una speciale interfaccia per la creazione di nuovi questionari. Tale interfaccia è bene che sia anch'essa molto semplice e intuitiva e che riduca al minimo le scelte necessarie per ottenere un sistema di domande sicuro e facile per l'utente. É bene quindi che sia il sistema stesso a prendersi l'onere di decidere gli aspetti presentazionali dell'interfaccia per i pazienti, lasciando ai dottori il compito di decidere soltanto il contenuto delle domande. La fase di creazione deve permettere al medico di scegliere il tipo di domanda, i possibili range di validità degli input e di aggiungere eventuali immagini. Inoltre deve essere fornito un modo semplice per impostare il calcolo finale del punteggio. Possono essere rese disponibili, a tal scopo, una serie di funzioni built-in come "somma delle domande completate", "somma,

media, minimo o massimo di certi valori di input numerico", ecc. Inoltre, in ogni momento della creazione, il medico deve essere in grado di lanciare una simulazione del questionario, per constatare in prima persona il risultato finale dello stesso.

I questionari creati, inoltre, andranno a far parte di un insieme di questionari visibili da tutti gli altri medici iscritti al sistema, in modo tale che quando un medico vuole somministrare un certo questionario a un suo paziente abbia buone probabilità di trovare un modello già pronto ai suoi scopi (o che richieda un minimo numero di modifiche).

4.6 Requisiti funzionali

Riprendiamo quanto detto finora per stilare una lista dettagliata degli aspetti funzionali che l'applicazione che stiamo progettando deve garantire. Al fine di permettere una buona organizzazione e una facile individuazione dei requisiti nelle successive fasi di progettazione e implementazione, ogni requisito verrà indicato con una lettera in maiuscolo e un numero. La lettera maiuscola indica la classe del requisito, il numero crescente serve a individuare un requisito ben preciso all'interno della sua classe. Le lettere relative alle classi dei requisiti sono spiegati nella seguente tabella:

S	B	A	U	M	P
Server	Browser	App	Utente	Medico	Paziente

in modo tale che, ad esempio, B2 indichi il secondo requisito all'interno della classe dei requisiti dei browser web.

4.6.1 Server web

Il server web deve:

S1. **Accettare connessioni HTTP e HTTPS** sul protocollo TCP/IP senza distinzione di provenienza e garantendo accesso concorrente. A tal proposito

sarebbe bene porre attenzione ai problemi di scalabilità [Hayes, 2008], in quanto il sistema potrebbe presumibilmente diventare molto usato e in breve tempo accogliere richieste di svariate centinaia o migliaia di utenti.

S2. Essere in grado di **discernere richieste Ajax da richieste non-Ajax** e quindi potersi comportare in maniera differente a seconda dei due casi.

S3. Essere in grado di **discernere il tipo di dispositivo client che ha effettuato la richiesta** e quindi potersi comportare in modo adeguato a seconda che sia un computer desktop, un cellulare, un tablet o altro. In particolare il server deve potere discernere se la richiesta HTTP o HTTPS proviene da un web browser su computer desktop o da un web browser su dispositivo mobile o da app. O ancora da dispositivo client di altro tipo (come ad esempio tramite il software *curl* o tramite un programma che fa uso diretto di primitive di connessione TCP/IP).

S4. **Essere il più possibile conforme all'architettura REST** [Fielding and Taylor, 2002], con particolare attenzione alla rappresentazione delle risorse tramite URI, alla loro classificazione in *collezioni* ed *elementi* e alla disponibilità dei metodi d'accesso alle risorse quali GET, PUT, POST e DELETE.

In particolare bisogna introdurre almeno le seguenti risorse:

S4.a *Questionari `PRO` "vuoti"*, ovvero contenenti soltanto le domande, senza le risposte date dai pazienti.

S4.b *Risultati dei questionari* inviati dai pazienti.

Tali risorse devono poter essere accedute in almeno due modi:

S4.c *Collezioni* di questionari `PRO` e dei risultati delle misurazioni.

S4.d *Singoli* questionari `PRO` e singoli risultati.

S5. **Implementare politiche di autenticazione e accesso alle risorse** memorizzate nel sistema. In particolare si chiede che il server possa gestire

l'autenticazione di due classi distinte di utenti: pazienti e dottori (cfr. 4.6.4 *Classi di utenti*).

Inoltre il sistema deve permettere l'autenticazione degli utenti mediante credenziali preesistenti, come ad esempio credenziali di posta elettronica che i pazienti e i medici già posseggono e/o mediante lo standard di autenticazione *OpenID* [Recordon and Reed, 2006].

S6. **Controllare meticolosamente ogni richiesta HTTP e HTTPS e ogni input fornito**, senza escludere: risorse (URI) richieste, credenziali di accesso, tipo e numero degli input forniti.

S7. **Gestire i casi di errore in modo accurato**, con particolare attenzione all'usabilità. Ciò implica, per esempio, la necessità di fornire messaggi di errore adeguati al tipo di utente e significativi, ovvero privi di dettagli tecnici (cfr. 4.5.2 *Requisiti di usabilità*).

S8. **Consentire l'utilizzo dei *cookie*** per il recupero dello stato dei client che si connettono al server.

S9. **Garantire accesso pubblico diretto a file di tipo presentazionale** come immagini di layout, fogli di stile css e codice eseguibile lato client.

S10. **Proibire accesso pubblico diretto a file privati** come immagini personali degli utenti, codice lato server, ecc.

S11. **Poter comunicare coi client in formato XML**, definendo un preciso formato per ogni input o output gestibile.

S12. **Fornire pagine o frammenti di pagine in HTML**, che siano il più possibile **prive di aspetti presentazionali**. Ciò significa che le pagine o i frammenti inviati contengano quasi esclusivamente contenuti e marcature dei contenuti e che le informazioni grafiche siano contenute principalmente nei fogli di stile.

4.6.2 Browser web

Uno dei metodi di accesso al server e alle sue funzionalità è per mezzo di un web browser. Come abbiamo già stabilito il server deve essere in grado di capire quando una richiesta HTTP o HTTPS proviene da tale categoria di client. In particolare, il punto S3 dei requisiti funzionali del server prevede che il server possa capire quando il richiedente è un browser su desktop o un browser su mobile. Il server invia al web browser pagine o frammenti di pagine HTML, immagini, fogli di stile e codice eseguibile lato client, ma essi avranno delle caratteristiche diverse a seconda del dispositivo al quale sono destinati. Di seguito delineiamo le specifiche funzionali del software lato client evidenziando, quando necessario, le caratteristiche che differiscono sui due tipi di dispositivi client.

HTML, fogli di stile e codice eseguibile lato client devono:

B1. **Definire (almeno) due tipi di interfacce**: un'interfaccia per la fruizione del servizio da browser desktop e un'interfaccia per la fruizione del servizio da browser mobili. Le due interfacce devono essere progettate per essere adeguate alle dimensioni tipiche degli schermi sui quali verranno visualizzati e al tipico metodo di navigazione e fruizione delle pagine. Bisogna quindi tenere conto che le pagine accedute da smartphone o tablet verranno visualizzate su schermi di dimensione ridotta e navigati via touch-screen. Fare riferimento alla sezione 4.5 *Requisiti delle interfacce utente* per tutti i dettagli.

Il codice eseguibile lato client deve:

B2. **Essere eseguito correttamente e uniformemente sui più comuni browser** per desktop e mobile, senza escludere le ultime versioni di: Microsoft Internet Explorer, Mozilla Firefox, Google Chrome, Apple Safari, Opera per quanto riguarda i browser desktop [W3Schools, 2011] e Google Android, Apple Safari e Mozilla Firefox for mobile per quanto riguarda i browser mobili.

B3. **Permettere comunicazioni Ajax col server**, in modo da garantire la realizzazione di interfacce web altamente interattive [Garrett, 2005]. In particolare sarebbe auspicabile se si potessero utilizzare i metodi HTTP sulle risorse sia mediante richieste via location bar del browser, sia mediante chiamate Ajax, senza che le risorse debbano avere nomi diversi[6].

B4. **Lavorare insieme ai fogli di stile** per permettere funzionalità presentazionali avanzate, le quali potranno essere diverse a seconda del tipo di interfaccia (cfr. requisito B1). Tali funzionalità presentazionali saranno anche alla base dei requisiti di usabilità del sistema (cfr. 4.5.2 *Requisiti di usabilità*).

4.6.3 App

Col termine *app* indichiamo un'applicazione sviluppata per dispositivi mobili quali smartphone o tablet e che possa essere lanciata direttamente sui loro sistemi operativi[7]. Il software in versione app è alla base dei requisiti di usabilità del sistema ℙ.ROse in quanto permette la realizzazione di interfacce altamente facili da usare per la stragrande maggioranza degli utenti grazie alle tecnologie touchscreen e permette una fruizione estesa dei servizi grazie all'utilizzo di dispositivi estremamente mobili e relativamente poco costosi. Il servizi che ℙ.ROse vuole mettere a disposizione sulle app sono per lo più focalizzati all'esecuzione dei questionari PRO da parte dei pazienti. Non è necessario infatti includere funzionalità quali la creazione dei questionari o la visualizzazione (avanzata) dei risultati.

Una app deve:

A1. **Funzionare (almeno) nei sistemi operativi Apple iOS e Google Android**. Ciò implica, in linea di principio, la codifica di (almeno) due app diverse, oppure il porting di una app da un sistema a un altro.

[6] Si ammette, ovviamente, che le due risposte possano differire per parte del contenuto: infatti è ammissibile che una risposta a una chiamata Ajax contenga soltanto una parte dell'HTML necessario alla pagina; una risposta a una chiamata non-Ajax conterrebbe invece il codice HTML dell'intera pagina.

[7] Per completezza ricordiamo al lettore che sistemi operativi diversi richiedono, in linea di principio, app scritte in linguaggi di programmazione diversi.

A2. **Avere un'interfaccia adeguata all'utilizzo sui dispositivi mobili**, con particolare attenzione alle dimensioni dei dispositivi e all'utilizzo del touch-screen come mezzo di navigazione (cfr. 4.5 *Requisiti delle interfacce utente*).

A3. **Permettere a pazienti e dottori di accedere ai loro profili online** e quindi alle loro informazioni personali.

A4. **Permettere ai pazienti di visualizzare una lista dei questionari ancora da effettuare**, mostrando loro informazioni quali il medico che l'ha somministrato e la scadenza. La lista deve inoltre contenere soltanto i questionari eseguibili nel preciso momento in cui viene generata (alcuni questionari hanno validità in un certo intervallo di tempo).

A5. Permettere ai medici di **visualizzare la lista dei loro pazienti**, di visualizzare **la lista dei questionari disponibili** e di **somministrare un questionario** a un loro paziente.

A6. **Permettere ai medici di eseguire un questionario per conto di un paziente**.

A7. **Permettere ai pazienti di eseguire un questionario**.

A8. **Poter memorizzare in locale i risultati di un questionario compilato da un paziente** nel caso la connessione salti durante la compilazione, per poi inviarlo al server alla prossima riconnessione.

4.6.4 Classi di utenti

Come abbiamo già accennato precedentemente $\mathbb{P}$.ROSE è pensato per fornire servizi a due tipi di utenti che, per semplicità, indicheremo come *pazienti* e *medici*[8]. In questa sezione specificheremo i servizi che il sistema deve fornire agli uni

[8]In realtà la distinzione in questi termini non è del tutto corretta. Infatti i questionari PRO possono essere somministrati anche a persone senza particolari patologie e non necessariamente da medici, ma da personale sanitario di aiuto ai medici, come ad esempio gli infermieri. Ad ogni modo utilizziamo la distinzione in pazienti e medici per comodità.

e agli altri, in risposta alle azioni che queste due classi di utenti sono abilitate a compiere.

Alcune funzionalità sono disponibili per entrambe le classi. Il software deve permettere agli utenti in generale di:

U1. **Registrarsi al servizio**. La procedura di registrazione deve tener conto del tipo di utente. É possibile che le informazioni richieste per le due classi di utenti differiscano. Ad ogni modo è necessario implementare una procedura di registrazione che risponda ai requisiti di usabilità di cui in sezione 4.5.2. Inoltre devono essere implementate misure di sicurezza all'atto della registrazione (cfr. sezione 4.7.1 *Sicurezza*) e la possibilità di registrarsi mediante credenziali già in possesso dell'utente (cfr. requisito S5).

U2. **Loggarsi al servizio**, una volta registrati e possibilmente mediante l'utilizzo di credenziali di accesso preesistenti (cfr. requisito S5). Inoltre, come già espresso in precedenza, client e server possono usare i cookie per mantenere le informazioni di login degli utenti (cfr. requisito S8).

U3. **Visualizzare e modificare le informazioni personali**, come per esempio nome, cognome, codice fiscale, indirizzo, ecc.

Il software deve permettere a un medico di:

M1. **Registrare un paziente**, includendolo tra i pazienti da lui seguiti. Tale procedura deve prevedere l'accettazione della richiesta da parte del paziente (cfr. sezione 4.7.1 *Sicurezza*).

M2. **Includere tra i propri pazienti utenti già registrati**, solo a seguito di richiesta da parte dei pazienti stessi. In nessun caso deve essere fornita una lista dei pazienti già registrati nel sistema (cfr. sezione 4.7.1 *Sicurezza*).

M3. **Visualizzare la lista dei propri pazienti**. Questa, per esempio, potrebbe essere una risorsa di tipo *collezione* a cui associare una URI.

M4. **Visualizzare il profilo di ogni suo paziente**, nel quale siano presenti tutti i suoi dati personali nonché i questionari e i relativi risultati `PRO`

che ogni paziente ha ricevuto da quel medico. Devono rimanere nascosti i questionari e i risultati PRO relativi ad altri medici.

M5. **Somministrare un PRO a un proprio paziente**. Il questionario o diario da assegnare deve essere stato precedentemente creato e, ovviamente, il medico deve avere i permessi di accedervi.

M6. **Lanciare l'esecuzione di un test PRO per conto di un paziente**, nel caso in cui il paziente si trovi in visita dal dottore. L'interfaccia relativa all'esecuzione di un test PRO in questa modalità deve essere identica a quella richiamata direttamente dal paziente (si veda 4.5 *Requisiti delle interfacce utente*) e prevedere un meccanismo di autenticazione del paziente (cfr. 4.7.1 *Sicurezza*).

M7. **Visualizzare i risultati di una serie di misurazioni** effettuate da un paziente o da più pazienti, in vari formati e con delle precise interfacce di visualizzazione (cfr. sezione 4.5 *Requisiti delle interfacce utente*).

M8. **Creare un nuovo strumento di misurazione PRO**. Il processo di creazione deve possedere una determinata interfaccia e precise caratteristiche di usabilità (cfr. sezione 4.5 *Requisiti delle interfacce utente*).

M9. **Lanciare una demo di un questionario PRO in creazione o già creato**. La demo deve presentare un'interfaccia uguale a quella dell'effettiva esecuzione del test, ma il sistema non deve permettere né l'invio al server né il salvataggio in locale di alcuna informazione ivi inserita.

M10. In previsione è possibile disegnare un sistema che possa facilmente essere ampliato per **permettere a medici di creare una rete di scambio di dati clinici**, utile in certi ambiti e contesti. Ad esempio un gruppo di medici e ricercatori che lavorano a un trial clinico potrebbero ricevere e analizzare i dati PRO di diversi pazienti e poi condividerli facilmente attraverso il sistema.

Il software deve permettere a un paziente di:

P1. **Visualizzare la lista dei propri dottori**.

P2. **Visualizzare il profilo pubblico di ogni suo dottore**.

P3. **Visualizzare la lista dei questionari ancora da effettuare**.

P4. **Lanciare l'esecuzione di un questionario** che gli sia stato somministrato, con un'interfaccia che rispecchi appieno tutti i requisiti di usabilità precedentemente discussi (si veda 4.5 *Requisiti delle interfacce utente*).

4.7 Qualità del software

Nell'ultima parte di questo capitolo discutiamo quali sono le qualità del software che vogliamo realizzare. Premesso che è sempre auspicabile che un software contenga tutte le "buone" qualità esistenti, ciò non è sempre un obiettivo facile da raggiungere. In realtà le varie caratteristiche del software spesso collidono le une con le altre, per cui è necessario stabilire, almeno in linea di massima, quali sono quelle di primaria importanza e quali sono, invece, quelle meno fondamentali.

Il sistema, è stato detto più volte, deve essere **sicuro**. I dati che si accinge a contenere sono molto riservati e nessun tipo di breccia nel sistema può essere considerata accettabile, nemmeno la più piccola. Parleremo poco più avanti dei problemi legati alla sicurezza, in un paragrafo a sé stante.

Un altro requisito fondamentale è la **manutenibilità**. Il software verrà sperabilmente utilizzato da centinaia se non migliaia di utenti e, come ogni sistema di grandi dimensioni, destinato al grande pubblico, necessiterà di costanti aggiornamenti, ampliamenti, aggiustamenti, ecc. La progettazione deve quindi tenere conto di tale caratteristica e, in un certo senso, questa dissertazione gioca un ruolo importante anche per quanto concerne il tema della manutenibilità.

Il software deve essere **affidabile** e deve essere **corretto**. Tantissimi utenti faranno uso del sistema da noi progettato e in particolare moltissimi saranno i pazienti che utilizzeranno il software per l'invio di dati sensibili. É necessario che essi percepiscano un certo grado di fiducia nel sistema, che è possibile ottenere soltanto attraverso la realizzazione di un sistema affidabile e corretto. Non sono

ammissibili situazioni in cui i dati vengano manomessi a causa di un malfunzionamento logico dei programmi. É anche quindi importante che le funzionalità più critiche vengano testate e verificate in maniera adeguata. Anche la **robustezza** è un fattore molto importante, in quanto la sua mancanza è spesso causa di malfunzionamenti che ne minano la affidabilità e la correttezza.

