UNIVERSITÀ DI PADOVA **SCUOLA DI INGEGNERIA**

TESI DI LAUREA

PANDI: A PANCREAS DIARY. PROGETTAZIONE E SVILUPPO DI UN' APP CROSS-PLATFORM PRE E POST OPERATORIA PER SOGGETTI AFFETTI DA TUMORE AL PANCREAS.

Laureando: Matteo Breschigliaro

Relatore: Prof. Sergio Canazza Targon
Correlatore: Dott. Michele Patella

Corso di Laurea Magistrale in Ingegneria Informatica

Data Laurea: 04/04/2017

Anno Accademico 2016/2017

Sommario

La tesi descrive la progettazione e l'implementazione di un' app cross-platform per aiutare pazienti malati di tumore al pancreas. L'app aiuta queste persone a prendere nota di farmaci assunti, sintomi percepiti e pasti consumati durante il decorso della malattia, e genera automaticamente un report riassuntivo per le visite di controllo.

Nel primo capitolo vengono spiegate le problematiche che hanno portato all'idea di quest'app e i requisiti tecnici e funzionali richiesti dal committente. Nel secondo capitolo vengono descritte le tecnologie utilizzate per lo sviluppo. Nel terzo viene spiegata l'architettura di ogni componente che compone l'app, mentre nel quarto viene spiegato in maggior dettaglio l'utilizzo pratico delle varie funzioni dell'app. Il capitolo quinto spiega la fase di beta test, debug e collaudo dell'app, e le presentazioni effettuate. Nel sesto capitolo vengono descritti gli sviluppi futuri del progetto.

Ringraziamenti

Al termine di questo mio percorso di studi desidero ringraziare le persone che mi sono state maggiormente vicine.

In primis, ringrazio i miei genitori e tutta la mia famiglia. Hanno sempre creduto in me e nelle mie capacità, molto più di me stesso. Mi hanno sempre sostenuto e spinto a raggiungere obiettivi che desideravo ma credevo irrealizzabili. Senza il loro sostegno probabilmente non sarei mai riuscito ad arrivare fin qui.

Ringrazio inoltre tutti i miei amici più cari, con cui ho condiviso bellissime esperienze e che mi sono stati molto vicini nei momenti più difficili della mia vita.

Ringrazio il Prof. Sergio Canazza per avermi seguito e consigliato durante la stesura della tesi con prontezza e disponibilità, ed il Prof. Antonio Rodà per avermi dato la possibilità di effettuare lo stage in Audio Innova.

Ringrazio Michele Patella e tutte le persone conosciute durante questo stage: da loro ho imparato moltissimo e sono stati una fonte di innumerevoli e utilissimi consigli.

Infine ringrazio Adele, che con la sua costante presenza è la forza e il sostegno di ogni mia giornata.

Grazie.

Indice

Sommario i

Ringraziamenti iii

1 Introduzione 1
1.1 Requisiti e obiettivi . . . 2
1.1.1 Organizzazione del lavoro e tempi di sviluppo . . . 4

2 Tecnologie utilizzate 7
2.1 Framework di sviluppo ibrido: Adobe Phonegap . . . 8
2.1.1 Phonegap Plugins . . . 10
2.1.2 Cocoon Webview+ . . . 11
2.2 Motore grafico: HTML5+CSS . . . 11
2.2.1 Framework7 . . . 12
2.3 Motore di scripting: Javascript . . . 13
2.3.1 JSZip . . . 13
2.3.2 Javascript Template . . . 14
2.4 Motore di database: PouchDB . . . 14
2.5 Licenze e Open Source . . . 15

3 Architettura App 17
3.1 Componenti grafiche . . . 19
3.2 Architettura del database e stoccaggio dei dati . . . 26
3.3 Liste di dati . . . 28
3.4 Foto . . . 29
3.5 Stampa . . . 30
3.6 Backup e ripristino . . . 31
3.7 Localizzazione . . . 31
3.8 Bug rilevati sui framework utilizzati . . . 33

4 Funzionalità App **35**
4.1 Diario 35
4.2 Agenda 38
4.3 Informazioni pancreas 40
4.4 Stampa report 40
4.5 Backup, ripristino e altre funzioni 41

5 Collaudo e presentazioni dell'app **43**
5.1 Beta testing e collaudo iniziale 43
5.2 Presentazioni 45

6 Sviluppi futuri e conclusioni **47**
6.1 Database remoto e sync 47
6.1.1 Implicazioni legali riguardanti la natura dei dati raccolti 48
6.2 Frontend per i medici 49
6.3 Studi statistici sui dati raccolti 49
6.4 Conclusioni 50

Appendici **52**

A Codice **53**
A.1 Funzione buildDiaryList() 53
A.2 Funzione buildAgendaList() 55
A.3 Funzione takePhotoFromCamera 56
A.4 Funzione localBackup 57
A.5 Funzione localRestore 58
A.6 Funzioni di localizzazione 59
A.7 Funzione di test 61

Elenco delle figure

1.1 Ciclo di raccolta dei dati attuale 2
1.2 Ciclo di raccolta dei dati utilizzando PanDi 3
1.3 Diagramma di Gantt del progetto e sviluppo di PanDi 5

2.1 Confronto tra diverse modalità di sviluppo di un' applicazione mobile. . 7
2.2 Logo di Adobe Phonegap. 9
2.3 Logo di Apache Cordova 9
2.4 Schema architetturale di Phonegap. 10
2.5 Logo di HTML5 e CSS3 11
2.6 Logo di Framework7 12
2.7 Logo di PouchDB 14

3.1 Ciclo funzionale di PanDi. 18
3.2 Navbar 20
3.3 Sidebar di sinistra 21
3.4 Sidebar di destra 21
3.5 Tab bar 22
3.6 Card di aiuto 22
3.7 Popover 22
3.8 Tab diario 23
3.9 Tab agenda 23
3.10 Tab info 23
3.11 Input form 23
3.12 Navbar evento 23
3.13 Modal 23
3.14 Input farmaco 24
3.15 Input sintomo 24
3.16 Input pasto 24
3.17 Autocomplete 24
3.18 Notifica aiuto 24
3.19 Card foto 25
3.20 Notifica per il riavvio 25

3.21 Bug di disallineamento del campo ora. 33

4.1 Prima slide introduttiva 36
4.2 Diario senza elementi nella lista 36
4.3 Menù per inserire un nuovo evento 36
4.4 Schermata di inserimento di un farmaco 37
4.5 Dropdown di inserimento di un farmaco 37
4.6 Schermata di inserimento di un sintomo 38
4.7 Dropdown di inserimento di un sintomo 38
4.8 Schermata di inserimento di un pasto 39
4.9 Alimenti che compongono un pasto 39
4.10 Diario con elementi nella lista 39
4.11 Sidebar per filtrare gli elementi del diario 39
4.12 Funzione agenda 40
4.13 Funzione informazioni pancreas 41
4.14 Schermata di selezione del periodo e dei tipi da includere nel report . . 42
4.15 Report generato attraverso Google Cloud Print 42
4.16 Schermata backup 42
4.17 Schermata impostazioni 42

5.1 Stand di Audio Innova alla fiera SMAU 2017 Padova 45

6.1 Seconda fase del progetto PanDi 48

Capitolo 1

Introduzione

I pazienti con una diagnosi di tumore del pancreas, sia esso esocrino che endocrino, affrontano un percorso di cura che prevede di incontrare con cadenza prestabilita i chirurghi, gli oncologi o i radioterapisti per delle visite in cui devono riportare tutte le informazioni rilevanti sul proprio stato di salute durante il periodo che intercorre tra una visita e quella successiva, le quali influenzeranno poi il seguito della cura. I pazienti, inoltre, sono tenuti, in fase pre e post operatoria, nonchè nel periodo di degenza successivo a questa fase, ad un'assunzione regolare di farmaci e a uno stretto controllo alimentare, che comporta la necessità di seguire una dieta prestabilita. [1]

Sono stati svolti diversi studi riguardo il fatto che il paziente tende a fornire un rendiconto molto falsato della sintomatologia durante queste visite di controllo. [2] Le informazioni fornite al medico risultano infatti incomplete e parziali e si focalizzano in particolar modo sugli eventi accaduti nei giorni precedenti alla visita, i quali influenzano negativamente la descrizione e l'importanza associata dal paziente agli eventi precedenti a questi. [3] Un ulteriore problema deriva dal fatto che molto spesso le visite mediche vengono vissute in una situazione emotivamente molto tesa e carica d'ansia da parte dei pazienti, compromettendo la razionalità del racconto. Inoltre, la descrizione dei sintomi è un'attività che si presta a molte ambiguità sia nella presentazione da parte del paziente sia di interpretazione da parte del medico. Tutti questi fattori portano lo specialista ad avere un'opinione sbagliata del reale stato di salute del paziente e ciò comporta la prescrizione di un tipo di cura solo parzialmente corretta.

Esiste quindi l'esigenza di una raccolta di questo tipo di dati da parte del paziente in modo preciso e puntuale, da effettuare non appena si verifica un determinato tipo di evento, per migliorare la qualità della visita di controllo e la cura del paziente che ne consegue.

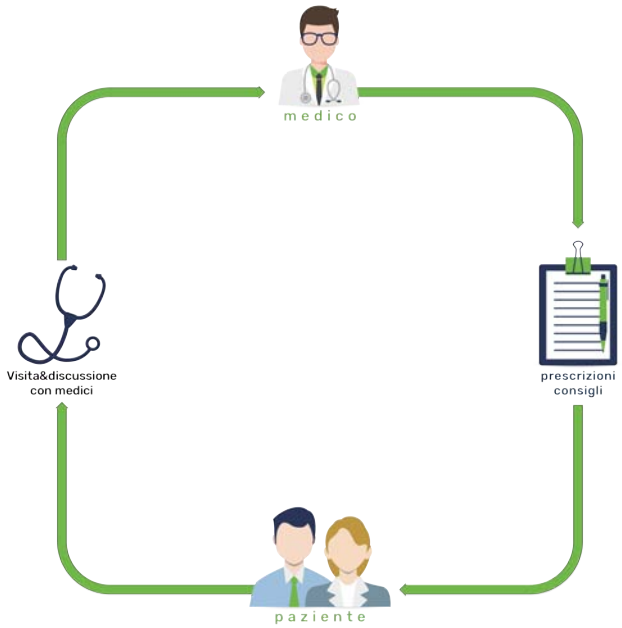

Figura 1.1: *Ciclo di raccolta dei dati attuale*

1.1 Requisiti e obiettivi

L'idea principale del progetto nasce dunque dall'esigenza di una raccolta dati precisa e immediata da parte dei pazienti stessi o di qualche familiare stretto. A seguito della raccolta dei dati, questi dovranno essere presentati ai medici durante le normali visite di controllo, e potranno essere oggetto di ulteriori analisi.

Il progetto è stato diviso in due fasi, la prima, che viene completata con questa tesi, ha come obiettivo principale la creazione di un'applicazione mobile per la raccolta dei dati; la seconda, come verrà illustrato in maggior dettaglio nel capitolo 6, prevede un' interfaccia medica per l'accesso ai dati raccolti attraverso un frontend dedicato e la possibilità di effettuare studi statistici sugli stessi.

I *requisiti tecnici* richiesti per l'applicazione sono i seguenti:

- Compatibilità con i sistemi operativi Android e iOS attraverso un unico codice sorgente (**cross-platforming**)
- Rilascio del codice sorgente secondo licenza **Open Source**

I *requisiti funzionali* richiesti per l'applicazione sono i seguenti:

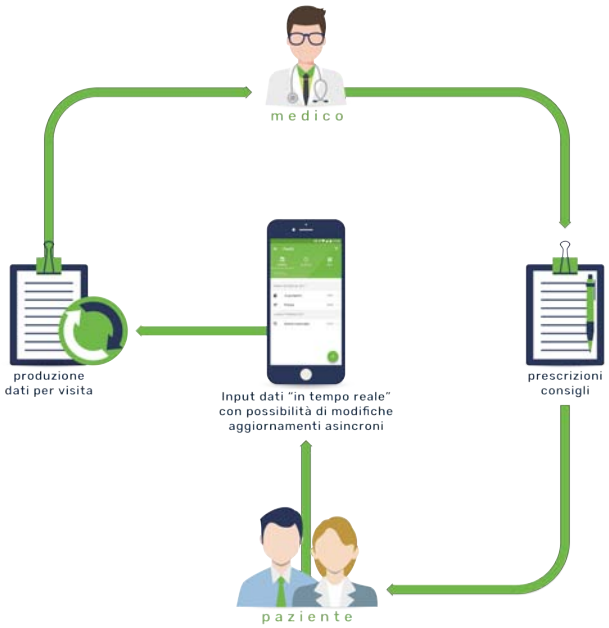

Figura 1.2: *Ciclo di raccolta dei dati utilizzando PanDi*

- Funzionalità per la raccolta dati sotto forma di **diario**
- Generazione di **report** a partire dai dati raccolti
- Possibilità di **filtraggio** dei dati raccolti
- Gestione degli appuntamenti medici attraverso un' **agenda**
- Funzionalità **informativa** con materiale medico divulgativo riguardante il pancreas e i relativi tumori
- Supporto a traduzioni in svariate lingue (italiano e inglese in questa prima versione)
- Predisposizione dell'architettura dell'applicazione per la seconda fase del progetto
- Garanzia della massima **facilità di utilizzo** e immediatezza per ogni genere di utente

I dati che è necessario raccogliere nel diario sono i seguenti:

- **Farmaci** assunti: nome del farmaco e quantità con relativa unità di misura (compresse, gocce, ecc.)
- **Sintomi** percepiti: tipologia del sintomo con eventuali foto
- **Pasti** consumati: tipologia del pasto (colazione, pranzo, spuntino, ecc.) con relativo elenco e quantità degli alimenti che lo compongono, eventualmente correlato da foto

Per ognuno di questi eventi deve essere salvata anche la data e l'ora ed eventuali note aggiuntive. È stato richiesto che l'inserimento di alcuni campi, come ad esempio il nome del farmaco o il tipo di sintomo, debba essere effettuato a partire da una lista di valori preimpostati: in questo modo si semplifica l'operazione di inserimento per l'utente e si rendono più omogenei i dati per un eventuale analisi futura.