Le interfacce utente, specialmente quelle destinate ai pazienti, devono essere **usabili**. Di questo abbiamo ampiamente discusso nelle sezioni precedenti. Inoltre il sistema deve essere altamente **portabile**. Come abbiamo visto, infatti, le sue funzionalità dovranno essere accessibili su una molteplicità di dispositivi, operanti con tecnologie differenti.

Il software deve garantire **disponibilità** del servizio. Non è accettabile che si incorra in situazioni in cui il servizio risulti non disponibile in maniera del tutto improvvisa o, ancora peggio, per un consistente intervallo di tempo. L'utilizzo del sistema è previsto essere distribuito in maniera costante nell'arco della giornata e l'impossibilità di utilizzo farebbe di certo perdere la fiducia dei suoi utenti verso esso stesso. Come vedremo nel capitolo successivo, questo requisito purtroppo contrasta in parte con i requisiti di sicurezza. Sarebbe inoltre auspicabile che il sistema possa garantire anche un buon grado di **scalabilità** ma anche in questo caso le soluzioni al momento disponibili presentano alcuni problemi.

Il sistema deve permettere la **riusabilità**, entro certi limiti, di sistemi software client per la gestione di `ePRO`, come potrebbe ad esempio essere il sistema ProClara di cui abbiamo precedentemente discusso.

4.7.1 Sicurezza

I temi legati alla sicurezza coprono uno spettro vastissimo dell'Informatica e spesso non sono confinati soltanto a essa. Sarebbe quindi insensato pretendere che questa trattazione possa trattarli tutti in maniera esaustiva. La sicurezza web è un tema altrettanto enorme. In questa sezione daremo alcune linee guida per la progettazione di un sistema web che sia il più possibile consono ai canoni moderni di sicurezza nel web. Vi sono da un lato requisiti di robustezza di fronte ai più comuni attacchi web e dall'altro dei requisiti funzionali che garantiscano

una maggiore sicurezza dei dati immagazzinati nel sistema. Spesso, in questi casi, tali aspetti sono più che altro dei protocolli che è bene che il sistema rispetti, al fine di garantire la sicurezza in tutte le sue accezioni possibili. Molti dei requisiti qui presentati, quindi, trattano di sotto-temi della sicurezza, come ad esempio la protezione, la confidenzialità e l'integrità dei dati o ancora temi legati alla sicurezza dei programmi o delle comunicazioni.

Tipici attacchi web

É necessario che il server sia il più possibile invulnerabile ai più famosi attacchi web. Qui di seguito ne citiamo alcuni: SQL injection, Cross-site Scripting (XSS), Cross-site Request Forgery (CSRF) (che comprende attacchi come man-in-the-middle, Session forging, Cookie forging, Session fixation, Session poisoning, ...), E-mail header injection, Directory traversal, Exposed error messages.

Registrazione dei pazienti

Un tema delicato riguarda la registrazione dei pazienti nel sistema. Il sistema conterrà dati dei pazienti in quantità massivamente maggiore rispetto ai dati dei dottori. Inoltre la nostra filosofia è quella di vedere i pazienti come la parte più delicata degli utenti del sistema. Se da un lato qualunque medico può far valere i propri diritti senza particolare fatica, alcuni tipi di pazienti, specie quelli più anziani, hanno tipicamente meno possibilità di seguire da vicino certi problemi e rischiano spesso di veder lesi i propri diritti senza poter controbattere. É quindi necessario che si implementino dei meccanismi che proteggano in primis i pazienti. Uno di essi è legato alla procedura di registrazione al servizio.

Il modo più innocuo di registrarsi al servizio è farlo autonomamente. É permesso farlo sia ai dottori che per ai pazienti e non presenta particolari problemi se vengono utilizzate le comuni procedure di registrazione ai siti web (email e link di conferma). Per motivi logistici è inoltre necessario consentire ai dottori di registrare dei nuovi pazienti, qualora essi non posseggano già un account. In tale procedura è bene garantire ai pazienti la possibilità di declinare l'invito a unirsi al sistema, prima ancora che comincino a ricevere somministrazioni e/o

comunicazioni da parte del dottore. Un dottore che volesse inserire un nuovo paziente sotto la propria ala avrà il solo potere di "invitare" il paziente stesso. La decisione finale spetterà quindi al paziente.

Aggiunta di pazienti già registrati

Un caso molto simile al precedente è quando un dottore vuole aggiungere sotto la propria ala un paziente già registrato al sistema. Sarà necessario che il dottore conosca già l'indirizzo email del paziente. Il sistema provvederà all'invio di una email di "invito" al paziente e, come prima, sarà richiesta la sua approvazione. In nessun caso deve essere fornita una lista dei pazienti presenti nel sistema a dottori non già collegati con essi. In generale non deve essere presente alcuna lista che contenga "tutti i pazienti del sistema". I dottori otterranno gli indirizzi email dei pazienti che vorranno aggiungere chiedendoli direttamente a loro.

Esecuzione dei questionari

Sempre nello stesso tema si pone il problema dell'esecuzione dei questionari su dispositivi non di proprietà del paziente. Tale situazione è utile, per esempio, nei casi in cui il paziente si rechi in clinica per una visita e il dottore voglia somministrargli in loco un questionario. In tal caso il medico utilizzerà un suo dispositivo elettronico da consegnare al paziente per il tempo necessario all'esecuzione del test. Prima di consegnare il dispositivo al paziente il medico selezionerà il test da eseguire e ne lancerà l'interfaccia di esecuzione. É fondamentale, ai fini della sicurezza, che il sistema a questo punto richieda i dati di autenticazione del paziente. Ciò serve a garantire che il test venga eseguito soltanto dal paziente corretto. Potrebbe ad esempio succedere che, per errore, il medico abbia consegnato il tablet al paziente sbagliato!

Sicurezza a livello di codice

In ultima analisi si richiede che il codice eseguibile lato client non contenga dati o informazioni sensibili legati al sistema di $\mathbb{P}$.ROse, che ne possano mettere a

nudo eventuali caratteristiche interne che possano essere sfruttate per commettere attacchi mirati al sistema. Nella scelta di cosa implementare sul lato server e cosa sul lato client, è bene quindi tenere a mente questo importante principio.

Capitolo 5

Scelte progettuali

Nel capitolo precedente abbiamo presentato in maniera dettagliata i requisiti che il sistema di gestione degli `ePRO` che ci prefiggiamo di realizzare deve soddisfare. Essi spaziano dai requisiti architetturali del sistema nel suo complesso ai requisiti funzionali per ogni componente principale che lo caratterizza deve garantire, per arrivare infine a una serie di requisiti qualitativi del software che ne garantiscano certe proprietà, quali ad esempio la sicurezza e la manutenibilità. Il lavoro fatto nel precedente capitolo deve essere ampliato e portato a un livello di dettaglio maggiore, nel quale vengano fatte delle scelte progettuali ben precise, al fine ultimo di lavorare alla effettiva scrittura del software. I requisiti hanno due finalità in questa fase: da un lato servono a guidare le scelte progettuali verso la realizzazione di un sistema in linea coi bisogni dell'utenza finale, dall'altro servono a limitare le scelte di progettazione escludendo ogni altra possibilità non prevista.

In questo capitolo delineeremo le caratteristiche del software che si vuole realizzare, adottando una strategia di progettazione *top-down*, ovvero procedendo a una modularizzazione del sistema con granularità via via più fine, via via meno astratta. Alla fine di questo capitolo mostreremo alcuni dei possibili ampliamenti futuri del software e come potrebbero venire implementati.

La visione più astratta del sistema è esattamente quella prevista in 4.3 *Architettura del sistema e interfacce*, nella quale il sistema è presentato come un

insieme di tre componenti: un database, un server e un serie di client. In quella visione, come abbiamo mostrato, esistono due interfacce tra le componenti: un'interfaccia di scambio dati tra server e gestore del database e un'interfaccia di richiesta e fruizione dei servizi tra il server e i suoi client. Nel seguito di questa trattazione utilizzeremo questa macro-divisione e andremo a delineare in maggior dettaglio ognuna delle sue componenti.

5.1 Interfacce XML

Nel paragrafo 4.3.1 *Interfacce software e di comunicazione* dei requisiti abbiamo introdotto la necessità per il sistema di potere comunicare con i client mediante scambio dati in formato XML. Ciò permetterà in futuro di poter riutilizzare client `ePRO` già esistenti, mettendoli in comunicazione diretta col server di ℙ.`ROse`. O, alternativamente, ciò permetterà ad altri client di utilizzare ℙ.`ROse` per la memorizzazione persistente e sicura dei dati `ePRO`. Inoltre, consentirà una semplificazione notevole della realizzazione del software di gestione dei dati, in quanto rende possibile la definizione di concetti astratti come i questionari secondo la metodologia di progettazione a modelli, nota in letteratura con nome di Model Driven Architecture (cui abbiamo precedentemente accennato).

Abbiamo stabilito che i client debbano potere inviare e ricevere dati XML da e verso il server in due precisi contesti: per la creazione di un nuovo questionario (da parte di un dottore) e per la compilazione di un questionario somministrato (da parte di un paziente). Per ottenere ciò è necessario, quindi, stabilire i formati XML per lo scambio di dati. Nei contesti descritti sono necessari tre tipi diversi di formati XML, ovvero:

1. Un formato XML per l'invio dal client al server della struttura di un nuovo questionario appena creato.

2. Un formato XML per l'invio dal server al client della struttura di un questionario da compilare.

3. Un formato XML per l'invio dal client al server dei risultati della compilazione di un questionario.

Nel resto di questa sezione specificheremo nel dettaglio questi tre formati.

5.1.1 XML per nuovo questionario

Le minime informazioni necessarie per la creazione di un nuovo questionario sono: il titolo del questionario e una sua descrizione informale; informazioni che il paziente leggerà sulla prima pagina del questionario, prima di procedere alla sua compilazione; una serie di domande e possibili opzioni; il/i risultato/i `ePRO` che il questionario si prefigge di calcolare e il/i metodi per calcolarli. Per ogni domanda, infine, è necessario specificare: il tipo di domanda (se opzionale, aperta, ecc.); la stringa testuale che il paziente leggerà; se la domanda prevede un input diretto, il range di valori accettabili; se è una domanda opzionale è necessario anche fornire una serie di opzioni tra quali è possibile scegliere.

Per quanto riguarda il calcolo del punteggio assumiamo, come espresso nei requisti di usabilità, di potere utilizzare soltanto semplici funzioni come "il numero di risposte date dall'utente", "il numero di risposte A fornite", e funzioni simili, supponendo che esse siano già implementate nel server. Ci riferiremo quindi al calcolo del risultato per mezzo di una generica funzione $f()$, definita dall'utente sulla base delle funzioni basilari di questo tipo[1]. Questa semplificazione ci è utile ai fini della rappresentazione dei dati nelle prime versioni del progetto, ma potrà essere facilmente generalizzata e migliorata in futuro, come discusso in sezione 5.4 *Semplificazione del metodo di visualizzazione dei risultati*.

In Appendice B è possibile vedere un esempio di documento XML che fornisca al server la rappresentazione di un nuovo questionario contenente sei domande: una domanda a opzione singola con tre opzioni disponibili; una a opzione multipla con tre opzioni disponibili; una domanda con branching logico che salta una domanda nel caso di risposta negativa dell'utente; una domanda numerica

[1]Tale approccio è molto simile al modo di calcolare valori nei fogli elettronici, dove si utilizzano combinazioni di speciali funzioni *built-in* per la computazione automatica dei valori di alcune celle sulla base dei valori presenti nelle altre celle del foglio stesso.

senza range; una domanda numerica con range; una domanda in forma di stringa testuale. Inoltre il documento mostra un XML contenente le domande sia in italiano che in inglese, così da rendere l'esempio più interessante. Da notare che il documento non contiene informazioni sulle chiavi primarie in quanto esse saranno create automaticamente dal DBMS nel caso in cui il questionario sarà accettato dal sistema e inserito nel database.

5.1.2 XML per un questionario da compilare

Il formato di tale XML è molto simile al precedente. Le differenze risiedono principalmente nel fatto che le domande e le opzioni contengono un attributo di indicizzazione che deriva direttamente dalle chiavi primarie nel database, che vengono fornite le informazioni nella sola lingua dall'utente (per evitare di sovraccaricare inutilmente la rete) e che il documento non contiene metadati, quali ad esempio informazioni riguardanti il creatore del questionario. Sempre in Appendice B è mostrato di seguito un esempio di documento XML che il server invierebbe a un client per un questionario nella sola lingua italiana, con la stessa struttura di quello descritto precedentemente.

5.1.3 XML per i dati di un questionario compilato

Anche quest'ultimo formato è molto simile ai precedenti, nella sua struttura. Ovviamente il documento non prevede l'invio di informazioni che il server conosca già. Si limita a indicare per ogni domanda le opzioni scelte dall'utente o gli input da egli o ella inseriti. Come è possibile vedere, ancora una volta in Appendice B, l'XML in questione, sempre relativo allo stesso esempio di prima, è molto più snello dei due precedenti.

É bene fare un'ulteriore considerazione. In realtà l'utilizzo di un XML in questo caso può sembrare un po' forzato, in quanto sarebbe perfettamente possibile inviare i risultati al server per mezzo del body HTTP in seguito a una chiamata con metodo POST, nel tipico formato <*chiave,valore*>. In effetti è possibile ampliare il server permettendogli di ricevere i dati anche in tale formato, senza

l'obbligo di passare attraverso XML. Ad ogni modo reputiamo che l'utilizzo di XML sia comunque doveroso per una questione di coerenza delle interfacce di comunicazione tra client e server finora discusse.

Modi di invio e ricezione delle risposte

Sempre in questo contesto è bene menzionare i diversi modi di invio dei dati al server. Esistono due approcci possibili: il primo prevede l'invio di singole risposte, ogniqualvolta il paziente risponda a una domanda e passi alla successiva; l'altro approccio prevede invece la raccolta di tutte le risposte lato client e l'invio al server di un unico corpo messaggio contenente tutte le risposte date. La struttura XML che abbiamo appena mostrato è capace di soddisfare entrambi gli scenari: nel caso di invio di risposte singole basterebbe inserire nel corpo una risposta per volta. Vi sono però alcuni "trade off" da considerare nella scelta di quale dei due modi utilizzare. Se da un lato volessimo aspettare la raccolta di tutte le risposte lato client prima di inviarle al server, un malfunzionamento sul client causerebbe la perdita dei dati fino a quel momento raccolti e obbligherebbe il paziente a rispondere nuovamente alle stesse domande per la stessa somministrazione; d'altro canto l'invio dei dati al server domanda per domanda avrebbe altri svantaggi, come ad esempio un rallentamento nelle operazioni di compilazione dovuto all'overhead della rete, in altri casi addirittura l'impossibilità nell'inviare tali informazioni qualora l'accesso alla rete sia intermittente (come accade spesso con gli apparecchi mobili) o ancora complicazioni aggiuntive se vogliamo permettere a un utente di cambiare le risposte date alle domande precedenti, cosa che accade non di rado.

La decisione finale ricade sul secondo approccio, in quanto al suo svantaggio prevale il fatto che in genere i test sono piuttosto corti e l'eventuale guasto momentaneo di un client causerebbe sì la necessità di reinserire gli stessi dati, ma tali dati sono in genere pochi. Il vantaggio di tale approccio è inoltre quello di consentire ai client di raccogliere i dati in maniera distaccata dal server, e ciò garantisce prestazioni dei client più elevate. Ciononostante si offre la possibilità a eventuali altri client di terze parti, che si vogliano interfacciare con $\mathbb{P}$.ROse, di

potere usare l'altro approccio, qualora credano che sia loro più conveniente. Per ottenere ciò, il server accetterà documenti XML contenenti anche un sottoinsieme delle risposte attese per un dato questionario e persisterà nel database una risposta per volta, man mano che esse arrivano dai client.

Gestione degli allarmi

Un'ulteriore problema relativo all'invio dei risultati di un questionario prevede l'implementazione di cosiddetti "allarmi". Come abbiamo già precedentemente accennato, ℙ.`ROse` deve essere in grado di controllare i dati ricevuti dai pazienti quando questi giungono al server e far scattare segnalazioni nel caso in cui questi risultino fuori norma. Tale comportamento può essere implementato molto semplicemente mediante una funzione che controlli tutti i risultati forniti dai pazienti prima di memorizzarli nel database e, in caso di necessità, invii un SMS o un'email al dottore interessato.

5.2 Il server web

Il server ha la necessità di garantire alte prestazioni e un'adeguata scalabilità, in quanto è l'unico punto di accesso per tutti i servizi offerti dal sistema. Vedremo, purtroppo, come quest'ultima necessità non sia, almeno al giorno d'oggi, soddisfacibile e come siamo invece costretti a optare per un'implementazione del sistema meno scalabile.

Oggigiorno sono disponibili numerose soluzioni con caratteristiche di scalabilità a costi abbastanza contenuti. Sono le soluzioni nella cosiddetta *cloud*, ovvero tecnologie che offrono hosting server e gestione dei dati distribuiti e a replicazione automatica, solitamente abbastanza facili da configurare (entro certi limiti, diversi per ogni fornitore del servizio). Un ulteriore vantaggio di tale tecnologia è che elimina i costi di acquisto e gestione di macchine server fisiche, nonché i costi di affitto e protezione dei locali ove le macchine saranno attive. Oltre all'abbattimento di tali spese viene aumentata di gran lunga la disponibilità dei server nel tempo, in quanto i server nella "nuvola" si ridistribuiscono il carico e

si autogestiscono (entro certi limiti) i malfunzionamenti. Il tutto mantenendo dei prezzi più che competitivi [Hayes, 2008][2].

Come abbiamo già accennato, la scelta finale non verte purtroppo verso il cloud computing. Il problema principale risiede nella sicurezza. La cloud infatti, proprio per garantire alte prestazioni, è organizzata in una struttura distribuita che fa uso di replicazione, dove le copie delle risorse, sia esse codice che dati, vengono dislocate e spostate automaticamente in ogni angolo del pianeta, spesso senza precisi vincoli, in un modo tale da rendere difficile prevedere in maniera esatta dove esse si trovino in un dato momento o, addirittura, dove possano trovarsi tutte le copie delle medesime [Kadambi et al., 2011, Kandukuri et al., 2009, Christodorescu et al., 2009]. Ciò, chiaramente, complicherebbe di gran lunga ogni qualsivoglia problema legale dovuto al mantenimento della privacy dei dati dei pazienti. Fino a quando non verranno proposte soluzioni accettabili al problema non sarà possibile gestire dati così sensibili, come quelli dei pazienti, mediante un'infrastruttura come quella della cloud.

Siamo quindi costretti, almeno al momento attuale, a scegliere una soluzione più "classica", ovvero un server fisico dislocato in un luogo ben preciso, senza grandi vantaggi in termini di tolleranza flessibile ai guasti o velocità. Per questo motivo non sarà possibile utilizzare la cloud per ottenere alta disponibilità del servizio. Un attacco di *denial-of-service*, per esempio, è molto difficile da contrastare in architetture non distribuite e automaticamente scalabili.

[2]Esistono moltissimi servizi che implementano hosting server e dati sulla cloud. Uno che reputiamo essere molto interessante è quello offerta da Google: la *Google App Engine* [Zahariev, 2009]. Il framework permette agli sviluppatori di ospitare i loro server web all'interno della stessa infrastruttura di Google e di realizzare e pubblicare le applicazioni in modo facile e veloce, di scrivere codice server dinamico in Java, Python o Go e di utilizzare il suo Datastore per il salvataggio dei dati. Un'altra soluzione che reputiamo interessante è quella fornita da Amazon, denominata *Amazon Elastic Compute Cloud* (Amazon EC2) [Amazon.com, 2011]. Essa è un servizio di cloud computing che offre le stesse qualità di scalabilità di quello di Google, ma in un modo parecchio differente: mette a disposizione dei gestori un server virtuale, al quale è possibile accedere, per esempio, via *ssh*. Il server così ottenuto è totalmente configurabile, come se fosse una macchina server "reale". Anche questo servizio è offerto a costi particolarmente contenuti, consentendo agli utenti di pagare in base alle risorse effettivamente utilizzate.

5.2.1 Architettura del server

La scelta più ovvia, considerando tutto quello che abbiamo appena stabilito, è quella di utilizzare un'unica macchina che contenga sia la componente software del server sia la componente di memorizzazione dei dati. L'istanza server che utilizzeremo sarà basata sul sistema operativo Linux e sul server web Apache, in quanto offrono il miglior rapporto qualità/prezzo. Utilizzeremo un database relazionale gestito mediante PostgreSQL, in quanto è tra i DBMS più veloci, sicuri e completi tra quelli disponibili in versione open-source. Per quanto concerne l'architettura del software lato server utilizzeremo il famosissimo pattern architetturale *Model-View-Controller* (MVC) in quanto offre numerosissimi vantaggi nella programmazione di sistemi web. L'idea chiave di tale pattern è quella di separare le interfacce utente dai dati sottostanti che le rappresentano [Leff and Rayfield, 2001]: il *Controller* è quella parte software che gestisce la logica delle richieste dell'utente, il *Model* è quella parte del sistema che gestisce i dati persistenti nel database e il *View* si occupa di presentare all'utente una visione dei dati secondo precisi requisiti presentazionali. Per tali motivi ci si riferisce a questo pattern anche col nome di *Presentation/Abstraction/Control* (PAC). Uno dei vantaggi chiave di questo approccio è che mantiene una bassa interdipendenza tra i moduli garantendo al contempo una forte coerenza al loro interno. Ciò garantisce, tra le altre cose, una elevata manutenibilità del codice.

A tutt'oggi esistono moltissimi framework per la realizzazione di applicazioni web e la maggior parte di essi utilizzano il pattern architetturale MVC. Giusto per citarne alcuni tra i più famosi: *CodeIgniter*, *Drupal*, *Google Web Toolkit*, *Grok*, *Ruby on Rails*. Quello che abbiamo scelto per la realizzazione di $\mathbb{P}$.`ROse` è *Django*, un web framework in *Python* basato anch'esso sul modello MVC. Django offre innumerevoli caratteristiche utilissime per la programmazione di servizi web tra cui: possibilità di definire le URL in maniera del tutto personalizzata mediante l'utilizzo di espressioni regolari; un completo sistema di *template* per la generazione delle view; un completo sistema di caching delle pagine; funzionalità di internazionalizzazione; interfacce di amministrazion; inoltre è facilmente configurabile e pubblicabile sul web. Grazie al linguaggio Python, tipicamente

poco verboso, e a buone tecniche di programmazione è inoltre possibile scrivere la parte server in un esiguo numero di righe di codice [Djangoproject.com, 2011].

5.2.2 Il database

Come abbiamo già accennato utilizzeremo il DBMS PostgreSQL. Come ogni DBMS che si rispetti esso offre un'estensione del linguaggio SQL sia per la gestione di tipi di dato, sia per l'automazione dei controlli di integrità dei dati, sia per la possibilità di programmare *trigger* all'inserimento, modifica o rimozione di tuple nelle tabelle.

Il lavoro fondamentale che è necessario effettuare in fase progettuale è quello di stabilire la struttura logica del database, ovvero il modo in cui i dati verranno strutturati in memoria e quindi poi recuperati per mezzo delle interrogazioni. Per progettare tale struttura logica abbiamo utilizzato il modello di rappresentazione dei dati denominato *Entity-Relationship* (ER), in quanto è sia abbastanza astratto da non contenere troppe caratteristiche implementative, ma è anche facilmente traducibile in schema relazionale SQL. Mostreremo infine la sua implementazione all'interno del framework Django e la sua traduzione finale in relazioni SQL.

Modellazione degli utenti e loro relazioni

Il fulcro del sistema sono i suoi utenti e il database deve quindi permettere la memorizzazione delle loro informazioni personali e del loro ruolo all'interno del sistema stesso. Come stabilito nei requisiti il sistema è pensato per essere usato da due tipi di utenti: i medici e i pazienti (cfr 4.6.4 *Classi di utenti*). I primi hanno il ruolo di creare i questionari e gestire le somministrazioni e la raccolta dati, i secondi hanno il ruolo di compilare i questionari somministrati entro le date stabilite. É inoltre necessario che si stabilisca un legame tra i pazienti e i loro medici: un paziente può ricevere somministrazioni soltanto dai suoi medici. Tale legame deve essere memorizzato in maniera persistente nel database.

Il database contiene innanzitutto una tabella *Users* che memorizza le informazioni di tutti gli utenti del sistema, indipendentemente dal tipo di utente:

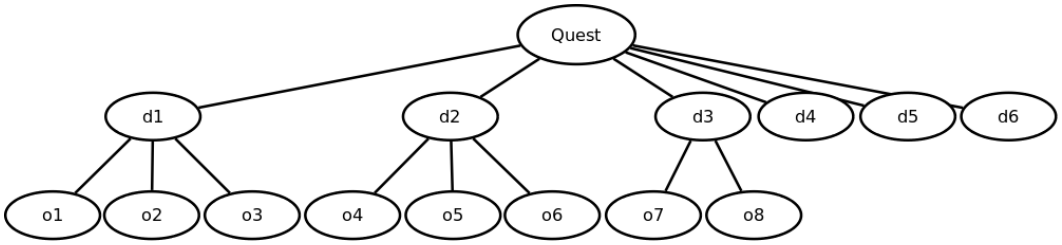

Figura 5.1: Esempio di struttura gerarchica di un questionario.

tali informazioni sono tipicamente l'email e la password per l'accesso alle aree riservate. Due tabelle specializzano gli utenti a seconda che siano medici o pazienti: *Doctors* e *Patients* contenenti dati anagrafici e informazioni aggiuntive dei medici e dei pazienti. Un'ultima tabella, la tabella *Follows* contiene coppie <*medico,paziente*> a indicare quali medici seguono un certo paziente o, viceversa, quali pazienti sono seguiti da un certo medico.

Modellazione dei questionari

Un questionario è implicitamente organizzato secondo una struttura gerarchica: contiene una serie di domande, ognuna delle quali contiene una serie di possibili risposte tra cui è concesso scegliere; alcune domande del questionario, inoltre, non prevedono opzioni, ma l'inserimento diretto di input numerici o stringhe di caratteri. La struttura di un questionario può quindi essere facilmente rappresentata con un albero ordinato. La Figura 5.1 mostra un esempio di tale situazione. Nella Figura esiste un questionario *Quest* il quale contiene sei domande: $d1$ è di tipo "opzione singola", ovvero una e una sola scelta possibile tra le tre presenti $o1$, $o1$ e $o3$; $d2$ è di tipo "opzione multipla", ovvero un numero qualunque di scelte effettuabili tra le tre disponibili $o4$, $o5$ e $o6$; $d3$ di tipo condizionale, ovvero crea un branch logico in cui se il paziente risponde "sì" si sposta alla prossima domanda altrimenti passa a $d5$; $d4$ di tipo numerico senza range; $d5$ di tipo numerico con range; $d6$ di tipo stringa. Supponiamo che un medico voglia assegnare tale questionario al paziente $p1$ con cadenza settimanale, ogni lunedì per due lunedì consecutivi e lo stesso questionario al paziente $p2$ un'unica volta senza ripetizio-

ni[3]. Per complicare un po' lo scenario supponiamo che vogliamo memorizzare il questionario *Quest* in due lingue: italiano e inglese. Il problema che ci poniamo è come strutturare in tabelle relazionali una struttura così complessa e ricca di possibilità.

La soluzione al problema appena discusso passa attraverso la separazione dei concetti di domande di un questionario, di opzioni relative a una domanda, di somministrazione e di outcome di un questionario. Quando si definiscono le domande di un questionario bisogna ammettere che una stessa domanda abbia più entry, una per ogni lingua usata. Inoltre bisogna permettere di selezionare il tipo della domanda: è una domanda con opzioni? è una domanda senza opzioni? se è senza opzioni, ci si aspetta un numero o una stringa come input? se ci si aspetta un input, che range deve avere? La nostra soluzione a questo problema è una tabella *Questions* nella quale si indica il tipo di domanda: 'o' per opzionale, 'b' per branch logico, 'n' per input numerico senza range, 'r' per input numerico con range, 's' per input stringa. Altri due attributi numerici $left\_val$ e $right\_val$ servono per impostare caratteristiche aggiuntive della domanda: se la domanda è di tipo 'r' i due valori indicano il range di validità dell'input numerico che l'utente può fornire; se la domanda è di tipo 's' i due numeri indicano la lunghezza minima e massima della stringa che l'utente deve inserire; se la domanda è di tipo 'o' i due valori indicano il numero minimo e il numero massimo di opzioni contemporaneamente selezionabili (per esempio $left\_val = 1$ e $right\_val = 1$ indica che è possibile scegliere una e una sola opzione, $left\_val = 0$ e $right\_val = \infty$ indica che è possibile scegliere un numero arbitrario di opzioni, anche nessuna). Infine, se la domanda è di tipo 'b' i due valori indicano la prossima domanda a cui andare se l'utente ha selezionato rispettivamente la prima o la seconda delle due opzioni disponibili, nel seguente modo: $left\_val$ indica l'*offset*, nella sequenza del questionario, tra la domanda attuale e la prossima domanda, qualora l'utente scegliesse la prima delle due opzioni disponibili; $right_Val$, invece, è l'offset qualora l'utente scegliesse la seconda opzione. Ad esempio, se $left\_val = -2$ e l'utente

[3]É esattamente lo stesso esempio discusso in precedenza, in sezione 5.1.1 *XML per nuovo questionario* e mostrato in versione XML in Appendice B.

scegliesse la prima opzione della domanda, nel momento in cui si procedesse nel questionario, l'utente verrebbe riportato indietro di due domande[4].

La tabella *Options* memorizza le opzioni possibili per quelle domande che sono di tipo 'o'. La tabella *Givings* invece viene utilizzata per la somministrazione dei questionari ai pazienti. In tale tabella è possibile indicare una finestra di tempo durante la quale il questionario è compilabile e non vi è nessuna restrizione a quante volte è possibile somministrare lo stesso questionario a un certo paziente. L'ultima tabella, la tabella *Outcomes*, è quella che memorizza i risultati delle esecuzioni dei questionari. Essa è capace di memorizzare, per ogni domanda, uno dei seguenti tre dati: 1) la/e scelta/e effettuata/e se si tratta di domanda di tipo opzionale; 2) il numero inserito se si tratta di domanda numerica; 3) la stringa inserita se la domanda è di tipo stringa. É inoltre previsto l'uso di speciali *triggers* che si attivino all'inserimento di dati all'interno della tabella *Outcomes*, al fine di controllare che vengano rispettati i vincoli stabiliti per il tipo di domanda e rifiutare input fuori norma.

Diagramma ER e schema relazionale

La Figura 5.2 mostra il diagramma ER che descrive quanto detto finora. Nel diagramma le entità sono rappresentate dai rettangoli, le relazioni dai rombi, le linee indicano partecipazione multipla, le frecce indicano partecipazione singola e linee e frecce in grassetto indicano partecipazione obbligatoria. Inoltre, gli attributi sottolineati indicano le chiavi primarie e quelle sottolineate con tratteggio indicano gli attributi unici. Da notare che sono stati omessi molti attributi supplementari per ragioni di semplicità del diagramma. Il primo passo è quello di trasformare tale modello dei dati in classi Python nel framework di Django. Esso stesso si preoccuperà di effettuare la trasformazione del modello a classi in modello relazionale SQL. Ad ogni modo, per completezza, riportiamo qui di seguito sia la rappresentazione del modello in classi Python, sia quella in SQL (in una forma leggermente diversa rispetto a quella che Django utilizza in pratica).

[4]Questo è ovviamente un esempio poco utile in pratica, ma adatto a chiarire le idee sull'utilizzo dei due valori.

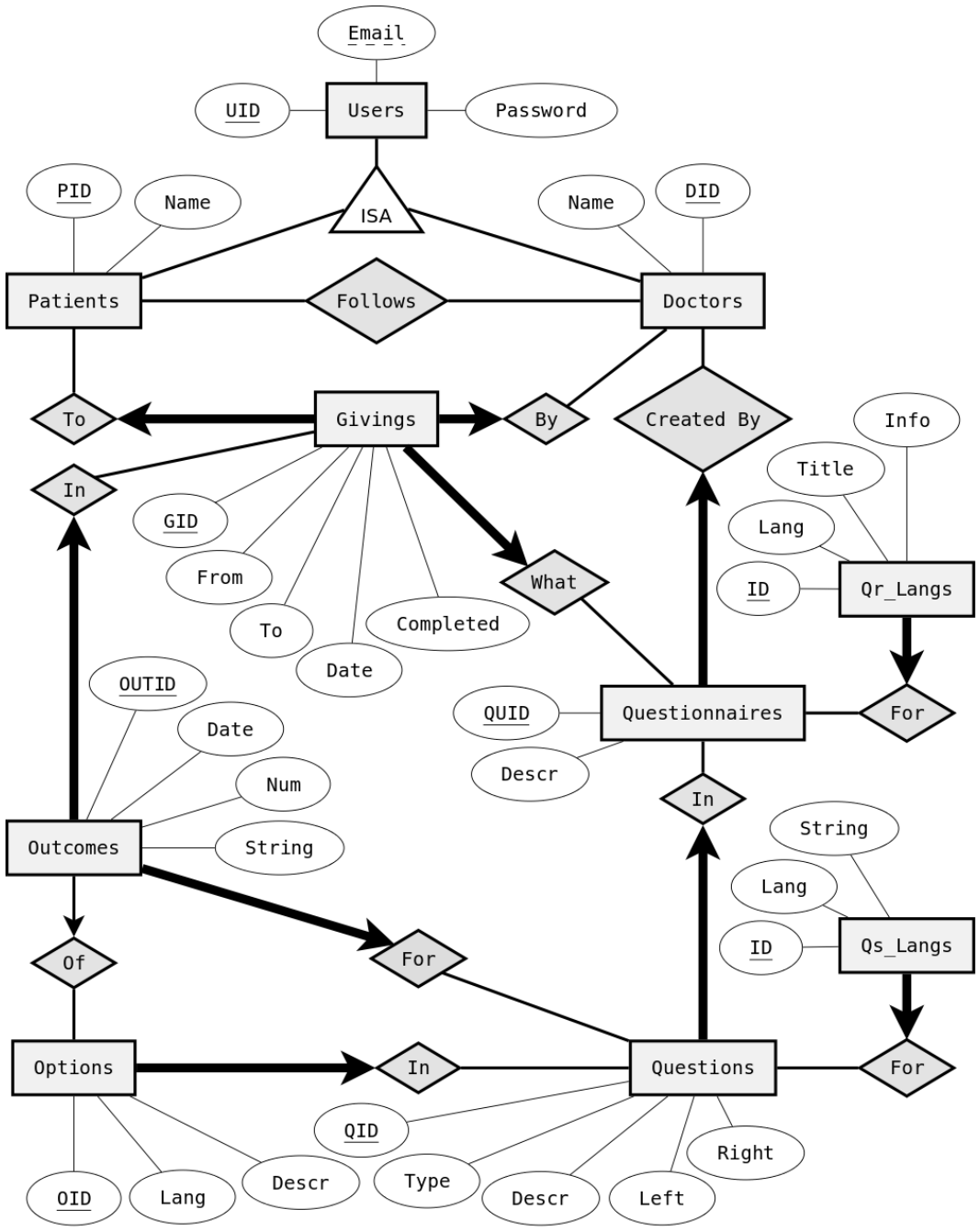

Figura 5.2: Diagramma ER del database relazionale.