Gli obiettivi della prima fase del progetto sono quindi la creazione dell'app e la sua distribuzione ai pazienti, in modo da permettere loro la familiarizzazione con questo nuovo approccio di raccolta dati.

1.1.1 Organizzazione del lavoro e tempi di sviluppo

Lo studio dell'architettura dell'applicazione e il suo relativo sviluppo sono stati effettuati attraverso uno stage di cinque mesi presso l'azienda Audio Innova Srl.

Il primo periodo di questo stage, della durata di circa tre settimane, è stato dedicato al progetto dell'architettura dell'app e all'identificazione dei framework e delle librerie necessarie alla realizzazione dell'app. Completata l'identificazione delle varie tipologie di componenti necessarie, è stato effettuato uno studio dello stato dell'arte di queste componenti, al fine di individuare quelle più adatte a questa applicazione.

In seguito, per un periodo di cinque settimane, è stato effettuato lo sviluppo di una prima parte dell'app contenente solamente la sezione diario, al termine del quale è stato effettuato un rilascio al committente di una prima versione per il solo sistema operativo Android.

Questa versione ha subito, nelle sei settimane successive, una massiccia fase di debug per il sistema operativo Android. Inoltre sono state inserite tutte le funzioni richieste ancora mancanti. In questo periodo sono anche state effettuate, su disposizioni del committente, numerose revisioni dal punto di vista grafico e dell'interazione con l'utente. Le versioni *beta* prodotte sono state rilasciate al committente svariate volte, per permettere allo stesso di visionarle e di presentarle al personale medico interessato.

Le ultime sei settimane dello stage sono state dedicate al termine del debug per Android e al debug completo per iOS, nonchè alla stesura della tesi.

Di seguito si può vedere il diagramma di Gantt di quanto descritto.

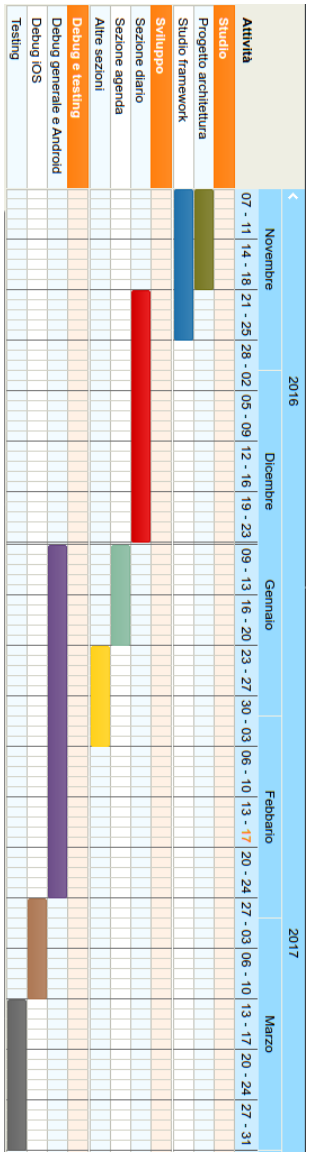

Figura 1.3: *Diagramma di Gantt del progetto e sviluppo di PanDi*

Capitolo 2

Tecnologie utilizzate

Esistono diverse modalità tecnologiche attraverso cui può essere sviluppata un' applicazione mobile, come illustrato nella figura 2.1.

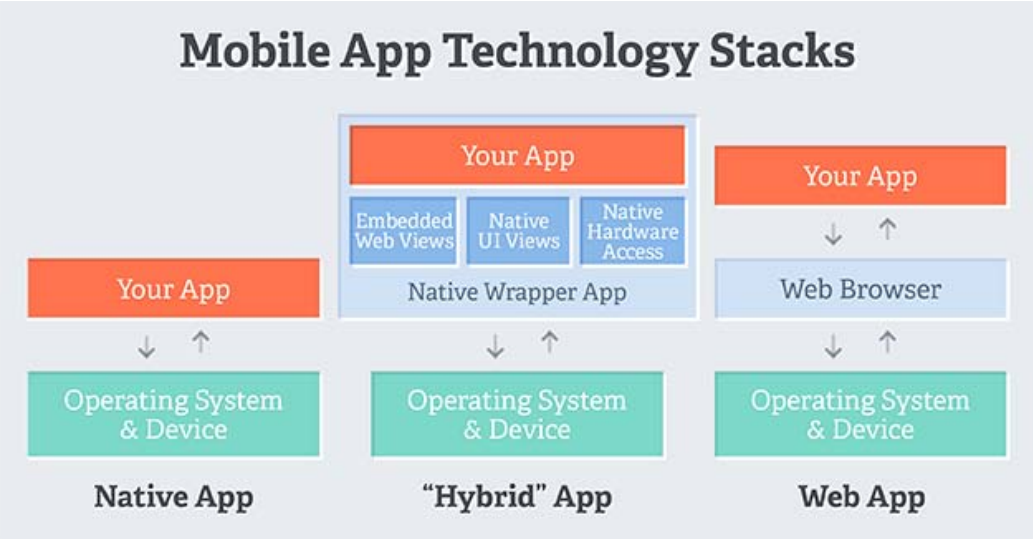

Figura 2.1: *Confronto tra diverse modalità di sviluppo di un' applicazione mobile.[4]*

Le varie tipologie di app si distinguono in:

- **App native**: queste app sono scritte nel linguaggio di programmazione nativo di ogni sistema operativo, e possono quindi essere compilate esclusivamente per il sistema operativo in questione. Hanno accesso a tutti i componenti hardware del device attraverso le API native, e sono quelle che riescono a ottenere le performance migliori.

- **App ibride**: queste app sono scritte in un linguaggio di programmazione che non dipende dal sistema operativo in questione, e vengono compilate attraverso compilatori che, a partire dallo stesso codice sorgente producono bytecode nativo. Ciò permette di avere accesso alle API native ma, aggiungendo uno strato intermedio tra il sistema operativo e l'applicazione, riduce le performance.
- **Web app**: questo tipo di applicazioni sono speciali siti web pensati appositamente per le piattaforme mobile. Queste applicazioni, essendo utilizzate direttamente dal browser, non hanno bisogno di essere compilate, ma non permettono l'accesso alle API del sistema operativo.

Durante la fase progettuale, si è ritenuto che l'applicazione avesse la necessità di accedere all'hardware di sistema (accesso ai file e alla fotocamera) e, dovendo soddisfare tale necessità unitamente al requisito di creare un unico codice sorgente, si è optato per lo sviluppo di un' app ibrida.

Sono perciò stati individuati i componenti principali da utilizzare, di seguito elencati:

- **Framework di sviluppo ibrido**: necessario per lo sviluppo e la compilazione cross-platforming dell'app
- **Motore grafico**: necessario per implementare le componenti grafiche dell'app
- **Motore di scripting**: necessario per implementare la logica dell'app
- **Motore di database**: necessario allo stoccaggio dei dati

Tutti i componenti sono stati arricchiti utilizzando alcune librerie che sono illustrate nei sottoparagrafi di ogni componente. In particolare, è stato utilizzato Framework7, che integra sia componenti grafiche che di scripting.

I componenti e le librerie necessarie sono stati scelti confrontando svariate alternative allo stato dell'arte, utilizzando come criterio di scelta in primo luogo l'adeguatezza al contesto applicativo. Sono stati tenuti in considerazione nella scelta anche la popolarità del componente, il supporto da parte della comunità e la licenza d'uso.

2.1 Framework di sviluppo ibrido: Adobe Phonegap

Adobe Phonegap è un framework di sviluppo ibrido open source ideato per realizzare app mobile attraverso tecnologie dedicate allo sviluppo di siti web, come HTML,

Figura 2.2: *Logo di Adobe Phonegap.*

CSS e Javascript.

Adobe Phonegap è la distribuzione più popolare di Apache Cordova, il motore che sta alla base di questo framework. Apache Cordova inizialmente era un progetto open source creato dall'azienda Nitobi, chiamato Phonegap. Nitobi in seguito è stata acquisita da Adobe, la quale ha deciso di mantenere in vita il progetto Phonegap donandolo ad Apache Software Foundation. L'operazione ha però richiesto la ridenominazione del progetto, ed è quindi stato scelto per esso il nome Apache Cordova. L'attuale versione di Adobe Phonegap è stata creata in seguito come fork di Apache Cordova. Apache Cordova e Adobe Phonegap sono entrambi coperti da licenza Apache. [5]

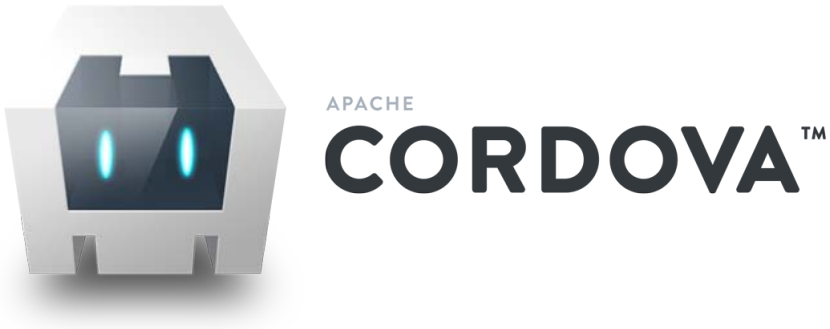

Figura 2.3: *Logo di Apache Cordova*

Come mostrato in figura 2.4, l'architettura di un' app compilata con Phonegap è costituita principalmente da tre macro sezioni:

- **Web App**: contiene tutto il codice sorgente e tutte le librerie per quanto riguarda la parte visuale e logica dell'app. Essa è a tutti gli effetti un sito web e quindi contiene codice HTML, CSS e Javascript.

- **Web View**: questa sezione è una versione particolare di un normale browser. Il componente interpreta il codice della Web App e consente all'applicazione di essere eseguita correttamente nel dispositivo.
- **Phonegap Plugins**: contiene svariati plugins necessari all'app per utilizzare i componenti nativi del dispositivo, come i sensori hardware oppure il file system. Questi plug-ins si interfacciano da una parte con il codice della Web App (in Javascript) e dall'altra con le API del sistema operativo.

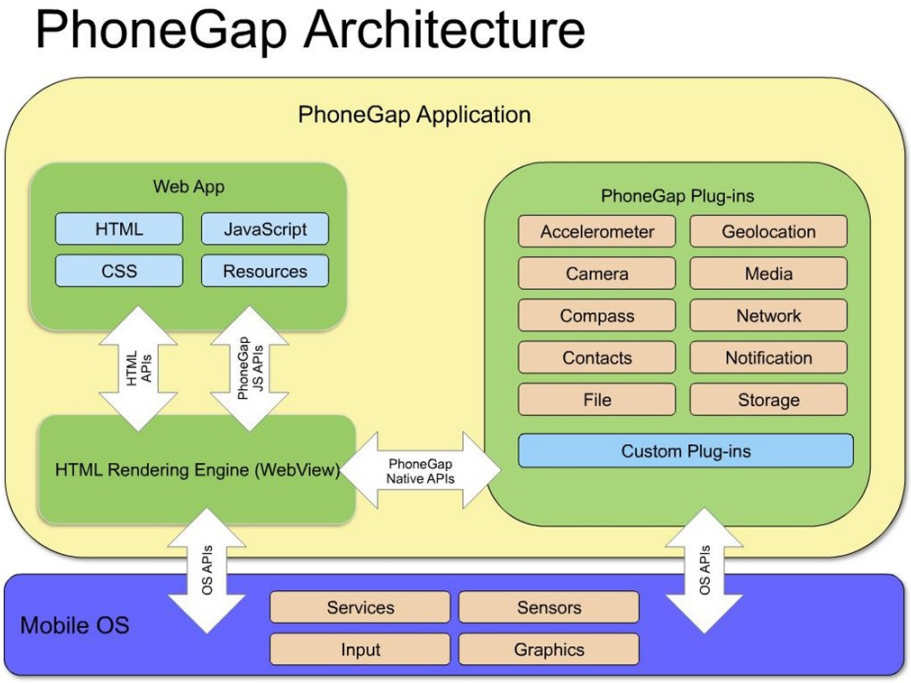

Figura 2.4: *Schema architetturale di Phonegap. [6]*

2.1.1 Phonegap Plugins

I plugins di Phonegap che sono stati utilizzati per PanDi sono i seguenti [7]:

- **cordova-plugin-camera**: Consente l'accesso alla fotocamera del dispositivo
- **cordova-plugin-compat**: Consente la gestione dei permessi dalla versione di Android 6.0 in poi
- **cordova-plugin-console**: Consente di stampare i log Javascript all'interno della console nativa

- **cordova-plugin-file**: Consente l'accesso al filesystem del dispositivo
- **cordova-plugin-keyboard**: Consente di controllare la tastiera virtuale del dispositivo
- **cordova-plugin-printer**: Consente un interfacciamento ai servizi di stampa nativi di ogni sistema operativo (Google Cloud Print, Apple AirPrint, etc.)
- **cordova-plugin-splashscreen**: Consente la visualizzazione di uno splashscreen iniziale
- **cordova-plugin-statusbar**: Consente la modifica di alcuni parametri della status bar del sistema operativo

2.1.2 Cocoon Webview+

Cocoon Webview+ è una webview alternativa a quella di default presente come componente nativo del sistema operativo, nata come fork del progetto open source Crosswalk.

Questa webview viene integrata direttamente nel pacchetto di installazione dell'app e consente di avere migliori performance, oltre a standardizzare il tipo di webview all'interno della quale viene eseguita l'applicazione, rendendo più semplice il debug. L'inclusione della webview nel codice dell'app avviene attraverso un tool di compilazione online, Cocoon Cloud.

2.2 Motore grafico: HTML5+CSS

Figura 2.5: *Logo di HTML5 e CSS3*

HTML (Hyper Text Markup Language) è il linguaggio di markup standard per la creazione di pagine web. HTML definisce la struttura delle pagine web utilizzando dei tag che identificano gli elementi della pagina.