L'entità *Users* è già implementata da Django nel suo modulo di autenticazione e quindi possiamo ometterla. Le classi per *Patients* e *Doctors* importeranno le chiavi primarie dall'entità *Users* e specializzeranno gli utenti a seconda che siano pazienti o dottori, come mostrato qui di seguito.

```
class Patient(models.Model):
    pid  = models.ForeignKey(User, primary_key=True)
    name = models.CharField(max_length=200)

class Doctor(models.Model):
    did  = models.ForeignKey(User, primary_key=True)
    pid  = models.ManyToManyField(Patient)
    name = models.CharField(max_length=200)
```

Tale rappresentazione può essere espressa in SQL nel seguente modo, dove la tabella *Follows* implementa la relazione many-to-many tra pazienti e dottori:

```
CREATE TABLE Patients (
    pid   integer PRIMARY KEY,
    name  varchar(200),
    FOREIGN KEY (pid) REFERENCES Users(uid)
);
CREATE TABLE Doctors (
    did   integer PRIMARY KEY,
    name  varchar(200),
    FOREIGN KEY (did) REFERENCES Users(uid)
);
CREATE TABLE Follows (
    did integer,
    pid integer,
    FOREIGN KEY (did) REFERENCES Doctors(did),
    FOREIGN KEY (pid) REFERENCES Patients(pid),
    PRIMARY KEY (did, pid)
);
```

Le tabelle *Questionnaires*, *Questions* e *Options* servono a memorizzare le strutture gerarchiche dei questionari. Nella prima tabella viene anche memorizzato l'ID del dottore che ha creato il questionario. Lo schema permette di memorizzare questionari in più lingue mediante l'uso di apposite tabelle e dell'attributo *lang*, una stringa che indichi la lingua secondo uno degli standard usati da HTTP, come ad esempio l'*RFC 5646* [Phillips and Davis, 2009]. La tabella delle domande permette inoltre di gestire il tipo di domanda mendiate l'attributo *type* e il range dei valori ammessi in input o del numero di opzioni contemporaneamente selezionabili o ancora la prossima domanda nel caso di branching condizionale, mediante *left_val* e *right_val*, come abbiamo precedentemente illustrato.

```
class Questionnaire(models.Model):
    quid        = models.AutoField(primary_key=True)
    created_by  = models.ForeignKey(Doctor)
    description = models.CharField(max_length=1000, null=True)

class Questionnaire_Lang(models.Model):
    quid  = models.ForeignKey(Questionnaire)
    lang  = models.CharField(max_length=5)
    title = models.CharField(max_length=1000)
    info  = models.TextField()

class Question(models.Model):
    qid         = models.AutoField(primary_key=True)
    quid        = models.ForeignKey(Questionnaire)
    type        = models.CharField(max_length=3)
    left_val    = models.DecimalField(max_digits=32, decimal_places=16)
    right_val   = models.DecimalField(max_digits=32, decimal_places=16)
    description = models.CharField(max_length=500, null=True)

class Question_Lang(models.Model):
    qid    = models.ForeignKey(Question)
    lang   = models.CharField(max_length=5)
    string = models.CharField(max_length=500)

class Option(models.Model):
    oid  = models.AutoField(primary_key=True)
    qid  = models.ForeignKey(Question)
```

```
    lang = models.CharField(max_length=5)
    description = models.CharField(max_length=500)
```

La struttura relazionale può essere espressa nel seguente modo, dove abbiamo omesso le tabelle delle lingue in quanto immediate:

```
CREATE TABLE Questionnaires (
    quid           serial PRIMARY KEY,
    description    varchar(1000) NOT NULL,
    created_by_did integer NOT NULL,
    FOREIGN KEY (created_by_did) REFERENCES Doctors(did),
);
CREATE TABLE Questions (
    qid         serial PRIMARY KEY,
    quid        integer,
    type        char(1) NOT NULL
                CHECK(type='n' OR type='s' OR type='o' OR type='b'),
    left_val    numeric(32,16),
    right_val   numeric(32,16),
    description varchar(500) NOT NULL,
    FOREIGN KEY (quid) REFERENCES Questionnaires(quid),
);
CREATE TABLE Options (
    oid         serial PRIMARY KEY,
    qid         integer NOT NULL,
    lang        varchar(5),
    description varchar(500),
    FOREIGN KEY (qid) REFERENCES Questions(qid),
);
```

Le ultime due tabelle memorizzano rispettivamente le somministrazioni dei questionari ai pazienti e i loro risultati, somministrazione per somministrazione. La tabella *Givings* memorizza l'ID del questionario somministrato, l'ID del medico che effettua la somministrazione, l'ID del paziente a cui si somministra il questionario, la data di inizio e di fine della somministrazione e un booleano che

indica se quella somministrazione è già stata completata[5]. La tabella *Outcomes* è atta a contenere il riferimento alla somministrazione cui si riferisce, l'ID della domanda cui si riferisce e un dato che indica la scelta effettuata o il valore di input inserito. Per risalire, ad esempio, al paziente che ha inviato il risultato basterà unire questa tabella con la tabella *Givings* selezionando le tuple con attributo *gid* uguale.

```
class Giving(models.Model):
    gid        = models.AutoField(primary_key=True)
    did        = models.ForeignKey(Doctor)
    pid        = models.ForeignKey(Patient)
    quid       = models.ForeignKey(Questionnaire)
    from_date  = models.DateTimeField()
    to_date    = models.DateTimeField()
    given_date = models.DateTimeField(auto_now=True)
    completed  = models.BooleanField(default=False)

class Outcome(models.Model):
    outid  = models.AutoField(primary_key=True)
    gid    = models.ForeignKey(Giving)
    qid    = models.ForeignKey(Question)
    oid    = models.ForeignKey(Option, null=True)
    num    = models.DecimalField(max_digits=32, decimal_places=16, null=True)
    string = models.CharField(max_length=5, null=True)
    date   = models.DateTimeField(auto_now=True)
```

Ecco, infine, la loro rappresentazione SQL:

```
CREATE TABLE Givings (
    gid        serial PRIMARY KEY,
    did        integer NOT NULL,       -- dottore che somministra
    pid        integer NOT NULL,       -- paziente a cui si somministra
    quid       integer NOT NULL,       -- questionario somministrato
```

[5]Tale ridondanza è stata inserita per motivi prestazionali, in quanto consente di evitare molteplici *join* per sapere se un dato questionario sia stato già completato o meno.

```
    from_date  timestamp NOT NULL,     -- data inizio compilazione
    to_date    timestamp NOT NULL,     -- data scadenza compilazione
    given_date timestamp DEFAULT NOW(),-- data di somministrazione
    completed  boolean DEFAULT FALSE,
    FOREIGN KEY (did)  REFERENCES Doctors(did),
    FOREIGN KEY (pid)  REFERENCES Patients(pid),
    FOREIGN KEY (quid) REFERENCES Questionnaires(quid)
);
CREATE TABLE Outcomes (
    outid  serial PRIMARY KEY,
    gid    integer NOT NULL,
    qid    integer NOT NULL,
    oid    integer,
    num    numeric(32,16),
    string varchar(500),
    date   timestamp DEFAULT NOW(),
    FOREIGN KEY (gid) REFERENCES Givings(gid),
    FOREIGN KEY (qid) REFERENCES Questions(qid)
    FOREIGN KEY (oid) REFERENCES Options(oid)
);
```

Inoltre, Django si occupa automaticamente di creare degli indici sulle tabelle, utilizzando le strutture dati a disposizione di PostgreSQL (es. B-tree, Hash, ecc.).

5.2.3 Funzioni MVC

Come abbiamo già accennato, in un MVC il Controller gestisce la logica di accesso alle risorse del server e fa da tramite tra l'utente e i dati presenti nel sistema. a una richiesta dell'utente (tipicamente una URL) il Controller esegue le azioni corrispondenti, se necessario interroga e/o modifica li database e infine genera una View che rappresenta il nuovo stato del sistema dopo l'operazione effettuata. Il Controller è quindi il centro nevralgico di un sistema basato sul pattern MVC. Programmarlo significa fornire all'utente tutte le funzionalità da lui richieste.

Nella filosofia di Django non vi è una netta distinzione tra le View e il Controller, in quanto si guarda a una View come a un modo unico di chiamare una

funzionalità del server, alla quale sia attribuito uno specifico modello di rappresentazione grafica[6]. Ciò che conta è che a ogni View corrisponde una specifica funzionalità del server e questo lo mette in stratta relazione con il Controller, tanto da sfocare i confini tra di essi. In questa sezione ci occuperemo di progettare le funzioni principali della componente View/Controller di ℙ.ROsE, al livello di astrazione più basso che forniremo in questa trattazione. Mostreremo qui di seguito le funzioni lato server secondo un preciso schema: in alto indicheremo la URL relativa alla risorsa che staremo per definire unitamente al metodo HTTP con il quale è possibile interrogarla (GET oppure POST); sotto indicheremo la categoria di utenti (cfr. 4.6.4 *Classi di utenti*) ai quali la risorsa verrà resa disponibile (alcune risorse sono disponibili anche a utenti non registrati, altri sono a tutti gli utenti registrati, altre ancora solo a dottori o solo a pazienti); infine descriveremo in linguaggio naturale la risorsa stessa e/o il suo comportamento atteso, evidenziando, qualora necessario, comportamenti speciali che il codice lato server deve avere.

- `POST /register`
  `Disponibile a: Utenti non registrati`
  Funzionalità che permette la registrazione a ℙ.ROsE per mezzo di un indirizzo email valido. Una volta registrati al servizio sarà necessario validare l'indirizzo email per mezzo di una speciale URL.

- `GET /register/confirm/<info>`
  `Disponibile a: Utenti registrati ma non confermati`
  Funzionalità che permette la finalizzazione della registrazione. Il campo *info* contiene le informazioni circa l'utente che si sta confermando e, possibilmente, stringhe per controlli aggiuntivi. Notare che il metodo utilizzato qui è il GET, poiché il link a tale risorsa viene comunemente fornito all'utente tramite email e invocato senza l'ausilio di un form.

[6]Come si può leggere nel sito stesso di Django: *"A view is a 'type' of Web page in your Django application that generally serves a specific function and has a specific template."*

- `GET /home`
  `Disponibile a: Utenti non registrati`
  Pagina principale di ℙ.ROsE. E' una pagina ad accesso pubblico.

- `GET /accounts/login`
  `Disponibile a: Tutti gli utenti`
  Mostra il form per effettuare il login.

- `POST /accounts/login`
  `Disponibile a: Tutti gli utenti`
  Funzione di autenticazione. Restituisce, in caso di successo, un cookie con sessione opportunamente impostata.

- `GET /accounts/profile`
  `Disponibile a: Dottori e pazienti`
  Permette di accedere alla pagina del proprio profilo personale, nella quale si possono visualizzare e modificare le proprie informazioni. L'utente deve essere già autenticato oppure si verrà reindirizzati alla pagina di login (ciò vale per tutte le funzioni da qui in poi).

- `POST /accounts/profile`
  `Disponibile a: Dottori e pazienti`
  Permette di modificare i dati presenti nel proprio profilo personale.

- `GET /accounts/profile/<pid>`
  `Disponibile a: Dottori`
  Permette di accedere alla pagina del profilo personale del paziente *pid*, il quale deve essere già connesso al dottore che la richiede.

- `GET /accounts/profile/<did>`
  `Disponibile a: Dottori`
  Permette di accedere alla pagina del profilo personale del dottore *did*, il quale deve essere già connesso all'utente che la richiede.

- `GET /doctors/mine`
  `Disponibile a: Pazienti`
  Visualizza la lista dei dottori del paziente correntemente loggato.

- `GET /patients/mine`
  `Disponibile a: Dottori`
  Visualizza la lista dei pazienti sotto l'ala del dottore correntemente loggato.

- `POST /patients/new`
  `Disponibile a: Dottori`
  Permette di aggiungere un paziente tra quelli del dottore. L'aggiunta avviene per mezzo di un indirizzo email al quale il sistema provvederà a inviare richiesta di accettazione. Solo quando il paziente avrà accettato la richiesta il paziente riceverà le somministrazioni del dottore e il dottore potrà ricevere i risultati dei questionari (cfr. 4.7.1 *Aggiunta di pazienti già registrati*).

- `GET /givings`
  `Disponibile a: Pazienti`
  Visualizza la lista dei questionari `ePRO` ancora da effettuare. Mostra soltanto i questionari il cui periodo di validità comprende la data e l'ora in cui la risorsa viene richiesta.

- `GET /givings`
  `Disponibile a: Dottori`
  Visualizza la lista dei questionari che un dottore ha somministrato ai propri pazienti, permettendo la navigazione sia di quelli già completati, sia di quelli ancora da completare. Deve consentire, come minimo, il raggruppamento dei questionari per paziente, per data di somministrazione e per data di validità.

- `GET /givings/outcomes/<gid>`
  `Disponibile a: Dottori`
  Prende in input un *gid*, ovvero l'ID di una singola somministrazione, e ne mostra tutti i risultati.

- `GET /givings/outcomes/<gid1,...,gidn>`
  `Disponibile a: Dottori`
  Prende in input una lista di *gid* separati da virgola o altro delimitatore e mostra una visualizzazione aggregata e comparata dei risultati di tutte le somministrazioni specificate, includendo informazioni come: la percentuale di pazienti che hanno completato i questionari, medie, massimi e minimi di ogni serie di questionari, grafici di andamento, ecc.

- `POST /givings/new/<quid>/<pid>`
  `Disponibile a: Dottori`
  Permette la somministrazione del questionario *quid* al proprio paziente *pid*.

- `GET /givings/compile/<gid>`
  `Disponibile a: Dottori e pazienti`
  Permette la ricezione lato client del questionario corrispondente alla somministrazione *gid* ai fini della sua compilazione. Questa richiesta lancia l'interfaccia di esecuzione del questionario. La funzionalità è disponibile sia ai pazienti che ai dottori. Nel caso di un paziente bisogna controllare che la somministrazione sia stata attribuita proprio a lui o lei. Se il questionario è stato richiesto da un dottore, è necessario accertarsi che la somministrazione sia stata effettuata dal dottore in questione e, per motivi di sicurezza, è necessario che il paziente inserisca comunque le proprie credenziali di accesso (cfr. 4.7.1 *Esecuzione dei questionari*). In entrambi i casi bisogna controllare che la somministrazione sia attiva nel momento della chiamata (ovvero che non sia troppo presto o troppo tardi).

- `GET /givings/getxml/<gid>`
  `Disponibile a: Dottori e pazienti`
  Questa funzione permette di ricevere una rappresentazione XML del questionario corrispondente alla somministrazione *gid*. Il formato è quello descritto in sezione 5.1 *Interfacce XML*.

- `POST /givings/done/<gid>`
  `Disponibile a: Dottori e pazienti`
  Funzione del server da chiamare quando si vogliono inviare i risultati di un questionario appena completato. La funzione effettua gli stessi controlli indicati nel punto precedente e infine notifica l'utente dell'avvenuto salvataggio dei dati o del fallimento dell'operazione.

- `GET /epros`
  `Disponibile a: Dottori`
  Permette di accedere alla lista dei questionari presenti nel sistema e di poterne selezionare uno da somministrare a un proprio paziente o a un gruppo di pazienti.

- `GET /epros/create`
  `Disponibile a: Dottori`
  Permette di accedere all'interfaccia per la creazione di nuovi questionari.

- `POST /epros/new`
  `Disponibile a: Dottori`
  Funzione del server da chiamare quando si vuole aggiungere al sistema un nuovo questionario appena creato. La definizione del questionario avviene lato client e mediante questa funzione è possibile persistere il questionario nel database centrale.

- `POST /epros/edit/<quid>`
  `Disponibile a: Dottori`
  Permette di modificare il questionario *quid*. Disponibile solo al dottore che ha creato il questionario e solo fintanto che il questionario indicato non sia mai stato somministrato (cfr. 5.4 *Possibili miglioramenti futuri* per una soluzione a questa limitazione).

5.2.4 Autenticazione

Implementare l'autenticazione lato server mediante Django è quasi immediato. Django infatti fornisce un modulo apposito che gestisce tutte le principali funzioni di autenticazione. Il modulo comunica con il modello dei dati *User* per recuperare le informazioni di autenticazione e fornisce delle funzioni per il login, insieme a speciali direttive (chiamate *decorators*) che, aggiunte alle funzioni lato server, permettono di forzare l'accesso a una funzionalità soltanto agli utenti loggati, reindirizzando automaticamente l'utente verso la pagina di login in caso negativo.

Il requisito funzionale del server numero S5 prevedeva che l'utente potesse autenticarsi mediante sue credenziali preesistenti, come credenziali per caselle email o altro. É possibile implementare ciò utilizzando apposite librerie messe a disposizione dai più famosi provider di servizi di autenticazione. Citiamo tra tutti OpenID [Recordon and Reed, 2006], nel quale l'identità univoca di ogni utente viene rappresentata da una risorsa URL. Esistono a tutt'oggi numerosi provider di tale servizio, tra cui citiamo Google [Google.com, 2011], Yahoo! [Yahoo.com, 2011] e AOL [AOL.com, 2011]. La libreria per Python *OpenID Enabled* [OpenID Enabled, 2011] permette di interfacciare $\mathbb{P}$.`ROse` con lo standard di OpenID.

Un'altra tecnologia che permette l'autenticazione degli utenti mediante le loro credenziali preesistenti si chiama *OAuth* [OAuth.net, 2011]. Essa stabilisce un preciso protocollo di autenticazione mediante HTTP e gestisce le autenticazioni riuscite mediante l'uso di speciali oggetti detti *token*, che permettono all'utente di rimanere autenticato fino alla fine della propria sessione di lavoro. Google ne ha implementato una sua versione che permette agli utenti di autenticarsi mediante i loro *Google Account* [Code.Google.com, 2011]. Tale libreria, attualmente in versione 1.0, esiste in due varianti, una per applicazioni web e una per applicazioni installate sul sistema operativo.