Con la versione 5 di HTML sono state introdotte importanti novità che hanno permesso, tra le altre cose, un impiego migliore di questo linguaggio nel contesto delle mobile app. In particolare, con questa versione è stata introdotta la possibilità di memorizzare una grande quantità di dati nel browser (Webstorage).

CSS (Cascading Style Sheets) è un linguaggio usato per definire la formattazione di pagine HTML.

L'accoppiata di HTML5 e CSS, oltre a fornire un linguaggio di Turing completo, permette quindi di creare pagine con contenuti formattati e con la possibilità di salvare dati.

2.2.1 Framework7

Figura 2.6: *Logo di Framework7*

Framework7 è un framework HTML per la costruzione di applicazioni mobile ibride con un aspetto nativo.

Esso permette di creare l' interfaccia grafica dell' applicazione attraverso il linguaggio HTML e contiene al suo interno una serie di librerie e componenti visuali, come navbar, sidebar, input form, etc.

Questo framework integra il tema nativo di iOS e il tema di Google Material Design, permettendo quindi di creare app con entrambi i temi e di utilizzare quello più adatto a

ogni tipo di device.

Inoltre, a differenza di molti altri presenti sul campo, non utilizza dei tag HTML specifici del framework, ma si integra perfettamente con i tag HTML standard, permettendo un riutilizzo del codice anche con altri framework.

Framework7 è rilasciato sotto licenza MIT.

A corredo di questo framework sono state utilizzate anche due librerie CSS contententi dei set di icone vettoriali:

- **Material Design Iconic Font**: contentente tutte le icone Material Design create da Google
- **Font Awesome**: set di icone per siti web e mobile app

2.3 Motore di scripting: Javascript

Javascript è un linguaggio di scripting orientato agli oggetti e agli "eventi", utilizzato principalmente nella programmazione web.

Gli "eventi" sono azioni compiute dagli utenti, come ad esempio un click o un movimento del mouse, o una pressione di un tasto sulla tastiera, i quali possono innescare delle funzioni.

La principale particolarità di questo linguaggio è che esso viene eseguito lato client, permettendo quindi l'esecuzione locale degli script. Questo consente di eseguire script particolarmente complessi senza sovraccaricare il server e senza essere connessi alla rete.

Essendo un linguaggio di scripting, esso è un linguaggio interpretato: il codice sorgente non viene quindi compilato ma interpretato (l'interprete può essere il browser stesso).

2.3.1 JSZip

JSZip è una libreria Javascript per creare, leggere e modificare archivi zip. Essa non ha dipendenze ed è compatibile con la stragrande maggioranza dei browser.

Le API della libreria consentono la creazione asincrona del file zip: in questo modo si possono comprimere i file senza inficiare sulle performance dell'app.

La libreria è coperta da doppia licenza: MIT e GPLv3.

2.3.2 Javascript Template

Javascript Template è una libreria Javascript senza dipendenze per produrre documenti HTML a partire da un template e da dati strutturati JSON.

È compatibile sia a livello client che a livello server, ed è utile per la generazione di documenti HTML ripetitivi a partire da dati non formattati.

La libreria è coperta da licenza MIT.

2.4 Motore di database: PouchDB

Figura 2.7: *Logo di PouchDB*

PouchDB è un motore di database Javascript open source non relazionale ispirato ad Apache CouchDB.

PouchDB è un database di tipo NoSQL basato sui documenti, e fa riferimento a oggetti di tipo JSON per le operazioni di inserimento, modifica e cancellazione dei dati dal database.

Le particolarità che differenziano PouchDB rispetto ad altri motori di database sono il fatto che possa funzionare direttamente all'interno del browser e che sia molto leggero (circa 150kb).

Inoltre, PouchDB permette alle applicazioni che lo utilizzano di salvare i dati localmente, diminuendo quindi la differenza strutturale tra una Web App e un' app classica, permettendo però di sincronizzare i dati tra vari client oppure tra client e server quando

il dispositivo torna online.

PouchDB è coperto da licenza Apache.

2.5 Licenze e Open Source

Ricapitolando, tutte le librerie utilizzate sono open source e sono coperte dalle seguenti licenze:

- **Adobe Phonegap**: Licenza Apache 2.0
- **Cordova plugins**: Licenza Apache 2.0
- **Framework7**: Licenza MIT
- **Javascript Template**: Licenza MIT
- **PouchDB**: Licenza Apache 2.0

La licenza MIT è una licenza che consente agli utenti di usare il software per ogni scopo, di distribuirlo, modificarlo e di distribuire versioni modificate del software. È quindi una licenza permissiva e permette il riutilizzo del codice in software proprietario a condizione che la licenza MIT sia distribuita con tale software.

La licenza Apache 2.0 è molto simile alla licenza MIT, con l'unica differenza che la licenza richiede esplicitamente di preservare le informative sui diritti d'autore e brevetti contenuti in ogni file licenziato e di aggiungere un' informativa sui file modificati.

Capitolo 3

Architettura App

Come viene mostrato nella figura 3.1, l'applicazione PanDi è costituita da tre macrosezioni: diario, agenda e informazioni.

Il paziente, interagendo con il diario è in grado di inserire e modificare gli eventi (farmaci, sintomi e pasti), cercarli per nome e per data e filtrarli per tipo. Inoltre, può produrre i report desiderati, che possono essere stampati o salvati in PDF per consentire al medico di visionarli. Infine, può gestire i suoi appuntamenti medici attraverso l'agenda e consultare le informazioni sul pancreas fornite dall'app.

L'applicazione PanDi è costituita da circa 4000 righe di codice sorgente (librerie escluse). Il codice è costituito da file HTML e CSS, che compongono la parte visuale, e Javascript, che fornisce la logica dell'app, oltre ad alcuni file di configurazione XML.

Ogni file HTML di seguito elencato costituisce una diversa schermata dell'app:

- **index.html**: È il primo file che viene lanciato una volta aperta l'applicazione: esso ha il compito di caricare tutti i file CSS e JS necessari al corretto funzionamento. Inoltre contiene il codice della schermata principale dell'applicazione e il codice delle componenti comuni a tutte le schermate dell'app, come ad esempio le sidebar laterali e la navigation bar.
- **farmaco.html**: Contiene il codice relativo alla schermata di inserimento/modifica di un farmaco.
- **sintomo.html**: Contiene il codice relativo alla schermata di inserimento/modifica di un sintomo.
- **pasto.html**: Contiene il codice relativo alla schermata di inserimento/modifica di un pasto.

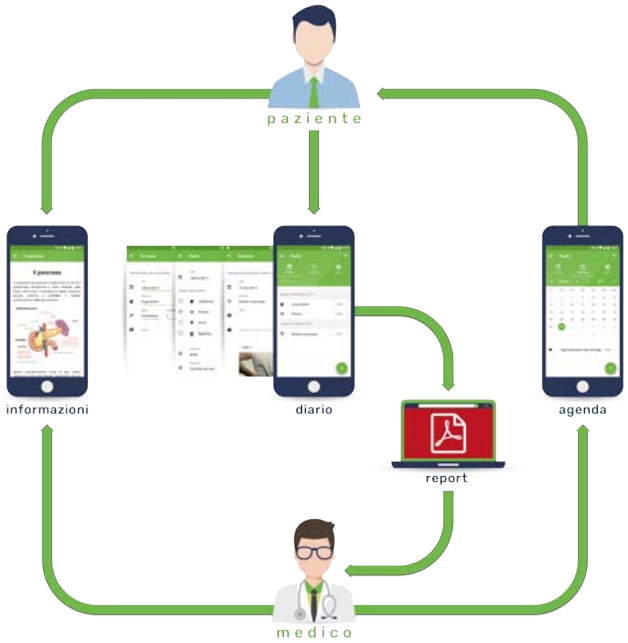

Figura 3.1: *Ciclo funzionale di PanDi.*

- **evento.html**: Contiene il codice relativo alla schermata di inserimento/modifica di un evento.
- **backup.html**: Contiene il codice relativo alla schermata per generare il backup del database e per effettuare il ripristino da un backup precedentemente generato.
- **settings.html**: Contiene il codice relativo alla schermata di gestione delle impostazioni. Consente di cambiare la lingua, la dimensione del testo e di disabilitare le notifiche di aiuto.
- **print.html**: Contiene il codice relativo alla schermata per generare il report del diario. Consente di scegliere il periodo su cui generare il report e di stampare solo alcuni tipi di eventi.

Inoltre, la sottocartella repo contiene i file HTML relativi alla sezione info, nelle varie lingue disponibili.

Il codice Javascript è invece suddiviso nei seguenti file:

- **agenda.js**: Contiene la logica della tab agenda e della pagina evento.html.
- **backup.js**: Contiene le funzioni che consentono di effettuare un dump/restore dei database e la logica della pagina backup.html.

- **core.js**: Contiene funzioni utili in tutti i contesti dell'app, ad esempio gli avvertimenti e le notifiche di aiuto.
- **diario.js**: Contiene la logica della tab diario.
- **farmaco.js**: Contiene la logica della pagina farmaco.html.
- **info.js**: Contiene la logica della pagina info.html.
- **init.js**: Contiene le funzioni di inizializzazione dell'app e la guida introduttiva.
- **language.js**: Contiene le funzioni che consentono di tradurre l'app nelle varie lingue presenti sul file string.js.
- **pasto.js**: Contiene la logica della pagina pasto.html.
- **printreport.js**: Contiene le funzioni per generare il report per la stampa.
- **settings.js**: Contiene la logica della pagina settings.html.
- **sintomo.js**: Contiene la logica della pagina sintomo.html.
- **string.js**: Contiene tutte le stringhe presenti nell'app nelle varie lingue disponibili.
- **utility.js**: Contiene funzioni utili come ad esempio la conversione e il padding della data.

Nei paragrafi successivi, verranno mostrate le principali componenti grafiche utilizzate e verrà spiegato il funzionamento delle funzioni principali dell'applicazione. In appendice si può trovare il codice relativo alle funzioni descritte.

3.1 Componenti grafiche

Tutte le componenti visuali dell'app si basano sulla libreria Framework7.[8] Sono stati utilizzati i seguenti componenti:

- **Pages**
- **Content Block**
- **Navbar**
- **Tab Bar**

- **Search Bar**
- **Side Panels**
- **Content Block**
- **Layout Grid**
- **Modal**
- **Popover**
- **List View**
- **Virtual List**
- **Cards**
- **Buttons**
- **Floating Action Buttons**
- **Form Element**
- **Checkboxes e Radios**
- **Smart Select**
- **Form Data**
- **Form Storage**
- **Tabs**
- **Photo Browser**
- **Autocomplete**
- **Calendar**
- **Notifications**

Figura 3.2: *Navbar*

Tutti i componenti grafici utilizzano il tema di Google Material Design, pertanto tutta la struttura e il layout dell'applicazione sono stati progettati seguendo le linee guida di questo standard. [9]

Ogni schermata di PanDi è composta almeno dalle seguenti componenti: la navbar (figura 3.2), che contiene il titolo della schermata e i bottoni laterali di azione (icone menù e filtri), due sidebar laterali e una pagina che contiene tutto il resto del contenuto. Le sidebar, sebbene strutturalmente sempre presenti in tutte le schermate, sono state disabilitate nei casi in cui il loro utilizzo risulta fuori luogo; quando abilitate, sono accessibili attraverso i bottoni laterali della navbar oppure attraverso uno swipe laterale. La sidebar sinistra (figura 3.3) contiene una list view a icone con il menù principale dell'app: da questo è possibile accedere alla maggior parte delle funzioni. La sidebar di destra (figura 3.4). contiene invece un checkbox form per filtrare gli elementi del diario e un calendar per visualizzare gli elementi alla data selezionata; poichè il contesto di questa sidebar è limitato al diario, essa è disabilitata in tutte le altre sezioni.

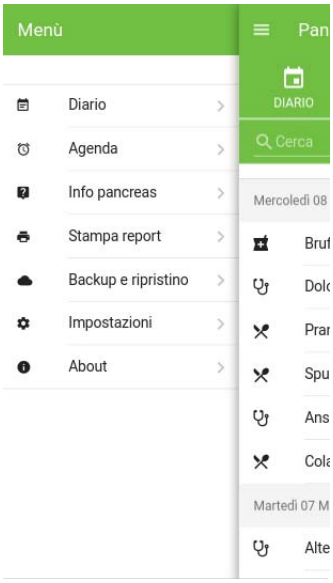

Figura 3.3: *Sidebar di sinistra*

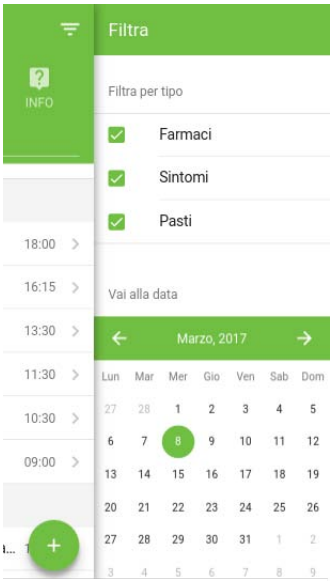

Figura 3.4: *Sidebar di destra*

La schermata principale di PanDi contiene nella page una tab bar (figura 3.5) per selezionare le varie tab e indicare la tab attualmente attiva. Sotto la tab bar è presente lo spazio dedicato alle sub-page di ogni tab.

La sub-page della tab diario (figura 3.8) contiene una search bar e uno spazio contenente una virtual list con gli eventi registrati nel diario, se presenti, oppure una card di

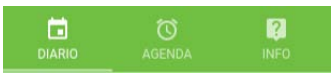

Figura 3.5: *Tab bar*

aiuto (figura 3.6) per inserire un elemento. Alla fine della pagina compare un floating action button che apre un popover (figura 3.7), il quale consente di accedere alle schermate di creazione di nuovi elementi del diario.

La sub-page della tab agenda (figura 3.9) contiene un calendar che consente di selezionare una data, per la quale verranno visualizzati, in una virtual list sotto di esso, gli appuntamenti memorizzati. Se non sono presenti appuntamenti, viene visualizzata una card di aiuto per inserire un elemento. Alla fine della pagina è presente un floating action button che apre la schermata di creazione di nuovo evento.