5.2.5 Sicurezza

Django offre un enorme supporto anche nell'ambito della sicurezza web, fornendo una serie di meccanismi automatici per la protezione del sistema da nu-

merosissimi tipi di attacco. L'attacco conosciuto con il nome di *SQL injection* è automaticamente gestito dal motore di Django: nel momento in cui Django interroga il database, nella query viene effettuato l'escape dei caratteri speciali di SQL. L'attacco *Cross-site Scripting* (XSS) può essere combattuto mediante l'escape esplicito di tutte le variabili contenenti dati provenienti dagli utenti. È possibile eseguire tale escape direttamente nei template di Django, apponendo la stringa `|escape` dopo ogni variabile che si voglia stampare a video. Ancora, l'attacco *Cross-site Request Forgery* (CSRF) può essere evitato mediante l'uso di buone tecniche di programmazione web: non salvando informazioni di sessione nelle URL; non salvando dati nei cookie ma nel database del server (il cookie conterrà soltanto l'ID della sessione); come prima, effettuando l'escape di dati presenti nella sessione. L'attacco denominato *man-in-the-middle* può essere mitigato mediante l'uso massiccio di HTTPS, anche per l'invio dei cookie di sessione. Un altro attacco, conosciuto col nome di *E-mail header injection*, può essere reso innocuo nello stesso modo in cui si combatte l'SQL injection, precedentemente descritto. Tutti questi metodi di protezione dai tipici attacchi web sono discussi ampiamente in questo articolo [Holovaty and Kaplan-Moss, 2009], dal quale abbiamo tratto spunto per la discussione presentata in questo paragrafo.

5.2.6 Trasformazioni XML-SQL

Il server deve essere in grado di effettuare due tipi di trasformazioni dei dati: in un primo caso una trasformazione da dati contenuti nel DBMS in un documento XML; nell'altro caso deve potere effettuare una trasformazione nell'altro verso, ovvero di dati XML in dati relazionali.

Il primo è il caso in cui un paziente richiede un questionario da compilare. Il server deve leggere nel database tutti i dati relativi al questionario e inviarli al client nel formato XML discusso precedentemente (cfr. 5.1.2 *XML per un questionario da compilare*). Ciò può essere effettuato molto facilmente mediante le tecnologie offerte da Django. Django infatti offre un potente motore di *template* che è in grado di produrre documenti XML ben formati a partire dai dati raccolti in un database. La trasformazione quindi avviene in maniera programmatica uti-

lizzando sia il linguaggio Python che il linguaggio specifico del motore di template di Django.

La trasformazione inversa avviene in due casi. Il primo caso è quando il paziente invia al server i risultati del questionario. Il server dovrà quindi leggere un file XML e inserire i relativi valori letti nel database. Tale operazione può essere programmata in modo molto semplice utilizzando un qualunque parser XML sviluppato in Python, direttamente nel codice lato server. Per esempio la libreria di funzioni Python `xml.tree.ElementTree` offre un modo estremamente semplice per visitare l'albero di un documento XML. Tale approccio è molto valido, se si considera soprattutto la semplicità della struttura dei dati XML in ingresso (cfr. 5.1.3 *XML per i dati di un questionario compilato*). L'ultimo caso in cui è necessario effettuare una trasformazione da XML in dati relazionali è quando si riceve un documento XML relativo a un nuovo questionario (cfr. 5.1.1 *XML per nuovo questionario*). Ma anche in tal caso la struttura del documento XML è alquanto semplice ed è quindi possibile utilizzare lo stesso metodo precedentemente descritto.

Infine, durante il parsing del documento XML, si creano delle query SQL per l'inserimento dei dati in modo permanente all'interno del database relazionale, in modo da completare il quadro sulle trasformazioni da XML in record.

5.3 I client

L'architettura di $\mathbb{P}$.ROse prevede la realizzazione di client che forniscano interfacce complesse e altamente interattive e che limitino le comunicazioni col server allo stretto indispensabile. Tali tipi di client vengono spesso denominati col termine di "fat client", a indicare il fatto che molta della logica applicativa è spostata sul client stesso. Nella progettazione dei client bisogna tenere presente numerosi aspetti: da un lato le diverse funzionalità che si vogliono sviluppare, come ad esempio l'esecuzione di un questionario o la visualizzazione dei risultati; da un altro lato i diversi tipi di client che verranno utilizzati, come i client web e le app; infine è bene considerare i requisiti di usabilità che ogni interfaccia presenta.

Come stabilito in sezione 4.3.1 i client interrogano il server per mezzo di HTTP e scambiano dati col lo stesso in formato XML. Alcune delle funzionalità dei client sono incentrate proprio su questi scambi. Ad esempio supponiamo che un paziente che stia usando la app su di un tablet voglia compilare un questionario che gli è stato somministrato. Nell'ordine, queste sono le interazioni necessarie tra il paziente e il suo tablet e fra il tablet e il server:

1. L'utente lancia la app di $\mathbb{P}$.ROse.
2. L'utente inserisce il proprio indirizzo email e la propria password per loggarsi.
3. Il client interroga il server per controllare che le credenziali di accesso siano valide.
4. Il client, in caso di login riuscito, mostra al paziente i questionari da compilare.
5. L'utente ne seleziona uno.
6. Il client chiede al server di inviargli un XML contenente la descrizione del questionario scelto dal paziente e lancia l'interfaccia di esecuzione dei questionari.
7. Una volta terminato il questionario, l'utente decide di inviare i dati al server.
8. Il client quindi impacchetta i risultati in un altro XML e li invia al server.
9. Il server risponde con una conferma di avvenuta ricezione.

In linea di principio un'interfaccia client web che gira su un pc desktop o laptop viene realizzata in XHTML, Javascript e CSS, una che gira su un browser per smartphone o tablet viene realizzata in HTML5, Javascript e CSS3. Una app invece viene solitamente creata per mezzo del linguaggio nativo del sistema operativo sul quale l'app dovrà girare. Per esempio una app per il sistema operativo Android viene normalmente scritta in Java, mentre una per il sistema

operativo iOS viene scritta in Objective-C. Quindi per garantire l'utilizzo dei servizi attraverso tutti questi dispositivi bisognerebbe programmare almeno quattro interfacce diverse. Fortunatamente oggi esistono nuove tecnologie mediante le quali è possibile limitare la scrittura del codice lato client in maniera sensibile e, in taluni casi, è addirittura possibile scrivere il codice una volta sola, per tutti i supporti elencati e altri ancora.

5.3.1 Compilazione di un questionario

Consideriamo innanzitutto il problema di realizzare l'interfaccia di compilazione di un questionario. Reputiamo sia importante avere la medesima interfaccia in ogni dispositivo sul quale il questionario verrà compilato[7]. Per realizzare questa interfaccia, senza doversi preoccupare troppo del dispositivo in cui verrà usato, utilizzeremo il framework *JQuery Mobile* [Jquerymobile.com, 2011]. Esso offre un ambiente integrato per la realizzazione di client web in HTML5 appositamente pensati per l'esecuzione su dispositivi touch-screen di qualunque tipo: dai cellulari, ai palmari ai tablet, siano essi basati su iOS, Android, BlackBerry, Windows Phone, ecc. JQuery Mobile è basata sulla vastissima e oggigiorno usatissima libreria Javascript denominata JQuery [Jquery.com, 2011], la quale permette, mediante la filosofia "write less, do more", di scrivere codice lato client molto complesso, graficamente ricco e molto usabile, in pochissime righe di codice.

Ciò che questa libreria mette a disposizione è molto interessante. Essa infatti non si limita a permette di programmare il codice eseguibile lato client, ma fornisce l'opportunità di governare aspetti grafici molto complessi in modo quasi del tutto automatico e di gestire gli eventi tipici dei dispositivi touch-screen, come ad esempio il "tap" sullo schermo, il trascinamento del dito a destra o a sinistra, ecc. Gli aspetti grafici sono introdotti mediante l'integrazione di tre tecnologie: HTML5, CSS e Javascript, tutte gestite automaticamente da JQuery Mobile. In tale framework, se si vuole inserire un oggetto grafico come un pulsante, un menu a tendina o un input testuale, basta inserire il *gadget* relativo e attribuirgli

[7] Ciò non era stato espressamente stabilito nei requisiti, ma crediamo che sia importante per una questione di coerenza e utile ai fini di semplificare lo sviluppo del software.

eventuali caratteristiche aggiuntive di personalizzazione. Anche la navigazione tra pagine è automaticamente gestita dal framework, il quale fornisce anche un metodo per la gestione della *history* del browser a seguito di chiamate Ajax, che è tipicamente problematica. Ultima caratteristica, ma anch'essa molto importante, è il fatto che un'applicazione realizzata in Jquery Mobile è perfettamente funzionante anche sulla maggior parte dei browser desktop. Ciò permette, quindi, di potere riutilizzare la stessa interfaccia di compilazione dei questionari anche su tali supporti.

Per poter eseguire un questionario su app, invece, utilizziamo una recente tecnologia che permette di convertire pagine realizzate in HTML5, CSS e Javascript in app native. Tale applicazione si chiama *PhoneGap* e permette di generare codice per moltissimi sistemi operativi mobili, tra cui iOS e Android e di accedere alle caratteristiche hardware e software native tipiche di tali dispositivi, come ad esempio accelerometri, videocamere, sistemi di geolocalizzazione, rete, ecc. In poche parole, grazie a questa tecnologia siamo in grado di riutilizzare il codice già scritto per i browser convertendolo in codice applicativo per supporti mobili, con il minimo sforzo [Phonegap.net, 2011].

Quanto stabilito in questo paragrafo soddisfa appieno i requisiti B1, B2, B3 e B4 dei browser web e A1 e A2 delle app.

Salvataggio temporaneo dei dati sul client

Per consentire al paziente di non perdere i dati, qualora dovessero esserci dei malfunzionamenti nella rete, è necessario permettere al client di salvare i dati temporaneamente nella sua memoria locale in modo da poterli inviare al server automaticamente in un secondo momento (cfr. requisito A8). Da un lato sembrerebbe auspicabile che il paziente rimanga ignaro di una tale eventualità, ma ciò darebbe problemi nel caso in cui il paziente, credendo che i dati siano già stati inviati al server, non riutilizzasse più il dispositivo per un lungo periodo di tempo. É quindi necessario che l'interfaccia di esecuzione, qualora non dovesse riuscire a inviare al server i dati e a ricevere dal server una conferma di ricezione, comunichi al paziente del fallimento dell'operazione e dell'avvenuto salvataggio

dei dati in locale. Il paziente potrà quindi controllare in un secondo momento se i dati siano stati spediti con successo o meno.

La memorizzazione in locale sulle app è immediata, in quanto si possono utilizzare comuni librerie di accesso al file system del dispositivo. Per quanto riguarda le applicazioni che girano su browser, invece, è possibile implementare tale caratteristica in vari modi. Il meno elegante è quello di utilizzare i cookie lato client. Un modo invece più interessante è quello di utilizzare la nuova tecnologia che va sotto il nome di *Web Storage*, per mezzo della quale è possibile memorizzare dati in modo permanente sul client, mediante coppie <*chiave,valore*> [W3C, 2011].

5.3.2 Altre interfacce utente

Abbiamo visto che è possibile quindi scrivere un'interfaccia client una volta sola, utilizzando HTML5, CSS e Javascript, e renderlo disponibile su un numero vastissimo di dispositivi client. Ci sono però alcune interfacce utente che non necessitano di essere eseguite sui dispositivi mobili, quali ad esempio le interfacce di creazione dei questionari o le interfacce per la visualizzazione complessa dei risultati. Tali interfacce verranno tipicamente accedute soltanto da browser desktop e pertanto non verranno realizzate con le tecnologie finora discusse. Si utilizzeranno invece XHTML, in quanto meglio supportato dalla maggior parte dei browser desktop oggi esistenti, CSS2 e Javascript. Quest'ultimo verrà codificato mediante l'uso della libreria JQuery, di cui abbiamo già brevemente accennato [Jquery.com, 2011], e attraverso una sua famosa estensione denominata *JQueryUI* [JQueryUI.com, 2011] che permette l'inserimento di effetti grafici, widget, animazioni di alta qualità in modo estremamente semplice.

Queste interfacce sono quelle tipicamente utilizzate dai dottori e sebbene presentino anch'esse requisiti di usabilità, tali requisiti non sono comunque fondamentali ai fini della validità di raccolta dei dati dai pazienti, come discusso in sezione 4.5.2 *Requisiti di usabilità*. Ad ogni modo, la tecnologia offerta da JQueryUI permette di realizzare interfacce grafiche altamente usabili con uno sforzo molto ridotto, anche per quanto concerne quelle caratteristiche più delicate co-

me la rappresentazione dei risultati per mezzo di grafici o la creazione di nuovi questionari.

5.3.3 Trasformazioni XML-HTML

Anche il client, come il server, necessita di effettuare delle trasformazioni sul formato dei dati. A differenza del server, però, il client effettua tipi diversi di trasformazione. Il client infatti riceve i questionari da compilare in formato XML, come descritto in 5.1.2 *XML per un questionario da compilare*, e deve utilizzare tale rappresentazione dei dati per modellare l'interfaccia di esecuzione dei questionari. Infine, una volta terminato il test, il client dovrà raccogliere i dati inseriti dai pazienti in un ulteriore documento XML, come descritto in 5.1.3 *XML per i dati di un questionario compilato*. La prima è una trasformazione da XML specifico in XHTML o HTML5, la seconda è esattamente l'opposto.

Entrambe queste trasformazioni possono essere eseguite in due modi. Il primo modo è quello di utilizzare un file di trasformazione XSLT (eXtensible Stylesheet Language Transformations), dal momento che sia l'input che l'output delle trasformazioni sono documenti XML. Il secondo metodo è quello di utilizzare un parser XML lato client per la navigazione del documento XML e infine adoperare funzioni Javascript per la manipolazione programmatica del DOM della pagina HTML. Per quanto concerne il secondo metodo è possibile utilizzare JQuery per entrambi gli step, ovvero sia per la navigazione dell'XML che per la manipolazione del documento HTML finale.

5.4 Possibili miglioramenti futuri

Il sistema fin qui presentato soddisfa la maggior parte dei requisiti funzionali presentati nel Capitolo 4 di questo elaborato. Tuttavia alcuni di essi non sono stati inclusi nelle scelte progettuali in quanto riteniamo che non siano necessari nelle primissime versioni. Il sistema è stato comunque progettato per consentire il loro facile inserimento, senza che ciò coinvolga la struttura stessa del sistema. Inoltre riteniamo che vi siano altri requisiti che potrebbero venire aggiunti al

sistema in fasi più avanzate, qualora i servizi venissero realmente usati dagli utenti finali. In questa sezione enunceremo tutte le caratteristiche non previste nell'attuale versione e indicheremo brevemente i modi per poterle implementare.

Utilizzo tramite sistemi client alternativi

Una delle caratteristiche a cui abbiamo brevemente accennato è la possibilità di aumentare il numero e la tipologia dei client che possano avere accesso ai servizi offerti dal server di ℙ.ROSE. In particolare abbiamo detto che sarebbe auspicabile che si possano compilare i questionari anche attraverso un sistema IVR (Interactive Voice Reponse), ovvero attraverso un telefono e la sua tastiera telefonica, o via SMS (Short Message Service). Le basi di dati e le interfacce client-server da noi proposte garantirebbero l'integrazione di tali servizi senza la necessità di fare alcuna modifica al server. Basterebbe infatti che si sviluppino dei client specializzati alla trasformazione dei documenti dal formato XML discusso al formato tipico dei servizi IVR o SMS. Più in generale, abbiamo visto come la struttura del server e le sue interfacce permettano l'integrazione dei servizi con una moltitudine di client non a priori stabilita, tra i quali vi potranno essere sistemi specializzati alla raccolta dei dati ePRO di un singolo ambito medico (com'è ad esempio il caso di ProClara).

Utilizzo congiunto con apparecchiature di misurazione

Abbiamo già parlato di questa eventualità nell'introduzione al capitolo dei requisiti. Qualora si volesse permette agli utenti di utilizzare hardware specializzato per la misurazione di valori biochimici dei pazienti, le modifiche necessarie sarebbero localizzate soltanto sui client, in quanto il sistema server è già predisposto ad accogliere tali dati, purché inviati nel corretto formato di scambio XML.

Semplificazione del metodo di visualizzazione dei risultati

Un'importante proprietà che abbiamo introdotto al punto S4.c dei requisiti funzionali del server prevedeva la possibilità, per un dottore, di creare "collezioni" di sue somministrazioni. Grazie a tali raggruppamenti un dottore potrebbe raffinare la granularità delle visualizzazioni dei risultati dei test, attraverso l'aggregazione sia dei risultati di uno stesso paziente, sia di quelli forniti da più pazienti su uno stesso test. Tale modifica richiederebbe la sola aggiunta al database di una tabella che serva a creare gruppi logici di somministrazioni (la tabella, quindi, avrebbe una chiave esterna verso *Givings*).

Inoltre sarebbe possibile migliorare il metodo di calcolo dei punteggi offrendo la possibilità di attribuire un punteggio numerico a ogni opzione delle domande dei questionari. In questo modo si potrebbero calcolare i punteggi finali dei test mediante l'uso di funzioni di calcolo a granularità maggiore rispetto a quelle in questo testo considerate (cfr. 5.1.1 *XML per nuovo questionario*). Per implementare questa modifica sarebbe necessaria l'aggiunta di un attributo *score* alla tabella *Options* e la conseguente aggiunta dello stesso attributo agli elementi `<option>` degli XML generati.

Un'ulteriore considerazione riguarda la visualizzazione grafica dei dati. Oltre alla libreria JQuery che abbiamo menzionato, sarebbe utile integrare l'utilizzo di librerie apposite che forniscano funzionalità avanzate di visualizzazione grafici. Una di queste è la libreria XML/SWF Charts [Maani.us, 2011] che permette la trasformazione di semplici file XML in grafici di ogni tipologia.