La sub-page della tab info (figura 3.10) contiene una list view che rimanda alle varie pagine informative.

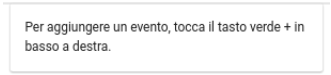

Figura 3.6: *Card di aiuto*

Figura 3.7: *Popover*

Nelle pagine farmaco.html, sintomo.html, pasto.html e evento.html, accessibili dal popover, dal floating action button oppure premendo su un elemento elencato nelle liste, sono presenti dei form element all'interno di una list view che consentono all'utente di inserire i dati (figura 3.11). In particolare, in tutte e quattro le pagine sono presenti i form di inserimento per la data e l'ora, i quali si compilano automaticamente in base ai valori correnti, e un form ridimensionabile per l'inserimento delle note. Il bottone sinistro (icona con freccia a sinistra) della navbar (figura 3.12) permette di aprire un modal che chiede all'utente se vuole tornare indietro senza salvare, mentre il bottone sulla destra (icona con spunta) consente di salvare l'elemento e tornare indietro. In queste schermate, si ottiene lo stesso risultato del bottone sinistro della navbar con il pulsante fisico back di Android, se presente nel dispositivo, oppure con il corrispondente soft-key.

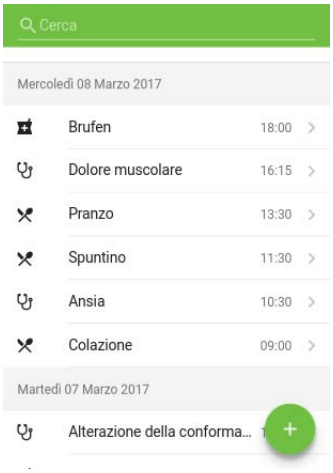

Figura 3.8: *Tab diario*

Figura 3.9: *Tab agenda*

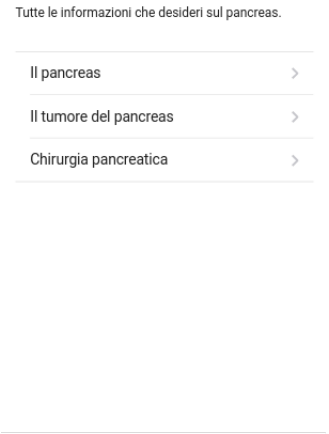

Figura 3.10: *Tab info*

Nel caso invece si sia aperta la pagina da un elemento della lista, essa si presenterà in modalità modifica, e i campi sono già compilati con i dati raccolti dal database. È presente inoltre un ulteriore bottone sulla destra della navbar (icona a cestino) che permette di aprire un modal che chiede all'utente se è veramente sicuro di voler eliminare quell'elemento. Allo stesso modo, il bottone di salvataggio apre un modal che chiede all'utente se è veramente sicuro di voler modificare l'elemento (figura 3.13).

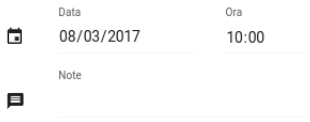

Figura 3.11: *Input form*

Figura 3.12: *Navbar evento*

Figura 3.13: *Modal*

Nella pagina farmaco.html sono inoltre presenti un form per l'inserimento del nome del farmaco, dell'unità di misura e della quantità assunta (figura 3.14). Nella pagina sintomo.html compare invece un form per l'inserimento del tipo di sintomo e un elemento della lista che permette di aprire la fotocamera di sistema per aggiungere foto (figura 3.15). La pagina pasto.html contiene un radio switch che permette di selezionare il tipo di pasto, un elemento della lista che consente di aprire la fotocamera di sistema per aggiungere foto, e un bottone che permette di aggiungere elementi a una lista che contiene gli alimenti e i pesi del pasto (figura 3.16). Nella pagina evento.html è infine presente un form per l'inserimento del nome dell'appuntamento.

Figura 3.14: *Input farmaco*

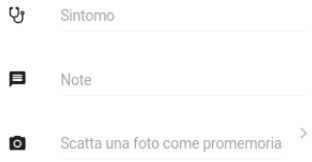

Figura 3.15: *Input sintomo*

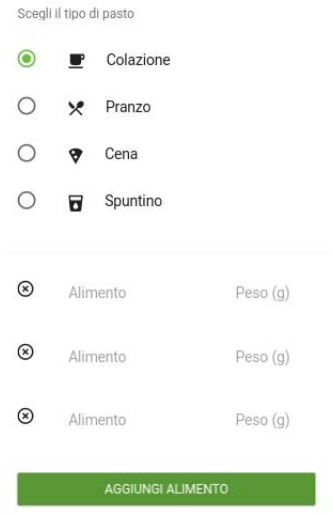

Figura 3.16: *Input pasto*

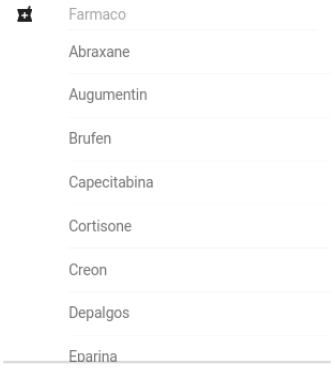

Figura 3.17: *Autocomplete*

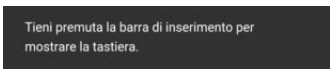

Figura 3.18: *Notifica aiuto*

I form per l'inserimento del nome del farmaco e della sua unità di misura, del tipo di sintomo e degli alimenti, una volta ricevuto il focus, aprono un dropdown-autocomplete (figura 3.17) che permette di selezionare l'elemento da un elenco preimpostato. Per questo motivo, i vari input a cui è associato un dropdown-autocomplete hanno l'attributo *onfocus="blur();"*, che consente all'elemento di non ricevere input dalla tastiera, e quindi impedisce la comparsa della soft-keyboard del sistema operativo. In alcuni elementi, dove si desidera invece che la tastiera compaia, per effettuare ricerche all'interno del dropdown-autocomplete, oppure per aggiungere elementi non presenti, è stata aggiunta una classe *show-keyboard*, la quale reagisce all'evento *taphold* e rimuove l'attributo *onfocus="blur();"*. Questo aspetto viene anche suggerito all'utente dalla comparsa di alcune notification di aiuto, disattivabili dalle impostazioni dell'app (figura 3.18). Se l'utente digita un elemento non presente nel dropdown-autocomplete e salva l'evento, l'elemento viene aggiunto automaticamente all'elenco. Questi elementi sono distinguibili da quelli preimpostati dall'app, perchè contengono un bottone con icona a forma di croce sulla sinistra: esso consente di cancellare l'elemento inserito dalla lista.

Quando l'utente preme sull'elemento che permette di aprire la fotocamera, l'app invia un intent di sistema e si apre l'applicazione della fotocamera desiderata (vedi paragrafo 3.4). Una volta scattata e salvata la foto, viene generata una card all'interno della rispettiva pagina contenente la foto croppata. (figura 3.19) Quando si clicca sulla foto, si apre un photo browser che permette di visualizzare la foto intera a tutto schermo e di zoomarla.

Figura 3.19: *Card foto*

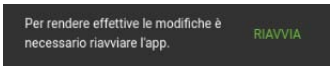

Figura 3.20: *Notifica per il riavvio*

Le pagine informative, accessibili dalla lista presente nella tab info, contengono content block di testo formattato con i formati standard di HTML e card con immagini che, se cliccate, aprono un photo browser, come nel precedente caso.

Le pagine printreport.html, backup.html e settings.html, accessibili dalla sidebar di sinistra, contengono form input di tipo switch, radio switch e smart select per selezionare le impostazioni relative a ogni funzione e bottoni per avviare le azioni. Nel caso fosse necessario riavviare l'app per rendere effettive le nuove impostazioni, compare

una notifica per avvertire l'utente che per applicare i cambiamenti l'app verrà riavviata (figura 3.20).

3.2 Architettura del database e stoccaggio dei dati

L'applicazione utilizza due database PouchDB, chiamati diaryDB e agendaDB, per salvare rispettivamente i dati di diario e agenda. Questi database, a loro volta, vengono salvati con tecnologia WebSQL nello storage della Webview. Si è optato per questa scelta, a discapito di un salvataggio in un database SQLite nativo, per non incorrere in problemi di performance legati a conversioni da una tipologia di database a un' altra, nonchè per il numero elevato di chiamate API native necessarie.

Il limite di dimensione imposto dal sistema operativo per i dati WebSQL nello storage di ciascuna app, nella peggiore delle ipotesi, è di 50 MB [10]: dal momento che un record del database occupa in media 500 byte (il database contiene solo dati testuali), la localstorage può contentere circa 100000 record. Si è stimato che questo valore sia più che sufficiente per quest'applicazione: considerato infatti che un paziente inserisce in media 10 record al giorno, lo spazio sarebbe sufficiente per inserire i dati di 27 anni.

Essendo PouchDB un database di tipo NoSQL, i dati sono organizzati in documenti non strutturati, i quali concettualmente corrispondono alle righe di una tabella di un database SQL. Ogni documento contiene svariati campi, i quali corrispondono alle colonne di una tabella in un database SQL. L'elemento identificativo di ogni documento è il campo `_id`, che corrisponde all'indice primario di un database SQL. I documenti sono salvati all'interno del database ordinati automaticamente per ID: si è scelto quindi di utilizzare come ID di ogni documento dell'applicazione la conversione sotto forma di timestamp Javascript di data e ora inserite dall'utente. In questo modo, si riesce a fare una query di tutti i documenti, o di una loro parte, ottenendo un elenco di documenti automaticamente ordinati per data e ora, senza impattare sulle performance per l'ordinamento degli stessi.

Ogni input field è salvato come un diverso campo all'interno del documento. Di seguito, l'esempio del documento di una entry del diario di tipo farmaco:

```
{
    "_id":"1488873674177"
    "_rev":"",
    "date":"2017-03-07",
    "time":"08:00",
```

```
    "name":"Augumentin",
    "unity":"Compressa",
    "qty":"2",
    "description":"",
    "type":"farmaco"
}
```

Poichè possono esserci svariati elementi con la stessa data e ora (il campo ora raggiunge una precisione dell'ordine dei minuti), la conversione di questi valori a un timestamp può comportare l'esistenza di elementi aventi stesso ID, cosa non accettabile, dal momento che l'ID è un valore identificativo univoco. Per evitare questo problema, l'ID viene generato aggiungendo al timestamp un numero randomico da 0 a 9999. Tale tecnica può essere implementata senza problemi dal momento che il timestamp in Javascript è un valore con la precisione dei millisecondi ma l'applicazione sfrutta solo la precisione dell'ordine dei minuti, quindi le ultime quattro cifre non vengono mai utilizzate e sono sempre pari a zero. Sfruttando questo fatto quindi, ogni ID viene creato in questo modo:

```
var id = dateId.getTime() + Math.floor((Math.random() *
↪ 10000));
```

dove al valore `dateId.getTime()`, che riporta il timestamp dei form, ci viene sommato un valore random da 0 a 9999. In questo modo, la probabilità che possa generarsi un conflitto è molto bassa.

PouchDB permette nativamente di salvare automaticamente anche tutte le revisioni di ogni documento: poichè questa funzione non è necessaria nell'app, per risparmiare spazio è stata disabilitata, attraverso il parametro `auto_compaction:true` in fase di creazione del database. Viene quindi salvata solo l'ultima revisione, e il motore di database si occupa in automatico di cancellare tutte le revisioni vecchie.

Per il database diaryDB viene sempre fatta una query di tipo `allDocs`: in pratica si leggono dal database tutti i documenti, i quali sono già ordinati in ordine descrescente di timestamp (dal più recente al meno recente) grazie all'ordinamento intrinseco fornito dall'ID. Questa query è molto veloce perchè non considera indici secondari nè utilizza algoritmi di ricerca. Nel caso in cui i documenti da leggere fossero molti e le performance di caricamento dei dati si riducessero, è possibile limitare il numero di documenti letti a un certo numero preimpostato, e riprendere la lettura da dove si era arrivati in un secondo momento.

Per il database agendaDB invece, viene effettuata una query utilizzando come indice secondario il campo `date`: per ogni query è infatti necessario leggere solo i documenti

corrispondenti a una singola data. Viene poi sfruttato l'ordinamento intrinseco degli ID per l'ordinamento orario dei documenti con la stessa data. Per effettuare questa query, viene utilizzato il plugin `pouchdb-find`, per effettuare la query con lo stile di MongoDB.

Gli altri dati dell'applicazione che che necessitano il salvataggio, come le variabili di configurazione o gli elementi aggiuntivi inseriti dall'utente negli elenchi dropdown-autocomplete, sono stati salvati invece nella localstorage sotto forma di coppie chiave-valore. È stato scelto di memorizzare questi dati sulla localstorage e non su un database perchè, in ottica futura, i database dovranno essere sincronizzati con un backend, mentre queste variabili sono impostazioni locali dell'app che possono variare in base al dispositivo utilizzato.

Vengono invece salvate nella memoria di massa del dispositivo le foto e i file di backup generati dall'app (vedi paragrafi 3.4 e 3.6).

3.3 Liste di dati

Le liste che contengono i dati delle sezioni Diario e Agenda sono create con il componente di Framework7 virtual list. Una virtual list, a differenza di una normale list view, permette di creare liste con con un numero elevato di elementi senza perdita di performance, in quanto questo tipo di lista viene renderizzata direttamente da Javascript. Oltre a ciò, permette di gestire in maniera molto più pratica i dati da codice Javascript, potendo trattarli come normali oggetti JSON e non come elementi HTML.

Le virtual list vengono inizializzate all'avvio dell'app e vengono popolate dalle rispettive funzioni `buildDiaryList()` e `buildAgendaList()` all'avvio dell'app e a ogni aggiunta o modifica di un elemento.