Creazione di una rete di condivisione tra dottori

Abbiamo più volte accennato alla possibilità di creare una rete di scambio dati tra i dottori del sistema (cfr. requisito M10). Ciò consentirebbe ai dottori che lavorano insieme a un certo trial di potersi scambiare in maniera automatica i dati raccolti e permetterebbe, inoltre, di creare gruppi di lavoro e ricerca a distanza. Tale caratteristica, tuttavia, non è stata inclusa nella progettazione in quanto non si è ritenuta necessaria per la sua prima versione. É stato però previsto lo spazio

per la sua aggiunta: sarebbe infatti possibile creare tale rete sociale mediante l'utilizzo di un'ulteriore relazione dottore-dottore. Anche qui sarebbe necessario introdurre misure di sicurezza atte a proteggere i dati dei pazienti. Bisognerebbe principalmente evitare che i dottori possano scambiarsi i dati dei pazienti senza la loro autorizzazione. Sarà possibile implementare ciò mediante l'uso di speciali permessi: Django offre all'interno del suo framework un modulo di gestione dei permessi molto elegante e semplice.

Possibilità di saltare una domanda

Una piccola modifica potrebbe consentire a un paziente di saltare una domanda di un questionario. Per implementare ciò basterebbe aggiungere un attributo *required* nella tabella *Questions* e al corrispettivo elemento XML. Tale attributo sarebbe *true* di default, settabile a *false* se si volesse permettere all'utente di non fornire una risposta per una particolare domanda. L'interfaccia lato client dovrebbe quindi fornire il link alla domanda successiva prima ancora di ricevere l'input da parte dell'utente.

Aggiunta delle versioni ai questionari

Abbiamo già visto che un questionario che sia stato già somministrato a un paziente non può essere modificato nella sua struttura, poiché ciò potrebbe rendere i dati già raccolti per esso non più coerenti con la nuova versione. Un'importante modifica dovrebbe permettere ai dottori di creare nuove versioni dello stesso questionario in modo tale da poter mantenere i dati precedentemente raccolti e al tempo stesso consentire la modifica dei questionari qualora ritenuto necessario. Una modifica del genere potrebbe essere implementata in due modi distinti: 1) mediante l'utilizzo di un attributo *version* nella tabella *Questionnaires*, con l'accortezza di usare come chiave primaria l'aggregazione dei due attributi *quid* e *version*; 2) mediante l'aggiunta di una nuova tabella che memorizzi soltanto le versioni dei questionari. Entrambi i metodi, comunque, richiedono minime modifiche al server, ben localizzate.

RESTfullness

Il requisito S4 prevedeva che il sistema permettesse l'accesso alle risorse mediante un'architettura di tipo REST, grazie alla quale si sarebbe consentito a un qualunque client di ricevere e operare modifiche sulle risorse attraverso l'uso dei quattro metodi del protocollo HTTP. Tuttavia, la nostra progettazione ha considerato l'utilizzo dei soli metodi GET e POST, in quanto sono i soli a essere largamente supportati dai browser. In uno scenario più grande, nel quale si preveda che il nostro server possa essere interrogato direttamente da programmi, per mezzo di terminali o ancora da altri server nella rete, sarebbe necessario implementare le funzionalità server anche per i metodi PUT e DELETE, e modificare la rappresentazione dei dati per i metodi già implementati. A tal proposito esistono parecchie librerie di supporto a Django, che permettono con poco sforzo di aggiungere tale caratteristiche al proprio server. Citiamo, tra tutte, la libreria *Django REST framework* [Christie, 2011], la quale consentirebbe di soddisfare anche il requisito S3, il quale implicitamente prevedeva l'accesso alle risorse con le medesime URL.

Creazione dei questionari da parte di amministratori

Nel corso di questa trattazione è stata fatta l'assunzione che debbano essere soltanto i medici a creare i questionari. Ciò non è in realtà del tutto ammissibile, in quanto è poco probabile che in uno scenario reale i medici abbiano il tempo necessario per occuparsi continuamente della creazione dei questionari e della loro eventuale modifica. Una piccolissima modifica al sistema prevederebbe l'aggiunzione di speciali utenti, i cosiddetti *amministratori* del sistema, che si occupino della creazione dei questionari per conto dei dottori.

Capitolo 6

Test e analisi

Il lavoro finora svolto ha stabilito un framework il più possibile completo ed esaustivo per la realizzazione di tutti i servizi che $\mathbb{P}$.ROse si prefigge di fornire. La realizzazione finale del sistema passa però attraverso una serie di fasi intermedie in cui si realizzano piccoli pezzi del sistema, si testano e infine si fondono insieme. Per questo lavoro di tesi abbiamo realizzato un semplice prototipo di interfaccia di esecuzione dei questionari, che ci ha permesso di testare alcune delle funzionalità del sistema: da un lato è stato possibile testare la robustezza delle basi di dati che abbiamo progettato; è stato inoltre possibile verificare il raggiungimento dei requisiti di usabilità che abbiamo ampiamente discusso; è servito a testare i meccanismi di interazione client-server nel caso dello scambio di dati in formato XML e infine a verificare la robustezza (e l'incredibile semplicità) del sistema server implementato mediante l'utilizzo di Django. Nel seguito di questo breve capitolo enunceremo tali risultati.

6.1 Prototipo di esecuzione di un questionario

L'obiettivo del prototipo che abbiamo sviluppato non è quello di implementare tutte le funzionalità finali che il servizio fornirà a medici e pazienti bensì un suo ristretto sottoinsieme. Abbiamo deciso di concentrare gli sforzi sulla realizzazione di un prototipo che mostri come sia possibile realizzare l'interfaccia di esecuzione

dei test mediante l'utilizzo delle tecnologie discusse, in particolare JQuery Mobile, HTML5 e CSS, con particolare cura dei requisiti di usabilità e che soddisfi la modalità di trasformazione dei dati relazionali in XML e dei dati XML in HTML5.

Il prototipo è quindi una piccola parte del sistema $\mathbb{P}$.ROse, implementata utilizzando tutte e sole le tecnologie discusse nella presente trattazione. In particolare il prototipo permette a un paziente di:

- Accedere al sistema attraverso un qualunque browser, sia esso desktop, laptop, mobile (tra cui tablet, smartphone, ecc.).
- Autenticarsi mediante username e password.
- Visualizzare la lista dei questionari da compilare.
- Lanciare l'esecuzione di uno dei questionari presenti nella lista.
- Compilare il questionario domanda per domanda, visualizzare un riepilogo delle risposte date e, infine, inviare i dati al server.

6.1.1 Test sul database relazionale

Il primo passo per la realizzazione del prototipo è l'implementazione del modello dei dati discusso in sezione 5.2.2 *Il database*. Il codice Python fornito in quella sezione è esattamente quello che abbiamo utilizzato per implementare i modelli dei dati in Django. Una volta scritte le classi relative al modello relazionale è possibile configurare Django in modo che possa comunicare con il server del DBMS di PostgreSQL. Fatto ciò è già possibile lanciare il server web di prova fornito da Django stesso, mediante lo script `python manage.py runserver` che mette in ascolto il server sulla porta 8000.

A questo punto vi sono almeno due modi per interrogare il database. É possibile utilizzare la shell di Python (o uno script Python, alternativamente) per istanziare oggetti corrispondenti alle classi definite nei modelli dei dati, oppure utilizzare la shell di PostgreSQL e il linguaggio SQL in maniera diretta. Il primo metodo è più generale e astrae della caratteristiche specifiche del dialetto SQL

che si sta utilizzando, in modo da rendere possibile, tra le altre cose, il riutilizzo del codice qualora si volesse cambiare DBMS.

Col metodo appena descritto abbiamo quindi popolato il database con dati iniziali di prova. In particolare abbiamo creato cinque utenti di cui tre pazienti e due dottori, e un questionario identico a quello discusso precedentemente, nelle sezioni 5.1.1 *XML per nuovo questionario* e 5.2.2 *Modellazione dei questionari*, la cui struttura gerarchica è stata mostrata in Figura 5.1. Infine abbiamo simulato la somministrazione del questionario da parte del secondo dottore a due suoi pazienti, nel seguente modo: al primo paziente il questionario viene somministrato due volte con una validità di circa tre mesi, ma con orari di scadenza differenti; al secondo paziente si somministra il test una volta sola con una validità di un solo giorno in una data futura.

Mostriamo qui di seguito come è possibile creare la struttura appena descritta mediante l'utilizzo del linguaggio di programmazione Python in Django, omettendo la creazione degli utenti in *Users* in quanto non molto interessante. L'esempio, inoltre, mostra come il questionario (qui denominato `quest`) venga realizzato in due lingue: italiano e inglese. Ciò è possibile grazie alle tabelle relazionali introdotte nel modello dei dati, che permettono di memorizzare le informazioni testuali specifiche di ogni lingua e collegarle agli oggetti presenti nelle altre tabelle cui si riferiscono[1].

```
# Creazione del questionario
quest = Questionnaire(created_by=d2, description='A ePRO about pain')

# Aggiunta delle informazioni in lingua
Questionnaire_Lang(quid=quest, lang='it', title='Questionario sul dolore', info='...')
Questionnaire_Lang(quid=quest, lang='en', title='Pain questionnaire', info='...')

# Aggiunta delle domande
q1 = Question(quid=quest, type='o', left_val=1, right_val=1)
q2 = Question(quid=quest, type='o', left_val=0, right_val='NaN')
q3 = Question(quid=quest, type='b', left_val=1, right_val=2)
q4 = Question(quid=quest, type='n', left_val=100, right_val=200)
q5 = Question(quid=quest, type='r', left_val=100, right_val=200)
q6 = Question(quid=quest, type='s', left_val=0, right_val=200)

# Aggiunta delle informazioni in lingua per ogni domanda
```

[1]Il questionario qui mostrato è puramente esemplificativo, ovvero non corrisponde a nessun test `ePRO` realmente esistente.

```
Question_Lang(qid=q1, lang='it', string='Che livello di dolore prova?')
Question_Lang(qid=q2, lang='it', string='Quando ha provato dolore última volta?')
Question_Lang(qid=q3, lang='it', string='Ha misurato la glicemia oggi?')
Question_Lang(qid=q4, lang='it', string='Valore di glicemia registrato')
Question_Lang(qid=q5, lang='it', string='Inserisca il valore della pressione')
Question_Lang(qid=q6, lang='it', string='Scriva la parola "casa"')
Question_Lang(qid=q1, lang='en', string='How much pain do you feel?')
Question_Lang(qid=q2, lang='en', string='When did you feel pain last time?')
Question_Lang(qid=q3, lang='en', string='Did you measure your glycaemia today?')
Question_Lang(qid=q4, lang='en', string='Please, insert the recorded glycaemia value')
Question_Lang(qid=q5, lang='en', string='Please, insert the blood pressure value')
Question_Lang(qid=q6, lang='en', string='Please, insert the word "casa"')

# Aggiunta delle opzioni (per le domande opzionali)
Option(qid=q1, lang='it', description='Nessun dolore')
Option(qid=q1, lang='it', description='Dolore moderato')
Option(qid=q1, lang='it', description='Dolore molto forte')
Option(qid=q1, lang='en', description='No pain, no gain')
Option(qid=q1, lang='en', description='Moderate pain')
Option(qid=q1, lang='en', description='Strong pain')

Option(qid=q2, lang='it', description='Láltro ieri')
Option(qid=q2, lang='it', description='Ieri')
Option(qid=q2, lang='it', description='Oggi')
Option(qid=q2, lang='en', description='Two days ago')
Option(qid=q2, lang='en', description='Yesterday')
Option(qid=q2, lang='en', description='Today')

Option(qid=q3, lang='it', description='Sì')
Option(qid=q3, lang='it', description='No')
Option(qid=q3, lang='en', description='Yes')
Option(qid=q3, lang='en', description='No')

# Somministrazione del questionario ai due pazienti
Giving(did=d2, pid=p1, quid=quest, from_date='2011-10-01 08:00', to_date='2012-01-01 20:00')
Giving(did=d2, pid=p1, quid=quest, from_date='2011-10-01 08:00', to_date='2012-01-01 10:00')
Giving(did=d2, pid=p2, quid=quest, from_date='2012-02-10 13:00', to_date='2012-02-10 23:59')
```

Il questionario utilizzato è particolarmente esemplificativo in quanto comprende tutti i tipi possibili di domanda. Il test mostra anche un esempio di doppia somministrazione a un paziente (il paziente `p1`) e un esempio di somministrazione differita (al paziente `p2`, se consideriamo che il questionario è stato "somministrato" nel mese di Ottobre 2011).

Questa struttura è stata quindi testata su parecchie query per verificarne la coerenza e i risultati sono stati più che soddisfacenti. Un problema che tuttavia conviene riportare è legato alla gestione delle lingue. Il modello stesso dei dati, infatti non impone che vengano offerte traduzioni per ogni elemento di un questionario. Ciò deve essere garantito dal livello applicativo del server il quale deve verificare, prima di inviare un questionario a un client, che siano presenti

tutte le traduzioni nella lingua da lui richiesta. Il risultato sarebbe, altrimenti, un questionario manchevole di alcune stringhe testuali.

É bene notare, inoltre, che finché non verrà implementato il concetto di "versione di un questionario" non sarà possibile modificare la struttura di un questionario che sia già stato somministrato anche solo una volta, in quanto una modifica rovinerebbe la coerenza dei risultati già ottenuti per quel questionario (cfr. 5.4 *Possibili miglioramenti futuri*).

6.1.2 Effettività del metodo di trasformazione dei dati

É stato possibile verificare il risultato ottenuto dalle due trasformazioni principali: quella che trasforma un questionario contenuto nel DBMS in XML e quella che trasforma, lato client, quello stesso XML in HTML. Il risultato della prima trasformazione è proprio quello riportato in Appendice B e che abbiamo già discusso precedentemente. La corrispondenza tra la struttura relazionale ed esso è evidente, in quanto la struttura dei dati memorizzati nel DBMS è essa stessa gerarchica. Infatti, sin dall'inizio abbiamo modellato i questionari proprio con una struttura ad albero di massimo tre livelli (in cui la radice è il questionario stesso, il primo livello è caratterizzato dalle domande e il terzo, eventuale, livello contiene le opzioni alle domande, come in Figura 5.1).

La trasformazione dei dati dal formato XML al formato HTML è stata implementata mediante JQuery, con semplici funzione di visita dell'albero del documento. Anche in questo caso è stato possibile ottenere una trasformazione immediata, sempre per la struttura stessa dei questionari. Inoltre, sono state sfruttate alcune funzionalità di JQuery Mobile che permettono di includere più pagine logiche all'interno di una stessa pagina (mediante l'utilizzo di diversi `<div data-role="page">` nella stessa pagina) e che vengono automaticamente navigate dal motore di JQuery Mobile come fossero pagine diverse, senza la necessità di re-interrogare il server ogni volta che si cambia pagina. In tal modo, per ogni `<question>` del documento XML è stato possibile creare una pagina logica all'interno del documento HTML5 risultante.

6.1.3 Verifica delle qualità di usabilità

Al fine di testare il sistema è stata realizzata un'interfaccia per la navigazione e compilazione di un questionario `ePRO`, che sia stato memorizzato nel database come precedentemente descritto. In Appendice C è possibile vedere delle schermate di esecuzione. La prima schermata mostra la lista di questionari che il paziente correntemente loggato deve compilare. Per ogni questionario è mostrato il titolo, la scadenza e il dottore che l'ha somministrato. In basso a sinistra è possibile notare il selettore di lingua. Nella seconda schermata è mostrata la prima domanda del questionario, che è di tipo opzionale a scelta unica. É bene notare come non sia possibile proseguire verso le altre domande finché non sia stata effettua una scelta. La schermata immediatamente successiva mostra cosa accade quando si effettua una scelta: in basso a destra compare un link per spostarsi alla domanda successiva. L'immagine dopo mostra una domanda a risposta opzionale nella quale è possibile selezionare un numero qualunque di risposte. Infine, l'ultima schermata, mostra un esempio di domanda con range numerico, nella quale l'utente può muovere la barra orizzontale o scrivere direttamente il numero richiesto nel campo di input. É interessante constatare come tale interfaccia implementi tutti i requisiti di usabilità discussi nel presente elaborato.

L'interfaccia di compilazione del questionario è stata testata da cinque utenti cui sono state richieste valutazioni dei seguenti tre aspetti, tutti in scale da 1 a 10:

1. Facilità d'uso, specialmente in relazione al tempo necessario a imparare a utilizzare il sistema.

2. Adeguatezza all'uso da parte di pazienti di qualunque tipo.

3. Grado di fiducia che l'utente sentirebbe di avere se usasse il sistema in un contesto reale, ovvero fornendo i propri dati clinici.

Nessuno degli utenti presi in esame era esperto del settore informatico e nessuno di loro aveva mai visto il sistema prima del test. In particolare uno degli utenti selezionati è un medico ed ha potuto fornirci una visione un po' più ampia sul

secondo criterio di valutazione. In tutti i test è stato chiesto agli utenti di eseguire, nell'ordine, le seguenti operazioni:

1. Raggiungere una URL tramite il proprio browser.
2. Inserire username e password fornite.
3. Compilare uno dei questionari presenti nel sistema, fino al raggiungimento della pagina contenente il riepilogo delle risposte date.