L'obiettivo della funzione `buildDiaryList()`, riportata in appendice A.1 è di leggere tutti i documenti dal database diaryDB (che sono già ordinati per timestamp, vedi paragrafo 3.2, e di aggiungerli alla relativa virtual list, creando un elemento divisorio contenente la data, nel caso in cui il documento corrente sia il primo incontrato con quella data (vedi figura 3.8). La funzione inizialmente reinizializza cinque array, dei quali però solo uno servirà per questa funzione: `itemsPerDate`. In questo array viene salvato il numero di documenti presenti nella lista per ogni data, utilizzando come indice dell'array il timestamp del documento per la sola parte relativa alla data. In questo modo, non è necessaria una ricerca su tutto l'array per determinare se è già presente un altro documento con quella data. Poichè il timestamp è un numero molto elevato, si potrebbe pensare a un'enorme occupazione di memoria per la sua memoriz-

zazione; in realtà non è così, in quanto Javascript permette di creare array sparsi, con occupazione di memoria pari a quella dei record realmente inseriti.[11] La funzione, dopo aver eseguito una query di tipo `allDocs`, se il risultato non è nullo, costruisce il primo elemento divisorio e lo appende alla virtual list: il primo elemento infatti dovrà essere sempre di questo tipo. Inoltre, aggiorna l'array `itemsPerDate` come spiegato in precedenza. Successivamente, per tutti gli elementi letti dalla query, controlla nell'array `itemsPerDate` se esiste già un elemento con quella data: se non esiste, crea un divisorio e appende alla virtual list divisorio ed elemento, mentre se esiste già appende solo l'elemento, mantenendo sempre aggiornato l'array `itemsPerDate`. La parte finale della funzione gestisce la parte relativa alla card di aiuto: se la virtual list risulta vuota e non sono già presenti card di aiuto, crea la card, mentre se la virtual list non è vuota ed è presente la card di aiuto, la rimuove.

Invece l'obiettivo della funzione `buildAgendaList()`, riportata in appendice A.2, è di cercare i documenti in agendaDB con una certa data, passata come parametro alla funzione sotto forma di timestamp. La funzione, attraverso l'uso di funzioni native di Javascript produce una data sotto forma di stringa nel formato "anno-mese-giorno" (le date sono infatti salvate in questo modo nel database). Successivamente, attraverso la funzione `find()` di agendaDB, viene fatta una query sulla data selezionata. Come spiegato nel paragrafo 3.2, in agendaDB il campo data è un indice secondario. La funzione, allo stesso modo di quanto avviene nella funzione `buildDiaryList()`, procede poi con l'appendere gli elementi trovati nella lista, oppure a creare la card informativa se non ci sono elementi.

3.4 Foto

Per scattare le foto dei sintomi e pasti si è utilizzato il plugin di Phonegap `cordova-plugin-camera` che permette, attraverso API native, l'accesso alla fotocamera di sistema. [12] Una volta scattata la foto, questa viene poi salvata nel dispositivo attraverso il plugin `cordova-plugin-file`, che consente l'accesso nativo al filesystem del dispositivo. [13]

E' stato scelto che le foto scattate possano essere accessibili liberamente dall'utente anche fuori dall'applicazione, quindi i file vengono salvati rispettivamente nella cartella della memoria esterna `<sdcard>/Android/data/com.audioinnova.pandi` in Android e `Documents/PanDi` in iOS, rappresentate attraverso le variabili `cordova.file.externalDataDirectory` e `cordova.file.documentsDirectory`. La cartella `Documents` di iOS permette anche il backup automatico dei file nel account iCloud personale.

Poichè ogni evento è associato a un ID univoco, le foto relative a ogni evento sono salvate in una sottocartella che ha come nome l'ID dell'evento. Il nome del file della foto è invece il timestamp in cui la foto è stata scattata. Quando viene aperta la pagina di un elemento salvato, viene controllato se all'interno della cartella dei dati dell'applicazione esiste una sottocartella con il nome uguale all'ID dell'elemento. Se questa esiste, vengono caricate come card nella pagina le foto presenti all'interno di essa. Nel caso in cui un utente cambiasse la data o l'ora a un evento già salvato, l'ID di quell'evento deve essere modificato, per permettere alla lista di essere ordinata correttamente. È necessario quindi rinominare anche la sottocartella relativa a quell'evento con il nuovo ID, per mantenere l'associazione tra le foto e l'evento.

La funzione che permette di scattare una foto e salvarla nella cartella corretta è riportata in appendice A.3. Essa inizialmente setta alcuni parametri di configurazione per il plugin della camera: in particolare, il parametro `quality` diminuisce la qualità della foto scattata, rendendola meno pesante e più adatta al contesto di utilizzo. Successivamente, la funzione chiama il metodo `getPicture()` dell'oggetto creato dal plugin camera, il quale passa il controllo alla fotocamera nativa del sistema per permettere all'utente di scattare la foto. Una volta scattata la foto con successo, viene chiamata la funzione `cameraSuccess()`, che riceve come parametro il percorso temporaneo dove è stata salvata la foto. Questa funzione ha il compito principale di spostare il file dalla cartella temporanea alla cartella dove vengono salvate le foto relativa all'evento in questione, creandola se necessario, e di appendere la foto a un array per permettere di essere visualizzata per la prima volta come card nella schermata del sintomo o pasto. Inoltre, la funzione controlla preventivamente se l'elemento di cui si sta salvando la foto abbia già un ID associato: nel caso non ce l'abbia infatti, è necessario crearlo.

3.5 Stampa

Per effettuare la generazione del report è stata utilizzata la libreria Javascript Templates, che consente di generare documenti HTML formattati secondo un template a partire da dati JSON. [14] I dati JSON vengono generati a partire dalla lettura del database diaryDB con una query `allDocs`, e vengono poi aggiunti a un array secondo i criteri di filtraggio selezionati dall'utente nella pagina `printreport.html`. Il template genera il documento HTML inserendo i dati in una tabella, ordinati per timestamp (ordinamento intrinseco) e generando delle righe separatori ogni volta che si incontra una nuova data, sullo stile di quanto succede nella lista del diario.

La stampa del file HTML generato viene gestito dal plugin cordova-plugin-print. [15] Il plugin permette di interfacciarsi, attraverso API native del sistema operativo, ai

servizi di stampa del sistema operativo: in particolare, si interfaccia per la parte Android a Google Cloud Print e per la parte iOS a Apple AirPrint. Il plugin prende in input, attraverso la funzione `print()`, codice HTML e fa il rendering di stampa a livello nativo. L'utente è quindi in grado di lanciare la stampa su una stampante compatibile con i suddetti servizi, oppure di generare un file PDF.

3.6 Backup e ripristino

Il backup dell'app viene effettuato solo per i database diaryDB e agendaDB. PouchDB consente attraverso il plugin PouchDB Replication Stream di effettuare il dump dei database in forma di ReadableStream. [16] Per permettere di utilizzare l'oggetto ReadableStream in Javascript è stata utilizzata la libreria Memorystream. [17] Una volta istanziati gli oggetti contenenti i dati dei dump dei due database, vengono creati due file di testo, dal nome `diaryDB_backup.txt` e `agendaDB_backup.txt`, con il rispettivo contentuto dei dump. Questi file vengono poi inseriti all'interno di un archivio zip grazie alla libreria JSZip, che permette di creare archivi zip in Javascript. [18] Il file zip creato viene poi salvato con il nome `backupPandi.zip` nella relativa cartella dei dati dell'applicazione.

La funzione di ripristino del database invece, si aspetta di trovare un file dal nome `backupPandi.zip` nella cartella dei dati dell'applicazione. Se questo file viene trovato, viene letto il suo contenuto attraverso la libreria JSZip e vengono estratti i file `diaryDB_backup.txt` e `agendaDB_backup.txt`. Per eseguire correttamente il restore è necessario distruggere i database esistenti attraverso la funzione `destroy()` di PouchDB, e in seguito ricrearli. I file vengono quindi caricati nei relativi database vuoti con la funzione `load()` del plugin PouchDB Replication Stream.

In appendice A.4 e A.5 è presente il codice delle funzioni localBackup e localRestore, che effettuano quanto spiegato.

3.7 Localizzazione

Per implementare la localizzazione dell'app è stato scelto di non utilizzare librerie esterne per la gestione delle stringhe e delle traduzioni, ma di scrivere direttamente alcune funzioni Javascript. Questa scelta è stata fatta nell'ottica di garantire le massime performance all'app: infatti le stringhe da tradurre non sono in numero eccessivo (circa un centinaio), rendendo quindi superfluo l'utilizzo di librerie con funzioni di traduzione molto più avanzate di quelle necessarie. Inoltre, poichè la richiesta che l'app implemen-

tasse la localizzazione è arrivata solo quando l'applicazione era quasi completata, in fase progettuale non è stata considerata una gestione unificata delle stringhe per permetterne la traduzione. Ciò ha portato ad avere stringhe dislocate sia su codice HTML che su codice Javascript, rendendone difficile la gestione da un' unica libreria. Le funzioni implementate sono riportate in appendice A.6.

Viene descritta di seguito l'architettura progettata per la localizzazione. Ogni stringa di testo da tradurre è stata sostituita, nel codice HTML e Javascript, da un ID univoco che identifica una stringa. Nel codice HTML, questo ID è stato inserito all'interno del tag speciale `<lng>` che sostituisce il testo originario, mentre nel codice Javascript la stringa è stata sostituita dalla funzione `lng(id)`, che riceve come parametro l'ID della stringa. Le stringhe sono invece state inserite in una matrice, chiamata `langTrans` dove ogni riga rappresenta una diversa stringa. La prima colonna contiene l'ID e ogni altra colonna contiene la stringa tradotta in una determinata lingua.

Il sistema inizializza la localizzazione con la funzione `getSystemLanguage()`: questa ha il compito di individuare la lingua da utilizzare. Come si può vedere dal codice della funzione, viene inizialmente cercato se l'utente ha scelto una determinata lingua nelle impostazioni dell'app: se l'ha fatto, è presente la variabile `f7form-language-switch` nella localstorage, che memorizza la preferenza sulla lingua. Se questa variabile è presente e il suo valore non è default, la funzione sceglie quella lingua. Negli altri casi, essa tenta di identificare la lingua utilizzata dalle impostazioni locali della Webview o del sistema operativo in uso. Se la lingua identificata è presente tra le traduzioni, sceglie quella, altrimenti viene scelta la lingua inglese, che in ogni caso viene sempre considerata come lingua di fallback.

Sono state poi implementate le funzioni `setLanguage()` e `lng(id)` che si occupano di sostituire l'ID con la stringa nella lingua scelta. La funzione `setLanguage()` riguarda la parte HTML di codice e si occupa di sostituire il valore prensente all'interno dei tag `<lng>` con la stringa tradotta, dopo aver salvato l'ID in un attributo `orig`, nel caso servisse per successive traduzioni. Questa funzione viene chiamata da Javascript nella fase di inizializzazione di ogni pagina: ciò assicura che la pagina venga mostrata a video già tradotta correttamente. La funzione `lng(id)` svolge lo stesso compito, ma per le stringhe presenti nel codice Javascript; in questo caso, non c'è bisogno di cercare nei tag perchè la funzione viene chiamata direttamente da Javascript nel punto in cui è necessario generare la stringa tradotta. Entrambe le funzioni accedono direttamente, attraverso l'ID, alla riga corretta della matrice: in questo modo non sono necessarie ricerche sequenziali e la traduzione è molto performante.

Alcuni componenti di Framework7 richiedono l'utilizzo di un array di stringhe per la

localizzazione: è il caso ad esempio del calendario, che richiede un array con i nomi dei mesi o dei giorni della settimana. Per la traduzione di questi elementi è stato necessario utilizzare degli array diversi per ogni lingua: infatti, al momento della creazione del calendario, l'array deve essere già stato istanziato, rendendo impossibile utilizzare la funzione `lng(id)`. Per questo motivo, una funzione in fase di inizializzazione dell'app assegna gli array contententi le stringhe con la lingua corretta alla variabile che identifica univocamente l'array.

3.8 Bug rilevati sui framework utilizzati

Durante la fase di creazione dell'app sono stati rilevati alcuni bug sui framework utilizzati. Non è stato possibile correggere i framework da questi bug ma sono stati escogitati dei workaround per raggiungere ugualmente l'obiettivo desiderato. Tutti i bug trovati e i relativi workaround adottati sono stati segnalati alla comunità sui forum di supporto relativi a ogni framework.

Un primo bug che è stato individuato fin dall'inizio su Framework7 è quello del disallineamento del testo nel campo ora rispetto a tutti gli altri campi, come si può notare dalla figura 3.21. Questo problema si presenta solo nella versione compilata con Adobe Phonegap, mentre non compare visualizzando la pagina da browser. Il problema è quindi probabilmente generato dal sistema operativo, che utilizza impostazioni diverse per tipi di dati diversi. Il workaround adottato è stato quello di forzare l'altezza degli input field in questo modo:

```
<input style="line-height: 40px;" type="time" name="time">
```

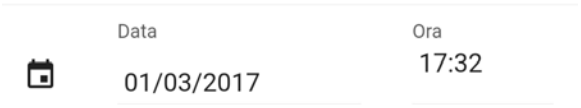

Figura 3.21: *Bug di disallineamento del campo ora.*

Un altro bug, sempre presente su Framework7, riguarda gli input field generati tramite virtual list. Nella schermata pasto infatti, c'è la necessità di generare gli input field della sezione alimenti in modo variabile: ogni volta che l'utente preme sul bottone "aggiungi pasto" deve aggiungersi un input field mentre ogni volta che l'utente preme sul bottone a sinistra dell'input field, esso deve eliminarsi. Per fare ciò, si era pensato di utilizzare una virtual list: in questo modo gli elementi sarebbero stati aggiunti e rimossi

tramite semplici funzioni Javascript. Dopo aver implementato questo sistema, si è scoperto il bug: ogni volta che l'utente tappava sull'input field generato dalla virtual list, l'elemento riceveva e subito perdeva il focus. Questo portava, nel dispositivo mobile, all'impossibilità di scrivere in quell'input field, in quanto la tastiera virtuale appariva e subito dopo scompariva. Il problema è stato subito segnalato sul forum di supporto, e si è scoperto essere già stato affrontato senza risoluzione. Il workaround adottato è stato quello di generare gli input field direttamente con codice Javascript manipolando il DOM della pagina HTML, senza l'ausilio della virtual list. Ciò ha portato in primis a una maggiore complessità del codice, e inoltre ad assumere alcuni compromessi per quanto riguarda il salvataggio nel database dei dati inseriti attraverso questi input field. È stato infatti necessario studiare tutte le casistiche possibili di inserimento e cancellazione che l'utente possa compiere, in modo da assicurare che in nessun caso i dati inseriti andassero persi o venissero visualizzati in maniera errata.

Capitolo 4

Funzionalità App

Come accennato nel capitolo 3, sono state sviluppate le seguenti funzionalità in PanDi:

- **Diario**
- **Agenda**
- **Informazioni pancreas**
- **Stampa report**
- **Backup e ripristino**

Inizialmente, al primo avvio dell'app, vengono mostrate delle slide introduttive che spiegano come utilizzare l'app, come si può notare dalla figura 4.1, che mostra la prima slide. Una volta completata o saltata la guida introduttiva l'app si apre automaticamente nella sezione diario.

Nel seguito, viene dedicato un paragrafo per la descrizione qualitativa di ogni funzionalità.

4.1 Diario

La prima volta che si entra nella sezione diario, essa non contiene elementi. Una card informativa aiuta l'utente a familiarizzare e capire come inserire il primo elemento. Per fare ciò, basta premere sul pulsante tondo verde in basso a destra: compare un menù che permette di scegliere che tipo di evento inserire (farmaco, sintomo, pasto).

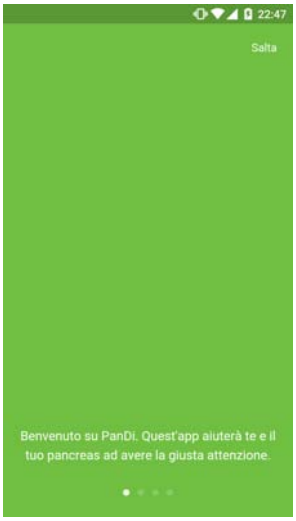

Figura 4.1: *Prima slide introduttiva*

Figura 4.2: *Diario senza elementi nella lista*

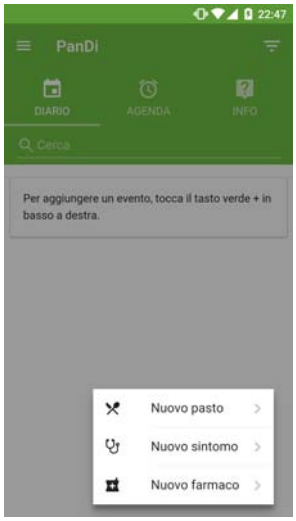

Figura 4.3: *Menù per inserire un nuovo evento*

Una volta selezionato il tipo di evento, compare una schermata dedicata per ogni tipo di evento. L'utente in tutti i casi si troverà già i campi data e ora automaticamente compilati con la data e ora corrente. Per modificarli, basterà selezionarli e si aprirà un form di sistema che consentirà di scegliere altri valori.

Per l'evento farmaco, l'utente dovrà inserire il nome del farmaco, selezionandolo da un elenco a cascata, e la sua unità di misura (compressa, goccia, ecc.). L'inserimento del nome del farmaco è obbligatorio per poter salvare l'elemento. Potrà poi inserire la quantità assunta ed eventuali note. In questo, come in tutti gli altri elenchi a cascata dove è possibile utilizzare la tastiera, l'utente potrà aggiungere elementi non presenti semplicemente scrivendo con la tastiera un valore non presente. Una volta che l'evento viene salvato, l'elemento viene memorizzato e sarà possibile selezionarlo direttamente dall'elenco a cascata.

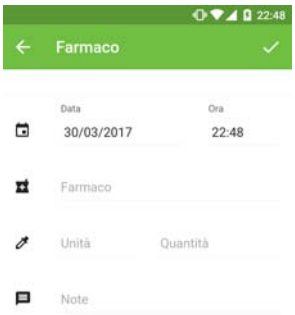

Figura 4.4: *Schermata di inserimento di un farmaco*

Figura 4.5: *Dropdown di inserimento di un farmaco*

Per l'evento sintomo, dovrà inserire il tipo di sintomo, sempre da un elenco a cascata, ed eventuali note. Anche in questo caso, l'inserimento del tipo di sintomo è obbligatorio per il salvataggio. L'utente potrà inoltre aggiungere foto per aiutare la descrizione del sintomo, attraverso l'apposito elemento della lista.

Per l'evento pasto, l'utente dovrà selezionare da un elenco il tipo di pasto in questione (colazione, pranzo, spuntino, etc.), e riportare nella lista sottostante gli alimenti e la relativa quantità in grammi di cui è composto quel pasto. Gli alimenti vengono scelti

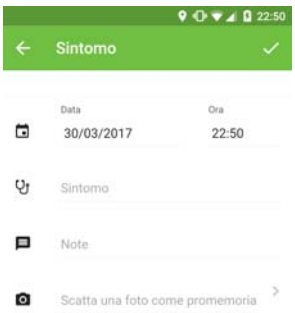

Figura 4.6: *Schermata di inserimento di un sintomo*

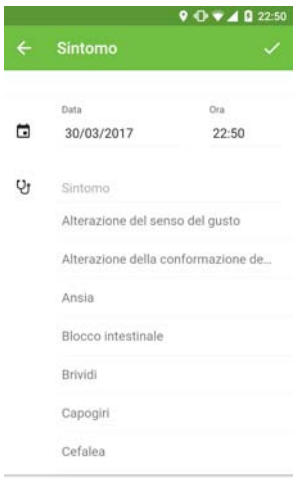

Figura 4.7: *Dropdown di inserimento di un sintomo*

da un elenco a cascata. In seguito, possono essere aggiunte note e foto.

Una volta inseriti gli eventi, essi appaiono nella lista della schermata principale. Gli eventi sono ordinati per data, dal più recente al meno recente, e sono divisi per giorno. È possibile accedere alla schermata completa di ogni evento tappando sul relativo elemento della lista. Da questa schermata è possibile visionare tutti i dettagli inseriti e apportare modifiche. Nel caso l'utente volesse eliminare un evento, può farlo toccando l'icona a forma di cestino. Le foto inserite sono visibili direttamente nella schermata relativa a ogni evento, sotto forma di preview. Per visualizzare le foto intere e per zoomarle, è necessario eseguire un tap su di esse.

Gli eventi possono essere ricercati attraverso la barra di ricerca presente sopra la lista: essa consente la ricerca dei farmaci per nome, mentre dei sintomi e pasti per tipo. Gli eventi possono anche essere filtrati: per fare ciò è necessario aprire la sidebar di destra, attraverso il relativo pulsante in alto a destra, oppure con uno swipe laterale. Da questa sidebar è possibile impostare la lista per visualizzare solo uno o più tipi di eventi oppure, selezionando un giorno dal calendario, scorrere la lista fino al giorno scelto.

4.2 Agenda

La sezione agenda, accessibile dal relativo tasto nella tab bar oppure dalla sidebar di sinistra, contiene nella metà superiore un calendario e nella metà inferiore una lista dove

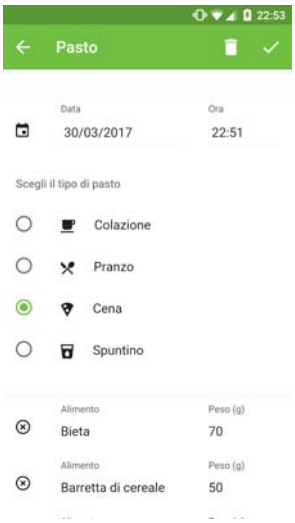

Figura 4.8: *Schermata di inserimento di un pasto*

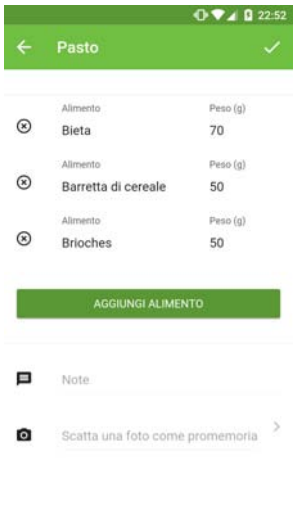

Figura 4.9: *Alimenti che compongono un pasto*

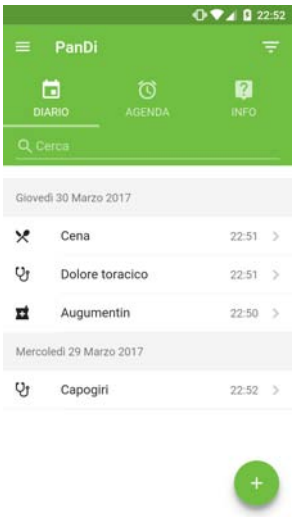

Figura 4.10: *Diario con elementi nella lista*

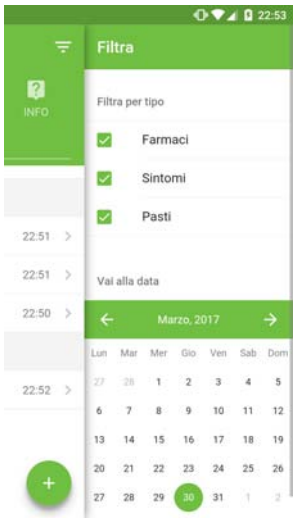

Figura 4.11: *Sidebar per filtrare gli elementi del diario*

vengono visualizzati gli appuntamenti del giorno selezionato. Tappando su un altro numero, è possibile cambiare il giorno selezionato. Inoltre, i giorni in cui sono stati salvati appuntamenti sono rappresentati da un pallino verde sotto il giorno nel calendario.

Per aggiungere un appuntamento, come suggerito dalla card informativa presente nel caso in cui non ci sono appuntamenti salvati, basta premere sul bottone verde in basso a destra. Come per il diario, si apre una schermata che consente di inserire il nome dell'appuntamento, la data e l'ora (anche in questo caso preimpostate di default al valore corrente) ed eventuali note. Per accedere nuovamente alla schermata completa degli eventi salvati basterà tappare sul relativo elemento della lista.

Figura 4.12: *Funzione agenda*

4.3 Informazioni pancreas

In questa sezione, accessibile attraverso il terzo tasto della tab bar oppure dalla sidebar di sinistra, è possibile accedere a informazioni mediche sul pancreas. I vari argomenti disponibili sono visibili in una lista, dalla quale tappando su un elemento si accede alla relativa pagina informativa. Le pagine che si aprono contengono immagini, le quali, attraverso un tap, possono essere zoomate e viste a schermo intero.

4.4 Stampa report

La sezione stampa report, accessibile solo dalla sidebar sinistra, consente di stampare un report degli eventi del diario. La schermata che si apre consente di selezionare

Figura 4.13: *Funzione informazioni pancreas*

il periodo per il quale stampare il report; di default, l'app seleziona l'ultima settimana. Inoltre la schermata consente di filtrare gli eventi per tipo, selezionandone solo alcuni. Il bottone stampa report consente di generare il report e salvarlo in PDF, oppure stamparlo con una stampante compatibile.

4.5 Backup, ripristino e altre funzioni

La sezione backup e ripristino consente di eseguire il backup e ripristino dei dati del diario e dell'agenda. I backup vengono salvati in un archivio zip nella cartella contenente i dati dell'applicazione selezionando il bottone esegui backup locale. Dalla stessa sezione è possibile eseguire il ripristino dei dati da un archivio precedentemente creato. Per eseguirlo correttamente, è sufficiente che l'archivio si trovi nella cartella dei dati dell'applicazione.

La sezione impostazioni, accessibile dal pannello di sinistra, consente di cambiare la lingua dell'app, la dimensione del carattere e di disabilitare le notifiche di aiuto.

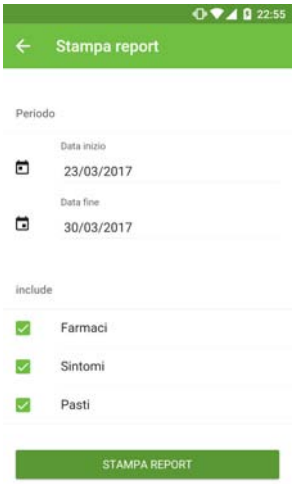

Figura 4.14: *Schermata di selezione del periodo e dei tipi da includere nel report*

Figura 4.15: *Report generato attraverso Google Cloud Print*

Figura 4.16: *Schermata backup*

Figura 4.17: *Schermata impostazioni*

Capitolo 5

Collaudo e presentazioni dell'app

5.1 Beta testing e collaudo iniziale

L'app, dal primo rilascio iniziale al committente, ha subito costantemente una fase di beta testing e debug. L'applicazione è stata testata sui seguenti dispositivi:

- OnePlus X
- Motorola Moto G (1a generazione)
- Motorola Moto G (2a generazione)
- Motorola Moto E (1a generazione)
- Huawei P9 Plus
- Sony M4 Aqua
- Nexus 7 2012
- Samsung Galaxy Note 10
- Apple iPad Air 2

Tutti i dispositivi Android montavano una versione compresa dalla 5.1 alla 7, mentre l'iPad montava iOS 10.

Il principale problema riscontrato su molti dispositivi Android è stato un limite nelle performance di qualche componente grafico di Framework7: ad esempio l'animazione di apertura della sidebar oppure la transizione da floating action button a popover. Inoltre, è emerso che alcuni bug inizialmente venivano riscontrati solo su alcuni dispositivi,

rendendone quindi molto complicato il debug.

Dopo numerose ricerche, è stato possibile individuare che questi problemi erano causati dalle differenti versioni di webview presenti nei vari dispositivi: infatti, oltre al problema che in alcuni dispositivi questa non era aggiornata, si è notato che nelle ROM di qualche dispositivo la webview non era quella stock di Android ma era stata personalizzata dal produttore. Il problema è stato quindi risolto includendo nel pacchetto di installazione dell'applicazione una webview personalizzata, chiamata Cocoon WebView+. In questo modo, si forza il sistema ad usare la webview presente all'interno del pacchetto di installazione, uguale per tutti i dispositivi. Questo ha comportato un aumento della dimensione del pacchetto di circa 30MB e dell'applicazione installata di circa 80MB, ma ha consentito un debug più rapido e uniforme, oltre a performance grafiche nettamente migliorate.