Riguardo all'ultimo punto in particolare non sono state date altre delucidazioni, proprio per rendere il test di usabilità più effettivo. Non è stato spiegato né come si fa a selezionare un questionario, né come si risponde alle varie domande o come si naviga tra una domanda e un'altra. I risultati ottenuti dai test hanno dato una media di 9 sulla facilità d'uso, 8.6 sulla adeguatezza e 8.4 sulla fiducia. In particolare è stato notato che il sistema per la navigazione tra le domande ha richiesto qualche attimo in più per essere trovato essendo stato inserito nella parte bassa dello schermo e per il fatto che, per alcune domande, è richiesto l'input da parte dell'utente prima di poter proseguire nel test. Il grado di fiducia nel sistema in taluni casi ha ricevuto un giudizio basso in quanto si reputa che un sistema del genere possa essere, in certi contesti, poco adatto al trattamento di dati così importanti e sensibili come lo sono quelli dei pazienti.

Riteniamo che i risultati di questo test preliminare siano più che soddisfacenti, specialmente se consideriamo che nessuno degli utenti è stato istruito su come compilare il questionario. Ciò dimostra che in linea di principio l'interfaccia di esecuzione dei questionari è molto intuitiva e autoesplicativa. Usata in contesti più grandi ci si aspetta si a necessario un tempo più che breve affinché i pazienti possano imparare a usarla. Sarà necessario comunque sviluppare ulteriori test, questa volta prendendo a campione una più vasta popolazione e selezionandola in base a età, esperienza informatica e problemi fisici quali ad esempio disfunzioni della vista o impedimenti medioleggeri nell'uso delle apparecchiature elettroniche.

Capitolo 7

Conclusioni

La progettazione di un sistema per il trattamento dei dati clinici richiede in primis un'attenta analisi dei requisiti che passi attraverso lo studio degli ambiti specifici e della modalità precisa con cui esso verrà usato, e infine una cura particolare ai temi legati all'usabilità delle interfacce e alla sicurezza dei dati. Tutto ciò deve essere unito a buone tecniche di progettazione, che garantiscano la realizzazione di un sistema ben modularizzato e coerente, mantenibile e verificabile. In questa trattazione abbiamo visto come sia possibile realizzare un'applicazione web per la gestione di generici questionari elettronici atti a essere usati per raccogliere dati sulla qualità della vita, la salute, lo stato di una malattia e molto altro, direttamente dai pazienti. Il sistema che abbiamo progettato si pone l'obiettivo di memorizzare e dare accesso ai dati così generati e a fornire interfacce intuitive e facili all'uso che possano essere usate dai pazienti in tutti quei contesti dove è necessario un feedback diretto da essi. Infine, per mezzo di un semplice prototipo e un breve test abbiamo ottenuto una prima osservazione sulla qualità dell'approccio e delle tecnologie usate, che ci ha fornito un incentivo a continuare il lavoro finora svolto e a sviluppare tutte le componenti finali del sistema.

Concludiamo accennando a quelli che reputiamo i possibili modi di introduzione del prodotto finale nel mercato, così da rendere meglio l'idea dell'effettiva attuabilità e concreta utilità del qui presente progetto. Crediamo che il miglior modo per poter cominciare a usare il sistema in reali ambiti clinici sia quello di

procedere attraverso una fase di testing e verifica del software e delle sue interfacce in situazioni concrete, con l'appoggio di dottori, cliniche o ospedali disposti a collaborare per il progetto. Solo attraverso questa fase sarà possibile evidenziare eventuali pecche del sistema e migliorarlo in tutti i suoi aspetti. La collaborazione con i medici e i ricercatori, infine, garantirà una maggiore adeguatezza del sistema alle loro reali esigenze. Essa potrà eventualmente passare attraverso alcune fasi di rivisita dei requisiti, dell'architettura del sistema, delle scelte tecnologiche o di refactoring. Una volta terminata questa fase sarà possibile sfruttare i risultati prodotti per proporre il sistema a nuovi ospedali e cliniche, nella speranza che il nostro sistema possa infine ottenere un bacino consistente di utenza.

Ringraziamenti

Questo è verosimilmente il capitolo con la minore probabilità di venire letto, se non da parenti e amici i quali, avendolo visto elencato nell'indice, siano saltati a piè pari alla presente pagina, cercando di vedere se l'autore si sia ricordato di menzionarli, ringraziarli, ricordarli. In realtà, le persone che dovrei ringraziare sono molte. No, sono troppe. Questa tesi di laurea è soltanto l'appendice di ciò che è stato un lungo cammino di crescita, che mi ha portato innanzitutto a iscrivermi all'Università di Bologna e infine a trovarmi qui oggi. Ciò che per me conta non è la laurea in sé, ma l'aver percorso la strada che ora mi vedo alle spalle e ciò non sarebbe stato mai possibile senza l'aiuto di moltissime persone (molte delle quali stanno proprio leggendo queste righe[1]). In questo breve paragrafo proverò ad andare un po' a ritroso nel tempo e ricordare tutte le persone che, grazie al loro aiuto, mi hanno permesso di ottenere questo risultato.

Un ringraziamento va innanzitutto al Professor Danilo Montesi, relatore di questa tesi, che mi ha permesso di utilizzare le mie competenze informatiche in un ambito che reputo socialmente utile. Ringrazio molto il correlatore, il Professore Fabio Vitali, per i suoi preziosissimi consigli sia durante le prime fasi del lavoro che nella sua tappa conclusiva. Ringrazio il Professor Fabio Panzieri per le delucidazioni circa i problemi di sicurezza che le soluzioni cloud computing oggi presentano. Un ringraziamento speciale va ai colleghi Andrea Aquino e Miro Mannino per avermi fornito utili visioni su alcuni dei problemi da me incontrati, e per essere stati ottimi colleghi con cui studiare e confrontarsi durante il triennio. Reputo di essere stato molto fortunato in questo. Vorrei ringraziare anche i

[1]Reputo molto probabile che chi mi abbia aiutato e supportato nel corso di questi anni si ritrovi qui adesso a leggere il capitolo dei ringraziamenti.

colleghi Seraj Abu-Seraj, Musa Rayyan, Nada Hashem e Ashwin Gopalakrishnan, i quali non leggeranno mai questa tesi per intero se non gliela traduco, ma che mi hanno molto aiutato durante il mio anno negli Stati Uniti.

Vorrei ringraziare il Professore Simone Martini per avermi aiutato (e incoraggiato) a raggiungere l'obiettivo di trascorrere un anno accademico alla University of California, un'esperienza che mi ha dato moltissimo e che è indirettamente legata anche alla scelta dell'argomento di questa tesi. Andando ancora indietro nel tempo, vorrei ringraziare il Professor Cesare Parenti, per avere riacceso il mio entusiasmo nei confronti della Matematica.

Un grazie infinito va a mia sorella Chiara, per avere sempre (e testardamente) creduto in me e al mio "brother-in-law" Fabio, per avermi aperto un'orizzonte sulle possibilità di prosecuzione degli studi dopo la laurea. Un grazie ai miei genitori, Silvia e Andrea, per avermi sempre concesso l'opportunità di fare quello che mi piaceva. Mi scuso con loro per averli fatti più volte preoccupare in passato. Ringrazio infine Consuelo, la mia compagna, per essermi stata vicina esattamente nel modo in cui avevo bisogno. Nonostante non le piaccia la Matematica e l'annoi l'Informatica, l'essere arrivato qui è per metà merito suo.

Bibliografia

[Allenby et al., 2002] Allenby, A., Matthews, J., Beresford, J., and McLachlan, S. (2002). The application of computer touch-screen technology in screening for psychosocial distress in an ambulatory oncology setting. *European Journal of Cancer Care*, 11(4):245–253.

[Amazon.com, 2011] Amazon.com (2011). Amazon elastic compute cloud (amazon ec2). `http://aws.amazon.com/ec2/`. Accessed October 17, 2011.

[AOL.com, 2011] AOL.com (2011). Aol. `http://www.aol.com/`. Accessed October 23, 2011.

[Apple.com, 2011] Apple.com (2011). Apple ipad 2. `store.apple.com/it/ipad`. Accessed October 22, 2011.

[Aquino, 2011] Aquino, A. (2011). Geco. Tesi di laurea in Informatica, discussa all'Alma Mater Studiorum - Università di Bologna, nel mese di Luglio 2011.

[Arrowhead Electronic Healthcare, 2011] Arrowhead Electronic Healthcare (2011). Multi-method epro. `http://www.aheh.com/`. Accessed October 23, 2011.

[CDC, 2011] CDC (2011). Health-related quality of life (hrqol). `http://www.cdc.gov/hrqol/hrqol14_measure.htm`. Accessed October 2, 2011.

[Christie, 2011] Christie, T. (2011). Django rest framework. `http://django-rest-framework.org/`. Accessed November 5, 2011.

[Christodorescu et al., 2009] Christodorescu, M., Sailer, R., Schales, D. L., Sgandurra, D., and Zamboni, D. (2009). Cloud security is not (just) virtualization security: a short paper. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, pages 97–102, New York, NY, USA. ACM.

[Code.Google.com, 2011] Code.Google.com (2011). Authentication and authorization for google apis: Oauth 1.0 for web applications. `http://code.google.com/intl/it-IT/apis/accounts/docs/OAuth.html`. Accessed October 23, 2011.

[Djangoproject.com, 2011] Djangoproject.com (2011). Django. `https://www.djangoproject.com/`. Accessed October 17, 2011.

[Eun-Hyun and Lee, 2009] Eun-Hyun and Lee (2009). Touch-screen computerized quality-of-life assessment for patients with cancer. *Asian Nursing Research*, 3(1):41 – 48.

[Fairbanks et al., 1980] Fairbanks, J., Couper, C., Davies, J., and O'Brien, J. (1980). The oswestry low back pain disability questionnaire. *Physiotherapy*, 66:271–273.

[Fayers and Machin, 2007] Fayers, P. and Machin, D. (2007). *Quality of life: the assessment, analysis and interpretation of patient-reported outcomes*. John Wiley & Sons Ltd.

[Fielding and Taylor, 2002] Fielding, R. T. and Taylor, R. N. (2002). Principled design of the modern web architecture. *ACM Transactions on Internet Technology*, 2:115–150.

[Food and Drug Administration, 2006] Food and Drug Administration (2006). Guidance for industry: patient-reported outcome measures: use in medical product development to support labeling claims: draft guidance. *Health and Quality of Life Outcomes*, 4(1):79.

[Garrett, 2005] Garrett, J. J. (2005). Ajax: A new approach to web applications. http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications. AdaptivePath.com.

[Gliffy.com, 2011] Gliffy.com (2011). Gliffy. http://www.gliffy.com/. Accessed October 23, 2011.

[Google.com, 2011] Google.com (2011). Google. http://www.google.com/. Accessed October 23, 2011.

[Guyatt et al., 1993] Guyatt, G. H., Feeny, D. H., and Patrick, D. L. (1993). Measuring health-related quality of life. *Annals of Internal Medicine*, 118(8):622–629.

[Gwaltney et al., 2008] Gwaltney, C. J., Shields, A. L., and Shiffman, S. (2008). Equivalence of electronic and paper-and-pencil administration of patient-reported outcome measures: A meta-analytic review. *Value in Health*, 11(2):322–333.

[Hayes, 2008] Hayes, B. (2008). Cloud computing. *Communications of the ACM*, 51:9–11.

[Holovaty and Kaplan-Moss, 2009] Holovaty, A. and Kaplan-Moss, J. (2009). The django book: Security. http://www.djangobook.com/en/beta/chapter20/. Accessed October 29, 2011.

[Hopper et al., 1996] Hopper, E., Cameron, C.-A., and B., T. (1996). The use of a personal digital assistant to administer visual analogue scales. *Journal of Psychopharmacology*, 10:A27.

[IEEE, 1998] IEEE (1998). IEEE Recommended Practice for Software Requirements Specifications. Technical report.

[Invivo Data, 2011] Invivo Data (2011). Diarypro ediary solution. http://www.invivodata.com/solutions/diarypro-ediary/. Accessed October 23, 2011.

[Jquery.com, 2011] Jquery.com (2011). jquery. http://jquery.com/. Accessed October 22, 2011.

[Jquerymobile.com, 2011] Jquerymobile.com (2011). jquery mobile. http://jquerymobile.com/. Accessed October 22, 2011.

[JQueryUI.com, 2011] JQueryUI.com (2011). Jqueryui. http://jqueryui.com/. Accessed October 29, 2011.

[Kadambi et al., 2011] Kadambi, S., Chen, J., Cooper, B. F., Lomax, D., Ramakrishnan, R., Silberstein, A., Tam, E., and Garcia-Molina, H. (2011). Where in the world is my data? *Proceedings of VLDB*.

[Kandukuri et al., 2009] Kandukuri, B., Paturi, V., and Rakshit, A. (2009). Cloud security issues. In *IEEE International Conference on Services Computing, 2009. SCC '09.*, pages 517 –520.

[Lawrence et al., 2010] Lawrence, S. T., Willig, J. H., Crane, H. M., Ye, J., Aban, I., Lober, W., Nevin, C. R., Nevin, C. R., Batey, D. S., Mugavero, M. J., McCullumsmith, C., Wright, C., Kitahata, M., Raper, J. L., Saag, M. S., and Schumacher, J. E. (2010). Routine, self-administered, touch-screen, computer- based suicidal ideation assessment linked to automated response team notification in an hiv primary care setting. *Clinical Infectious Diseases*, 50(8):1165–1173.

[Leff and Rayfield, 2001] Leff, A. and Rayfield, J. (2001). Web-application development using the model/view/controller design pattern. In *Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International*, pages 118 –127.

[Maani.us, 2011] Maani.us (2011). Xml/swf charts. http://www.maani.us/. Accessed November 5, 2011.

[Migliorino, 2011] Migliorino, G. (2011). Proclara per i medici: l'app gratuita per misurare l'artrite reumatoide. http://www.ipaditalia.com/. Accessed October 23, 2011.

[OAuth.net, 2011] OAuth.net (2011). Oauth: An open protocol to allow secure api authorization in a simple and standard method from desktop and web applications. `http://oauth.net/`. Accessed October 23, 2011.

[OpenID Enabled, 2011] OpenID Enabled (2011). Openid enabled. `http://www.janrain.com/openid-enabled`. Accessed October 23, 2011.

[Osterhaus et al., 2002] Osterhaus, J. T., Dedhiya, S. D., Ernst, M. E., Osterhaus, M., Mehta, S. S., and Townsend, R. J. (2002). Health outcomes assessment in community pharmacy practices: A feasibility project. *Arthritis Care & Research*, 47(2):124–131.

[Palmblad and Tiplady, 2004] Palmblad, M. and Tiplady, B. (2004). Electronic diaries and questionnaires: Designing user interfaces that are easy for all patients to use. *Quality of Life Research: An International Journal of Quality of Life Aspects of Treatment, Care & Rehabilitation*, 13(7).

[Patrick et al., 2007] Patrick, D. L., Burke, L. B., Powers, J. H., Scott, J. A., Rock, E. P., Dawisha, S., O'Neill, R., and Kennedy, D. L. (2007). Patient-reported outcomes to support medical product labeling claims: Fda perspective. *Value in Health*, 10:S125–S137.

[Phillips and Davis, 2009] Phillips, A. and Davis, M. (2009). Rfc 5646: Tags for identifying languages. `http://tools.ietf.org/html/rfc5646`.

[Phonegap.net, 2011] Phonegap.net (2011). Phonegap: The only open source mobile framework that supports 7 platforms. `http://phonegap.com/`. Accessed October 23, 2011.

[PHT, 2011] PHT (2011). Pht firsthand knowledge. `http://www.phtcorp.com/`. Accessed October 23, 2011.

[Rebollo et al., 2010] Rebollo, P., Castejon, I., Cuervo, J., Villa, G., Garcia-Cueto, E., Diaz-Cuervo, H., Zardain, P., Muniz, J., Alonso, J., and the Spanish CAT-Health Research Group (2010). Validation of a computer-adaptive test

to evaluate generic health-related quality of life. *Health and Quality of Life Outcomes*, 8(1):147.

[Recordon and Reed, 2006] Recordon, D. and Reed, D. (2006). Openid 2.0: a platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital identity management*, DIM '06, pages 11–16, New York, NY, USA. ACM.

[Ronald and Melzack, 1975] Ronald and Melzack (1975). The mcgill pain questionnaire: Major properties and scoring methods. *PAIN*, 1(3):277 – 299.

[Rothman et al., 2007] Rothman, M. L., Beltran, P., Cappelleri, J. C., Lipscomb, J., Teschendorf, B., and the Mayo/FDA Patient-Reported Outcomes Consensus Meeting Group (2007). Patient-reported outcomes: Conceptual issues. *Value in Health*, 10:S66–S75.

[Salaffi et al., 2009] Salaffi, F., Gasparini, S., and W., G. (2009). The use of computer touch-screen technology for the collection of patient-reported outcome data in rheumatoid arthritis: comparison with standardized paper questionnaires. *Clinical and experimental rheumatology*, 27(3):459–68.

[Soley and the OMG Staff Strategy Group, 2000] Soley, R. and the OMG Staff Strategy Group (2000). Model driven architecture. *Object Management Group*.

[Stone et al., 2002] Stone, A. A., Shiffman, S., Schwartz, J. E., Broderick, J. E., and Hufford, M. R. (2002). Patient non-compliance with paper diaries. *BMJ*, 324(7347):1193–1194.

[Stone et al., 2003] Stone, A. A., Shiffman, S., Schwartz, J. E., Broderick, J. E., and Hufford, M. R. (2003). Patient compliance with paper and electronic diaries. *Controlled Clinical Trials*, 24(2):182 – 199.

[Taieb et al., 2011] Taieb, D., Baumstarck-Barrau, K., Sebag, F., Fortanier, C., De Micco, C., Loundou, A., Auquier, P., Palazzo, F., Henry, J.-f., and Mundler, O. (2011). Heath-related quality of life in thyroid cancer patients following radioiodine ablation. *Health and Quality of Life Outcomes*, 9(1):33.

[Velikova et al., 2004] Velikova, G., Booth, L., Smith, A. B., Brown, P. M., Lynch, P., Brown, J. M., and Selby, P. J. (2004). Measuring quality of life in routine oncology practice improves communication and patient well-being: a randomized controlled trial. *Journal of Clinical Oncology*, 22(4):714–24.