Per testare le performance dell'app con molti elementi nel database, è stata creata una funzione di testing, riportata nell'appendice A.7. Questa funzione, per simulare un reale inserimento da parte dell'utente, crea degli oggetti con dati random e li inserisce ad uno ad uno nel diario. Il numero di oggetti viene selezionato dall'utente in una schermata di test dedicata.

Con questa funzione sono stati inseriti fino a 20000 record all'interno del database diaryDB. Per eseguire i benchmark è stato usato il dispositivo OnePlus X. Dopo l'inserimento sono quindi state testate tutte le varie funzioni dell'app con questo numero di record in memoria. Non sono state notate grosse differenze di performance dovute al fatto di avere un numero così grande di elementi in memoria. Si nota solamente un rallentamento nell'avvio dell'app, dell'ordine di qualche secondo, dovuto al fatto che l'app legge tutti gli elementi dal database e li carica in un array. La creazione della virtual list rimane invece veloce: sebbene riceva un array con tutti gli elementi, la renderizzazione degli elementi avviene solo poco prima che questi vengano caricati a schermo quando l'utente scrolla la lista. Un ulteriore rallentamento è stato riscontrato nella generazione dei report. Questo rallentamento però non è solamente causato dal numero di documenti nel database, ma da quanti di questi devono essere renderizzati nel report (dipende quindi anche dalla lunghezza del periodo selezionato e dai filtri applicati). In ogni caso, i risultati ottenuti sono stati soddisfacenti: per renderizzare un periodo di un mese con all'interno 5190 documenti ci sono voluti 10 secondi, ottenendo un report di 173 pagine.

Un possibile miglioramento a questi lag si può ottenere limitando il numero di documenti acquisiti attraverso la query `all_docs`: in questo modo si garantisce un tempo di avvio indipendente dal numero di documenti presenti. I documenti successivi possono essere caricati, in caso di necessità, quando l'utente arriva ad un determinato punto

nello scorrimento della lista. In ogni caso, sebbene questa tecnica migliori i tempi di caricamento dell'app, peggiora le performance dello scrool della lista, poichè è necessario caricare questi elementi dinamicamente durante l'interazione con l'utente.

5.2 Presentazioni

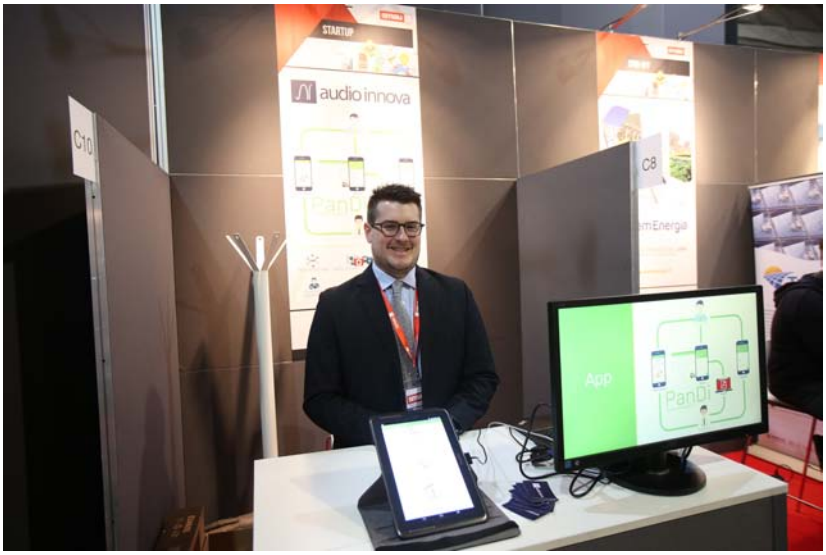

Figura 5.1: *Stand di Audio Innova alla fiera SMAU 2017 Padova*

Nel corso dello sviluppo dell'app grazie all'input del committente si è avuto modo di presentare il concept dell'app ad un primo evento, dal titolo *"I tumori pancreatici: sono possibili una prevenzione e una diagnosi tempestiva?"*, tenutosi a Rimini il 21 gennaio 2017 e organizzato dall'associazione dei pazienti "Oltre la Ricerca".

Questa presentazione ha creato l'opportunità per un successivo incontro di approfondimento a fine febbraio 2017 con il Prof. Alessandro Zerbi, primario della chirurgia pancreatica dell'ospedale Humanitas, Rozzano, e il suo team, in cui è stata presentata una prima versione dell'app. I riscontri sono stati positivi e hanno portato alla programmazione di un altro incontro con il top management del gruppo Humanitas per una valutazione più generale del progetto nel contesto del gruppo.

A inizio marzo 2017 è stato poi organizzato un incontro con il team del Centro del tumore del pancreas dell'Ospedale S. Raffaele diretto dal Prof. Massimo Falconi, costituto da chirurghi, oncologi, ecoendoscopisti, radiologi, radiologi nucleari e radioterapisti. L'incontro ha innescato notevole interesse per un iniziale utilizzo dell'app da

parte dei pazienti del centro nella fase pre-operatoria e post-operatoria. Una prima riunione di verifica dei requisiti specifici del team del pancreas del S. Raffaele è prevista per il 4 aprile 2017.

Come ulteriore risultato dell'incontro del S. Raffaele c'è stato un invito a presentare l'app all'interno della conferenza Europea di Ecoendoscopia, EURO EUS 2017, tenutasi dal 22 al 24 Marzo 2017 a Milano, incentrata in questa edizione sulle tematiche del tumore del pancreas. Il titolo della presentazione è stato: *"How mobile apps can Improve the relationship of patient and physician to affect quality of life for a better outcome"*, P. Rivizzigno.

Un' ulteriore presentazione dell'app, questa volta in ambito informatico, si è tenuta durante l'edizione 2017 della manifestazione SMAU 2017 Padova, tenutasi il 30 e 31 marzo presso la Fiera di Padova. (figura 5.1)

Nel mese di gennaio 2017 è stato creato anche un sito di supporto all'app, `www.pandiapp.it`, dove in questa prima fase vengono raccolti indirizzi e-mail di persone interessate a ricevere maggiori informazioni o a partecipare alla fase di testing dell'app.

Capitolo 6

Sviluppi futuri e conclusioni

Come anticipato nel capitolo 1, lo sviluppo dell'app conclude la prima fase del progetto. Questo prevede in futuro una seconda fase, illustrata nella figura 6.1, nella quale si vuole creare un' interazione diretta tra paziente e medico, in modo tale da consentire al medico di accedere direttamente ai dati raccolti dall'app e al paziente di ricevere direttamente nell'app supporto da parte del medico. Inoltre, si prevede di aggiungere una serie di funzionalità, come ad esempio la generazione di alert per il medico o per il paziente, se si verificano una serie di eventi per cui è necessario che questi vengano avvisati.

Per implementare queste funzioni, è innanzitutto necessaria una gestione centralizzata del database, attraverso un server che raccolga i dati raccolti dai vari dispositivi che utilizzano PanDi. Inoltre, sarà necessaria la creazione di un frontend per computer che si interfacci a questo server, per permettere al medico di accedere ai dati.

Poichè esistono molte complicanze legali derivanti dalla gestione e trattamento di dati sanitari, la seconda fase verrà avviata solo in seguito all'attivazione di una collaborazione con uno o più enti in ambito sanitario. Alla data odierna sono già in corso dei contatti con molteplici potenziali partner, che hanno già dimostrato interesse per il progetto, come illustrato nel paragrafo 5.2.

6.1 Database remoto e sync

Come è già stato spiegato nei paragrafi 2.4 e 3.2, l'app è stata implementata con una tipologia di database che consente nativamente di sincronizzare e replicare i dati con un database remoto. Ogni utente che utilizza l'app, una volta che la seconda fase sarà attiva, dovrà crearsi un account di PanDi e identificarsi all'interno dell'app attraverso il suo account. Questo sarà necessario per poter distinguere ogni utente dagli altri. Dopo

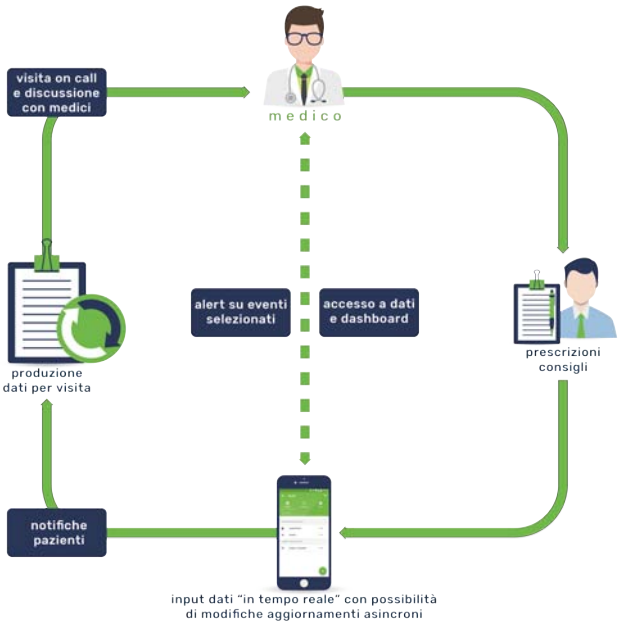

Figura 6.1: *Seconda fase del progetto PanDi*

aver configurato il database dal lato server, dal lato app sarà quindi sufficiente abilitare la sincronizzazione del database locale con il database remoto. Il motore di database installato sull'app si occupera automaticamente di tenere sincronizzati i database, quando vengono fatte delle modifiche e la connessione ad internet è disponibile. Inoltre, sfruttando la sincronizzazione, questo sistema permetterà di poter utilizzare PanDi su più dispositivi contemporaneamente.

Il database remoto verrà implementato attraverso l'applicativo PouchDB Server, che consente di gestire la sincronizzazione remota dei database PouchDB. [19] Anche PouchDB Server è basato su CouchDB, ed è quindi possibile utilizzare tutte le funzioni di CouchDB, come l'autenticazione e la criptazione del database per garantire la sicurezza dei dati.

6.1.1 Implicazioni legali riguardanti la natura dei dati raccolti

Elemento rilevante per le future implementazioni è la corretta conservazione e gestione dei dati raccolti. Le informazioni raccolte secondo quanto indicato dal D.Lgs. 196/2003 (Codice della Privacy) sono da considerare *dati personali*, per quel che concerne i campi identificativi di un eventuale sistema di registrazione, e *sensibili di natura sanitaria* per quelli imputati nella sezione diario.

I dati raccolti nei registri elettronici secondo l'art 22, comma 6 del succitato decreto devono quindi essere trattati con tecniche di cifratura o mediante l'utilizzazione di codici identificativi o di altre soluzioni che, considerato il numero e la natura dei dati trattati, li rendono temporaneamente inintelligibili anche a chi è autorizzato ad accedervi e permettono di identificare gli interessati solo in caso di necessità. La norma specifica inoltre all'art 34, comma 1, che il trattamento di dati personali effettuato con strumenti elettronici è consentito solo se si adottano "tecniche di cifratura o di codici identificativi per determinati trattamenti di dati idonei a rivelare lo stato di salute o la vita sessuale effettuati da organismi sanitari".

Quanto sopra descritto sono tecniche applicabili e consolidate che però non sono sufficienti a permettere l'avvio del progetto in quanto la norma prescrive anche l'identificazione dei soggetti coinvolti e le loro responsabilità, nel dettaglio:

1. Interessato (utente che imputa i dati)
2. Titolare (ente proprietario dell'app/committente)
3. Responsabile (ente gestore dei dati)
4. Incaricato (Soggetto operante sui dati su mandato del titolare o del responsabile)

Delle figure esposte non risultano ancora essere state definite dal committente dell'app la titolarità e la resonsabilità dei dati.

6.2 Frontend per i medici

Le specifiche riguardanti la realizzazione di un frontend medico non sono ancora state definite: si sta decidendo infatti se realizzare un frontend standalone, magari attraverso una piattaforma web dedicata, oppure se implementare qualche interfacciamento con i frontend medici esistenti. Tutto ciò sarà deciso anche in base alle collaborazioni che verranno intraprese con gli enti sanitari e le loro relative esigenze.

6.3 Studi statistici sui dati raccolti

Un altro importante aspetto che è possibile implementare sfruttando il database remoto è lo studio dei dati statistici raccolti. Utilizzando la potenza di calcolo del server è possibile infatti effettuare aggregazioni sui dati e studi di data mining, per permettere di rilevare andamenti statistici che possono migliorare la qualità della cura. Il tutto senza inficiare sulle performance dell'app, in quanto la complessità computazionale dell'operazione è dislocata nel server. Qualora fosse necessario, per effettuare studi statistici

più complessi, si può convertire il database PouchDB presente nel server in un database NoSQL più largamente utilizzato come MongoDB, oppure in un database di tipo SQL come MySQL.

Esistono già casi promettenti sotto questo punto di vista: ad esempio, è stato effettuato uno studio sui dati raccolti da una webinterface sviluppata per tenere traccia di svariati parametri medici e dei sintomi che intercorrono tra le visite di controllo di pazienti malati di tumore al polmone. Lo studio, effettuato su 121 pazienti, ha permesso di aumentare sensibilmente la loro aspettativa di vita rispetto a quelli che non hanno preso parte allo studio. [21]

6.4 Conclusioni

Durante questa esperienza di stage è stata dunque progetta e implementata l'app cross-platform PanDi, che conclude la prima fase del progetto. Sono state soddisfatte tutte le specifiche progettuali e l'app è stata predisposta per la seconda fase, che verrà attivata in seguito a collaborazioni con uno o più enti sanitari. Nell'ultimo periodo l'app è stata oggetto di svariate presentazioni, anche a livello internazionale, ed è risultata di interesse per svariati istituti.

Come è stato illustrato nel paragrafo 5.1, le performance ottenute dall'app sono adeguate al suo utilizzo, considerando che in ogni caso non verranno mai svolte operazioni di calcolo intensivo direttamente dall'app. I problemi di performance grafiche dell'app nel sistema operativo Android, causati della vastità di dispositivi e di webview, spesso non correttamente aggiornate, sono stati risolti embeddando nel pacchetto dell'app una webview più performante rispetto a quella utilizzata dal sistema operativo.

L'esperienza di stage che si è appena conclusa si è rivelata essere stata molto positiva, portando all'acquisizione di competenze ritenute molto utili a livello lavorativo e aziendale: infatti, oltre le conoscenze di programmazione acquisite progettando e sviluppando l'app, grazie alla disponibilità dell'azienda sono state sviluppate molte altre competenze che spaziano anche oltre l'ambito del progetto stesso.

Appendici

Appendice A

Codice

A.1 Funzione buildDiaryList()

```
function buildDiaryList() {
  itemsPerDate = [];
  farmacoIndexes = [];
  sintomoIndexes = [];
  pastoIndexes = [];
  dividerIndexes = [];
  diaryDB.allDocs({
    include_docs: true,
    attachments: true,
    descending: true
  }).then(function (result) {
    if (result.rows.length != 0) {
      //Construction of the first item.
      var item = new Object();
      item.divider = true;
      item.date = result.rows[0].doc.date;
      diaryList.appendItem(item);
      itemFirstDate = new Date(item.date);
      itemFirstDate.setHours(0,0,0,0);
      itemsPerDate[itemFirstDate.getTime()] = 1;
    }
    //For every element of diaryDB, check if the date is
    ↪ not equal to the current date and, in case is not
    ↪ equal, set a new divider item with the new date.
    for (i=0; i<result.rows.length; i++) {
```

```
24          itemDate = new Date(result.rows[i].doc.date);
25          itemDate.setHours(0,0,0,0);
26          if(itemsPerDate[itemDate.getTime()] == undefined) {
27            itemsPerDate[itemDate.getTime()] = 1;
28            var item = new Object();
29            item.divider = true;
30            item.date = result.rows[i].doc.date;
31            diaryList.appendItem(item);
32            diaryList.appendItem(result.rows[i].doc);
33          }
34          else {
35            itemsPerDate[itemDate.getTime()]++;
36            diaryList.appendItem(result.rows[i].doc);
37          }
38        }
39        if (diaryList.items.length == 0) {
40          var diaryHelp =
    ↪ document.getElementById('diary-help');
41          if (!diaryHelp.hasChildNodes()) {
42            var card = document.createElement('div');
43            card.className = 'card-content';
44            card.innerHTML = '<div
    ↪ class="card-content-inner">' +
    ↪ lng("to_add_an_event_tap_the_green_button_
    ↪ at_the_bottom_right") + '</div>';
45            diaryHelp.appendChild(card);
46          }
47        }
48        else {
49          var diaryHelp =
    ↪ document.getElementById('diary-help');
50          if (diaryHelp.hasChildNodes()) {
51            diaryHelp.removeChild(diaryHelp.childNodes[0]);
52          }
53        }
54      }).catch(function (err) {
55        console.log(err);
56      })
57    }
```

A.2 Funzione buildAgendaList()

```
function buildAgendaList(dateTimestamp) {
  var dateVar = new Date(dateTimestamp);
  year = dateVar.getFullYear();
  month = padDate(dateVar.getMonth() + 1);
  day = padDate(dateVar.getDate());
  dateString = year + '-' + month + '-' + day;
  agendaDB.find({
    selector: {date: dateString}
  }).then(function (result) {
    for (var i = 0; i < result.docs.length; i++) {
      agendaList.appendItem(result.docs[i]);
    }
    if (agendaList.items.length == 0) {
      var agendaHelp =
↪ document.getElementById('agenda-help');
      if (!agendaHelp.hasChildNodes()) {
        var card = document.createElement('div');
        card.className = 'card-content';
        card.innerHTML = '<div
↪ class="card-content-inner">' +
↪ lng("to_add_an_event_tap_the_green_button_
↪ at_the_bottom_right") + '</div>';
        agendaHelp.appendChild(card);
      }
    }
    else {
      var agendaHelp =
↪ document.getElementById('agenda-help');
      if (agendaHelp.hasChildNodes()) {
        agendaHelp.removeChild(agendaHelp.childNodes[0]);
      }
    }
  }).catch(function (err) {
    console.log(err);
  });
}
```

A.3 Funzione takePhotoFromCamera

```
1  Dom7(document).on('click', '.take-photo-from-camera',
   ↪ function() {
2    var options = {
3        quality: 50,
4        destinationType: Camera.DestinationType.FILE_URI,
5        sourceType: Camera.PictureSourceType.CAMERA,
6        encodingType: Camera.EncodingType.JPEG,
7        mediaType: Camera.MediaType.PICTURE,
8        allowEdit: false,
9        correctOrientation: true  //Corrects Android
   ↪ orientation quirks
10   }
11   navigator.camera.getPicture(function
   ↪ cameraSuccess(imageUri) {
12     var formData = myApp.formToData('#form');
13     if (formData._id == "") {
14       dateTime = formData.date + 'T' + formData.time;
15       var dateId = new Date(dateTime);
16       //This trick is for prevent the collision of the
   ↪ element with the same date and time.
17       var id = dateId.getTime() +
   ↪ Math.floor((Math.random() * 10000));
18       formData._id = id.toString();
19       myApp.formFromData('#form', formData);
20     }
21     photoList.appendItem(imageUri);
22     window.resolveLocalFileSystemURL
   ↪ (cordova.file.externalDataDirectory, onGetSuccess,
   ↪ onGetFail);
23     function onGetSuccess(fileSystem) {
24       fileSystem.getDirectory(formData._id, {create: true,
   ↪ exclusive: false}, onGetDirectorySuccess,
   ↪ onGetDirectoryFail);
25       function onGetDirectorySuccess(dir) {
26         moveFile(imageUri, dir);
27       }
28       function onGetDirectoryFail(error) {
29         console.log("Error creating directory " +
   ↪ error.code);
```

```
30        }
31      }
32      function onGetFail(error) {
33        console.log("Error getting directory " +
    ↪  error.code);
34      }
35    }, function cameraError(error) {
36        console.debug("Unable to obtain picture: " + error,
    ↪  "app");
37    }, options);
38  });
```

A.4 Funzione localBackup

```
1   Dom7(document).on('click', '.local-backup', function () {
2     var zip = new JSZip();
3
4     var dumpedStringDiary = '';
5     streamDiary.on('data', function(chunk) {
6       dumpedStringDiary += chunk.toString();
7     });
8
9     var dumpedStringAgenda = '';
10    streamAgenda.on('data', function(chunk) {
11      dumpedStringAgenda += chunk.toString();
12    });
13
14    diaryDB.dump(streamDiary).then(function () {
15      zip.file("diaryDB_backup.txt", dumpedStringDiary);
16      agendaDB.dump(streamAgenda).then(function () {
17        zip.file("agendaDB_backup.txt", dumpedStringAgenda);
18        zip.generateAsync({type : "blob"}).then(function
    ↪  (blob) {
19          var folderName =
    ↪  cordova.file.externalDataDirectory;
20          window.resolveLocalFileSystemURL(folderName,
    ↪  onGetSuccess, onGetFail);
21          function onGetSuccess(fileSystem) {
22            fileSystem.getFile("backupPandi.zip", {create:
    ↪  true, exclusive: false}, function(fileEntry) {
```

```
23                writeFile(fileEntry, blob, false);
24              });
25            }
26            function onGetFail(err) {
27              console.log(err);
28            }
29          })
30        });
31    });
32  });
```

A.5 Funzione localRestore

```
1  Dom7(document).on('click', '.local-restore', function () {
2    var zip = new JSZip();
3    var fileName = cordova.file.externalDataDirectory +
   ↪  "/backupPandi.zip";
4
5    JSZipUtils.getBinaryContent(fileName, function(err,
   ↪  data) {
6      if(err) {
7        throw err; // or handle err
8      }
9      zip.loadAsync(data).then(function () {
10       zip.file("diaryDB_backup.txt").async("string").
   ↪  then(function success(content) {
11         diaryDB.destroy().then(function () {
12           diaryDB = new PouchDB('diary', {adapter:
   ↪  'websql', auto_compaction: true});
13           diaryDB.load(content);
14           })
15       }, function error(e) {
16           console.log(e);
17       });
18       zip.file("agendaDB_backup.txt").async("string").
   ↪  then(function success(content) {
19         agendaDB.destroy().then(function () {
20           agendaDB = new PouchDB('agenda', {adapter:
   ↪  'websql', auto_compaction: true});
21           agendaDB.load(content);
```

```
            })
        }, function error(e) {
            console.log(e);
        });
    });
  });
});
```

A.6 Funzioni di localizzazione

```
function getSystemLanguage() {
  var setLanguage =
↪  localStorage.getItem("f7form-language-switch");
  var obj = JSON.parse(setLanguage);
  var sysLang = "en";
  if (setLanguage == null || obj.language == "default") {
    var systemLang = navigator.language ||
↪  navigator.browserLanguage || "";
    if (systemLang.indexOf("it") > -1) {
      sysLang = "it";
    } else {
      sysLang = "en";
    }
  }
  else {
    sysLang = obj.language;
  }
  return sysLang;
}

function setLanguage(lng, block) {
  lng = lng || defaultLanguage;
  currentLanguage = lng;
  if(!block) {
    block = "lng";
  } else {
    block = block + " > lng";
  }
  Dom7(block).each(function () {
    var item = Dom7(this);
```

```
29      var text = item.html();
30      var id;
31      var trans = "";
32      if (!item.attr("orig")) {
33        item.attr("orig", text);
34      }
35      id = item.attr("orig");
36      trans = item.attr("orig");
37      if(langTrans[id] !== undefined) {
38        if(langTrans[id][currentLanguage] !== undefined) {
39          trans = langTrans[id][currentLanguage];
40        } else {
41          if(langTrans[id][defaultLanguage] !== undefined) {
42            trans = langTrans[id][defaultLanguage];
43          }
44        }
45      }
46      item.html(trans);
47    });
48  }
49
50  function lng(id) {
51    var trans;
52    if(langTrans[id] !== undefined) {
53      if(langTrans[id][currentLanguage] !== undefined) {
54        trans = langTrans[id][currentLanguage];
55      } else {
56        if(langTrans[id][defaultLanguage] !== undefined) {
57          trans = langTrans[id][defaultLanguage];
58        }
59      }
60    }
61    if (trans) {
62      id = trans;
63    }
64    return id;
65  }
```

A.7 Funzione di test

```
Dom7.(document).on('click', '.start-test', function () {
  var formData = myApp.formToData('#test-form');

  for (var i=0; i<formData.number; i++) {
    var tempData = new Object;
    var rndDate = randomDate(1553965075000, 1584889875000,
↪  6, 22);
    tempData._id = rndDate.getTime().toString();
    tempData.date = nowDateTimestamp(rndDate.getTime());
    tempData.time = nowTimeTimestamp(rndDate.getTime());
    tempData.type = typeArray[Math.floor(Math.random() *
↪  typeArray.length)];
    tempData.name = nameArray[Math.floor(Math.random() *
↪  nameArray.length)];
    tempData.description =
↪  nameArray[Math.floor(Math.random() *
↪  nameArray.length)];
    diaryDB.put(tempData);
  }

  diaryList.deleteAllItems();
  buildDiaryList();
  mainView.router.back();
});

function randomDate(start, end, startHour, endHour) {
  var date = new Date(+start + Math.random() * (end -
↪  start));
  var hour = startHour + Math.random() * (endHour -
↪  startHour) | 0;
  date.setHours(hour);
  return date;
}
```

Bibliografia

[1] M Ducreux et al. “Cancer of the pancreas: ESMO Clinical Practice Guidelines for diagnosis, treatment and follow-up”.
In: *Annals of Oncology* 26.suppl 5 (2015), pp. v56–v68.

[2] A.A. Stone et al.
The Science of Self-report: Implications for Research and Practice.
Taylor & Francis, 1999. ISBN: 9781135677411.

[3] Trudy L Bush et al. “Self-report and medical record report agreement of selected medical conditions in the elderly.”
In: *American journal of public health* 79.11 (1989), pp. 1554–1556.

[4] Jen Looper. *What is a WebView?* 2015. URL: `http://developer.telerik.com/featured/what-is-a-webview`.

[5] J.M. Wargo. *Apache Cordova 4 Programming*. Mobile Programming.
Pearson Education, 2015. ISBN: 9780134048277.

[6] *Cordova Overview*. URL: `https://cordova.apache.org/docs/en/latest/guide/overview`.

[7] J.M. Wargo. *Apache Cordova API Cookbook*. Mobile Programming.
Pearson Education, 2014. ISBN: 9780133838596.

[8] *Framework7 Documentation*. URL: `https://framework7.io/docs`.

[9] I.G. Clifton.
Android User Interface Design: Implementing Material Design for Developers.
Pearson Education, 2015. ISBN: 9780134191959.

[10] *How much data can PouchDB store?*
URL: `https://pouchdb.com/faq.html#data_limits`.

[11] D. Flanagan. *JavaScript: The Definitive Guide*. Definitive Guide Series.
O’Reilly Media, Incorporated, 2011. ISBN: 9780596805524.

[12] *Cordova Plugin Camera*. URL: `https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-camera`.

[13] *Cordova Plugin File*. URL: https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-file.

[14] *Javascript Templates*.
URL: https://blueimp.github.io/JavaScript-Templates.

[15] *Cordova Plugin Printer*.
URL: https://github.com/katzer/cordova-plugin-printer.

[16] *PouchDB Replication Stream*. URL: https://github.com/nolanlawson/pouchdb-replication-stream.

[17] *Memorystream*.
URL: https://github.com/JSBizon/node-memorystream.

[18] *JSZip*. URL: https://stuk.github.io/jszip.

[19] *PouchDB Server*.
URL: https://github.com/pouchdb/pouchdb-server.

[20] J.M. Humber e R.F. Almeder. *Privacy and Health Care*.
Biomedical Ethics Reviews. Humana Press, 2001. ISBN: 9781592590896.

[21] F Denis et al. "Overall survival in patients with lung cancer using a web-application-guided follow-up compared to standard modalities: Results of phase III randomized trial". In: *J Clin Oncol* 34.suppl; abstr LBA9006 (2016).