[Velikova et al., 1999] Velikova, G., Wright, E., Smith, A., Cull, A., Gould, A., Forman, D., Perren, T., Stead, M., Brown, J., and Selby, P. (1999). Automated collection of quality-of-life data: a comparison of paper and computer touch-screen questionnaires. *Journal of clinical oncology : official journal of the American Society of Clinical Oncology*, 17(3):998–1007.

[W3C, 2011] W3C (2011). Web storage. `http://dev.w3.org/html5/webstorage/`. Accessed October 29, 2011.

[W3Schools, 2011] W3Schools (2011). Browser statistics. Accessed October 9, 2011.

[Wiklund, 2004] Wiklund, I. (2004). Assessment of patient-reported outcomes in clinical trials: the example of health-related quality of life. *Fundamental & Clinical Pharmacology*, 18(3):351–363.

[Wilson and Cleary, 1995] Wilson, I. B. and Cleary, P. D. (1995). Linking clinical variables with health-related quality of life. *JAMA: The Journal of the American Medical Association*, 273(1):59–65.

[Yahoo.com, 2011] Yahoo.com (2011). Yahoo! `http://www.yahoo.com/`. Accessed October 23, 2011.

[Zahariev, 2009] Zahariev, A. (2009). Google app engine. *Seminar on Internetworking*.

Appendice A

NECK PAIN DISABILITY INDEX QUESTIONNAIRE

PLEASE READ*: This questionnaire is designed to enable us to understand how much your neck pain has affected your ability to manage your everyday activities. Please answer each section by circling the ONE CHOICE that most applies to you. We realize that you may feel that more than one statement may relate to you, but *PLEASE JUST CIRCLE THE ONE. CHOICE WHICH MOST CLOSELY DESCRIBES YOUR PROBLEM RIGHT NOW.

SECTION 1 - Pain Intensity **A I have no pain at the moment.** **B The pain is very mild at the moment.** **C The pain is moderate at the moment.** **D The pain is fairly severe at the moment.** **E The pain is very severe at the moment.** **F The pain is the worst imaginable at the moment.**	***SECTION 6 - Concentration*** **A I can concentrate fully when I want to with no difficulty.** **B I can concentrate fully when I want to with slight difficulty.** **C I have a fair degree of difficulty in concentrating when I want to.** **D I have a lot of difficulty in concentrating when I want to.** **E I have a great deal of difficulty in concentrating when I want to.** **F I cannot concentrate at all.**
SECTION 2 -Personal Care (Washing, Dressing, etc.) **A I can look after myself normally without causing extra pain.** **B I can look after myself normally, but it causes extra pain.** **C It is painful to look after myself and I am slow and careful.** **D I need some help, but manage most of my personal care.** **E I need help every day in most aspects of self care.** **F I do not get dressed, I wash with difficulty and stay in bed.**	***SECTION 7 - Work*** **A I can do as much work as I want to.** **B I can only do my usual work, but no more.** **C I can do most of my usual work, but no more.** **D I cannot do my usual work.** **E I can hardly do any work at all.** **F I cannot do any work at all.**
SECTION 3 - Lifting **A I can lift heavy weights without extra pain.** **B I can lift heavy weights, but it gives extra pain.** **C Pain prevents me from lifting heavy weights off the floor, but I can manage if they are conveniently positioned, for example, on a table.** **D Pain prevents me from lifting heavy weights, but I can manage light to medium weights if they are conveniently positioned.** **E I can lift very light weights.** **F I cannot lift or carry anything at all.**	***SECTION 8 - Driving*** **A I can drive my car without any neck pain.** **B I can drive my car as long as I want with slight pain in my neck.** **C I can drive my car as long as I want with moderate pain in my neck.** **D I cannot drive my car as long as I want because of moderate pain in my neck.** **E I can hardly drive at all because of severe pain in my neck.** **F I cannot drive my car at all.**
SECTION 4 - Reading **A I can read as much as I want to with no pain in my neck.** **B I can read as much as I want to with slight pain in my neck.** **C I can read as much as I want to with moderate pain in my neck.** **D I cannot read as much as I want because of moderate pain in my neck.** **E I cannot read as much as I want because of severe pain in my neck.** **F I cannot read at all.**	***SECTION 9 - Sleeping*** **A I have no trouble sleeping.** **B My sleep is slightly disturbed (less than 1 hour sleepless).** **C My sleep is mildly disturbed (1-2 hours sleepless).** **D My sleep is moderately disturbed (2-3 hours sleepless).** **E My sleep is greatly disturbed (3-5 hours sleepless).** **F My sleep is completely disturbed (5-7 hours)**
SECTION 5 - Headaches **A I have no headaches at all.** **B I have slight headaches which come infrequently.** **C I have moderate headaches which come infrequently.** **D I have moderate headaches which come frequently.** **E I have severe headaches which come frequently.** **F I have headaches almost all the time.**	***SECTION 10 - Recreation*** **A I am able to engage in all of my recreational activities with no neck pain at all.** **B I am able to engage in all of my recreational activities with some pain in my neck.** **C I am able to engage in most, but not all of my recreational activities because of pain in my neck.** **D I am able to engage in a few of my recreational activities because of pain in my neck.** **E I can hardly do any recreational activities because of pain in my neck.** **F I cannot do any recreational activities at all.**

COMMENTS:_____

_____

_____

NAME: _____ **DATE:** _____ **SCORE:** _____

SCORING TECHNIQUE FOR THE OSWESTRY LOW BACK DISABILITY QUESTIONNAIRE AND NECK DISABILITY INDEX

1. Each of the 10 sections is scored separately (0 to 5 points each) and then added up (max. total = 50).

***EXAMPLE*:**

Section 1. Pain Intensity	**Point Value**
A. _____ I have no pain at the moment	**0**
B. _____ The pain is very mild at the moment	**1**
C. _____ The pain is moderate at the moment	**2**
D. _____ The pain is fairly severe at the moment	**3**
E. _____ The pain is very severe at the moment	**4**
F. _____ The pain is the worst imaginable	**5**

2. If all 10 sections are completed, simply double the patients score.

3. If a section is omitted, divide the patient's total score by the number of sections completed times 5.

***FORMULA*:** $\frac{\text{PATIENT'S SCORE}}{\text{\# OF SECTIONS COMPLETED X 5}}$ **X 100 = _____ % DISABILITY**

***EXAMPLE*:**

If 9 of 10 sections are completed, divide the patient's score by 9 X 5 = 45; if……..

Patient's Score: 22

Number of sections completed: 9 (9 X 5 = 45)

22/45 X 100 = 48 % disability

4. Interpretation of disability scores (from original article):

SCORE	INTERPRETATION OF THE OSWESTRY LBP DISABILITY QUESTIONNAIRE
0-20% Minimal Disability	**Can cope w/ most ADL's. Usually no treatment needed, apart from advice on lifting, sitting, posture, physical fitness and diet. In this group, some patients have particular difficulty with sitting and this may be important if their occupation is sedentary (typist, driver, etc.)**
20-40% Moderate Disability	**This group experiences more pain and problems with sitting, lifting and standing. Travel and social life are more difficult and they may well be off work. Personal care, sexual activity and sleeping are not grossly affected, and the back condition can usually be managed by conservative means.**
40-60% Severe Disability	**Pain remains the main problem in this group of patients by travel, personal care, social life, sexual activity and sleep are also affected. These patients require detailed investigation.**
60-80% Crippled	**Back pain impinges on all aspects of these patients' lives both at home and at work. *Positive intervention is required.***
80-100%	**These patients are either bed-bound or exaggerating their symptoms. This can be evaluated by careful observation of the patient during the medical examination.**

Reference: Fairbanks CT, Couper C, Davies JB, O'Brien JP. The Oswestry low back pain disability questionnaire. Physio Ther 1980;66:271-273.

Appendice B

Esempio di XML per la modellazione di un nuovo questionario

```
<?xml version="1.0" encoding="utf-8"?>
<questionnaire>

  <author did="4" />
  <title xml:lang="it">Questionario sul dolore</title>
  <info xml:lang="it">Questo è un questionario di esempio</info>
  <title xml:lang="en">Pain questionnaire</title>
  <info xml:lang="en">This is a test questionnaire</info>
  <description>A ePRO about pain</description>
  <measures>
    <measure>
      <name xml:lang="it">Livello di dolore</name>
      <name xml:lang="en">Pain Level</name>
      <method>f()</method>
    </measure>
  </measures>

  <question type="o" left_val="1" right_val="1">
    <string xml:lang="it">Che livello di dolore prova?</string>
    <option xml:lang="it">Nessun dolore</option>
    <option xml:lang="it">Dolore moderato</option>
    <option xml:lang="it">Dolore molto forte</option>
    <string xml:lang="en">How much pain do you feel?</string>
    <option xml:lang="en">No pain, no gain</option>
    <option xml:lang="en">Moderate pain</option>
    <option xml:lang="en">Strong pain</option>
  </question>
```

```
    <question type="o" left_val="0" right_val="NaN">
      <string xml:lang="it">Quando ha provato dolore l&#39;ultima volta?</string>
      <option xml:lang="it">L&#39;altro ieri</option>
      <option xml:lang="it">Ieri</option>
      <option xml:lang="it">Oggi</option>
      <string xml:lang="en">When did you feel pain last time?</string>
      <option xml:lang="en">Two days ago</option>
      <option xml:lang="en">Yesterday</option>
      <option xml:lang="en">Today</option>
    </question>

    <question type="b" left_val="1" right_val="2">
      <string xml:lang="it">Ha misurato la glicemia oggi?</string>
      <option xml:lang="it">Sì</option>
      <option xml:lang="it">No</option>
      <string xml:lang="en">Did you measure your glycaemia today?</string>
      <option xml:lang="en">Yes</option>
      <option xml:lang="en">No</option>
    </question>

    <question type="n" left_val="" right_val="">
      <string xml:lang="it">Valore di glicemia registrato</string>
      <string xml:lang="en">Glycaemia value</string>
    </question>

    <question type="r" left_val="100" right_val="200">
      <string xml:lang="it">Inserisca il valore della pressione misurata</string>
      <string xml:lang="en">Please, insert the blood pressure measurement</string>
    </question>

    <question type="s" left_val="0" right_val="200">
      <string xml:lang="it">Scriva la parola &quot;casa&quot;</string>
      <string xml:lang="en">Please, insert the word "casa"</string>
    </question>

  </questionnaire>
```

Esempio di XML per la modellazione di un questionario da compilare.

```
<?xml version="1.0" encoding="utf-8"?>
<questionnaire quid="1">

  <title xml:lang="it">Questionario sul dolore</title>
  <info xml:lang="it">Questo è un questionario di esempio</info>

  <question qid="1" type="o" left_val="1" right_val="1">
    <string xml:lang="it">Che livello di dolore prova?</string>
    <option xml:lang="it" oid="1">Nessun dolore</option>
    <option xml:lang="it" oid="2">Dolore moderato</option>
    <option xml:lang="it" oid="3">Dolore molto forte</option>
  </question>

  <question qid="2" type="o" left_val="0" right_val="NaN">
    <string xml:lang="it">Quando ha provato dolore l&#39;ultima volta?</string>
    <option xml:lang="it" oid="7">L&#39;altro ieri</option>
    <option xml:lang="it" oid="8">Ieri</option>
    <option xml:lang="it" oid="9">Oggi</option>
  </question>

  <question qid="3" type="b" left_val="1" right_val="2">
    <string xml:lang="it">Ha misurato la glicemia oggi?</string>
    <option xml:lang="it" oid="13">Sì</option>
    <option xml:lang="it" oid="14">No</option>
  </question>

  <question qid="4" type="n" left_val="" right_val="">
    <string xml:lang="it">Valore di glicemia registrato</string>
  </question>

  <question qid="5" type="r" left_val="100" right_val="200">
    <string xml:lang="it">Inserisca il valore della pressione misurata</string>
  </question>

  <question qid="6" type="s" left_val="0" right_val="200">
    <string xml:lang="it">Scriva la parola &quot;casa&quot;</string>
  </question>

</questionnaire>
```

Esempio di XML per la modellazione dei risultati di un questionario.

```
<?xml version="1.0" encoding="utf-8"?>
<results gid="1">

  <question qid="1">
    <chosen-option oid="2"/>
  </question>

  <question qid="2">
    <chosen-option oid="7"/>
    <chosen-option oid="8"/>
  </question>

  <question qid="3">
    <chosen-option oid="13"/>
  </question>

  <question qid="4">
    <input type="n">200</input>
  </question>

  <question qid="5">
    <input type="r">105</input>
  </question>

  <question qid="6">
    <input type="s">casa</input>
  </question>

</results>
```

Appendice C

Figura 2: Schermata di selezione del questionario.

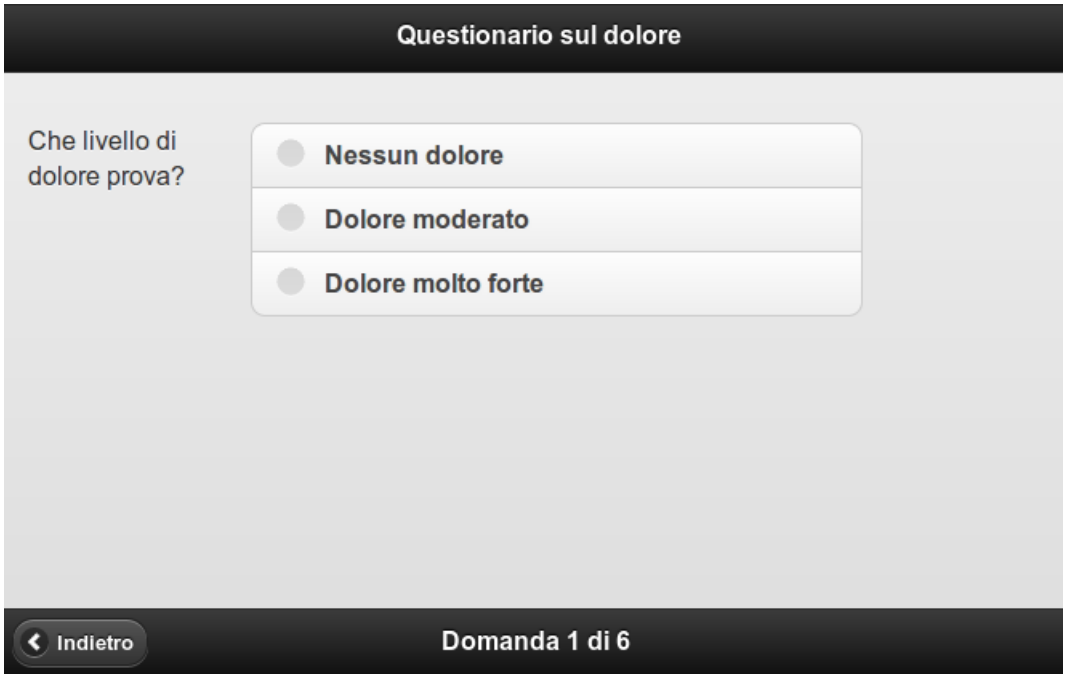

Figura 3: Schermata per domanda opzionale a risposta singola prima che l'utente abbia effettuato una scelta.

Figura 4: Schermata per domanda opzionale a risposta singola dopo che l'utente ha effettuato una scelta.

Figura 5: Schermata per domanda opzionale a risposta multipla.

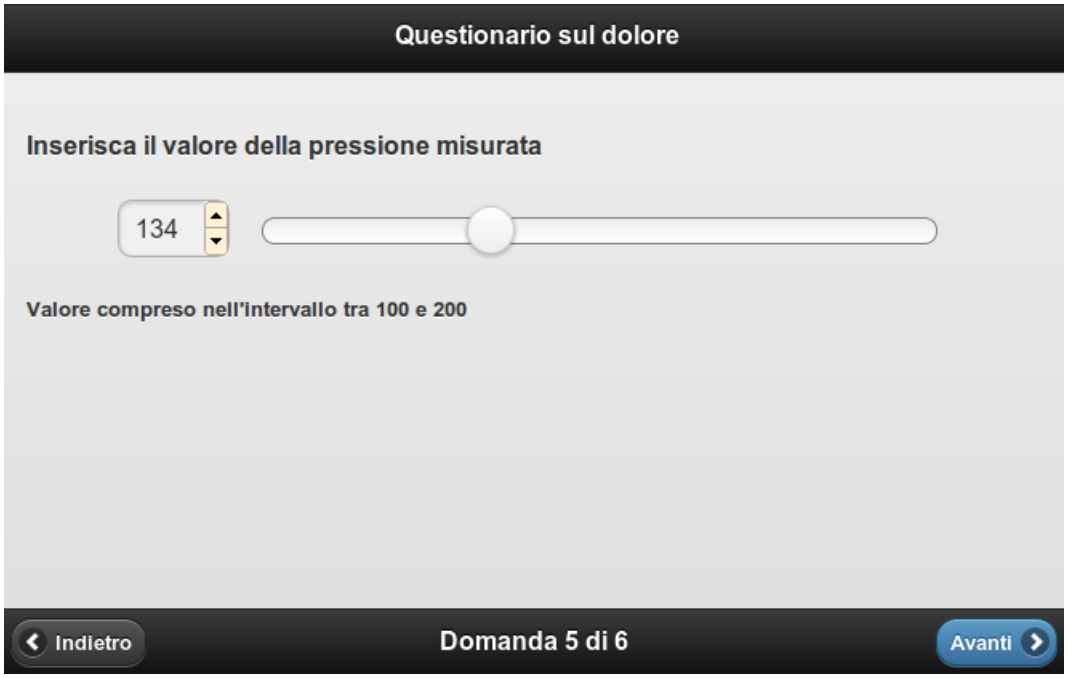

Figura 6: Schermata per domanda numerica con range.