Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA

CORSO DI LAUREA IN INFORMATICA

Sviluppo di un'applicazione mobile multipiattaforma per il monitoraggio della qualità dell'aria

Laureando	Lain Daniele
Relatore	Prof.ssa Gaggi Ombretta

ANNO ACCADEMICO 2012/2013

Indice

Elenco delle tabelle

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 ARPA Veneto

ARPA (Agenzia Regionale per la Prevenzione e Protezione Ambientale) Veneto, conosciuta anche come ARPAV, è un'agenzia regionale che si occupa di protezione, prevenzione e controllo ambientale. Ha responsabilità di monitoraggio dell'ambiente, di previsione metereologica e di informazione al cittadino, per tutelare salute e sicurezza della popolazione. Si occupa inoltre della gestione del sistema informativo per il monitoraggio ambientale e per la divulgazione e diffusione dei dati rilevati.

Il SIER, Servizio Informatica e Reti di ARPAV, si occupa della gestione, sviluppo e mantenimento del sistema informativo regionale ambientale. In particolare, si occupa della gestione del portale web dell'agenzia, della gestione della rete informativa, delle banche dati, degli elaboratori e degli strumenti software che realizza. In questo contesto si inserisce il progetto appARPAV.

1.2 Il progetto di stage

Dal sito arpa.veneto.it:

> *Il progetto appARPAV prevede lo sviluppo di applicazioni [Open Source, n.d.r] per smartphone dedicate ai dati ambientali in diretta prodotti dall'Agenzia.*

Al fine di rendere accessibili al pubblico le misurazioni ambientali effettuate da ARPAV attraverso quanti più dispositivi possibili, si vuole realizzare un'applicazione per smartphone che presenti i parametri di qualità dell'aria più comuni: ozono e PM10.
La fruizione dei dati attraverso smartphone deve essere quanto più veloce ed intuitiva possibile, sia a causa delle ridotte dimensioni degli schermi dei dispositivi, che per incontrare le abitudini dell'utente.
Si vogliono quindi presentare non solo i dati validati dagli operatori, ma anche i dati in arrivo in diretta dalle centraline sparse sul territorio, permettendo un monitoraggio quasi in tempo reale dei valori degli inquinanti. Questo in forma di grafici, in modo che l'andamento sia chiaro all'utente.

Inoltre, l'ecosistema dei sistemi operativi per smartphone è frammentato in tre grandi gruppi: Android, iOS e Windows Phone. Ogni piattaforma ha il proprio linguaggio di sviluppo, sia esso Java, Objective-C oppure C#, e questo costringe gli sviluppatori a realizzare tre versioni della propria applicazione, oppure rinunciare ad alcune fasce di utenti.

Il progetto di stage vuole sfruttare le specifiche W3C riguardanti i Widget HTML5 per scrivere un'applicazione con una componente logica scritta in JavaScript, ed una componente di presentazione scritta con le comuni tecnologie web, HTML e CSS, in modo che questa possa funzionare interoperabilmente sui browser dei diversi sistemi operativi per smartphone, comportandosi al tempo stesso come un'applicazione nativa.

Grazie alle tecnologie web, è quindi possibile realizzare un'unica versione dell'applicazione ed ottenerne il suo buon funzionamento su Android, iOS e Windows Phone. I dati di qualità dell'aria saranno così accessibili a quanti più utenti possibili grazie a questa nuova modalità di presentazione, parallela al sito web aziendale.

L'applicazione, come tutte le applicazioni del progetto appARPAV, sarà resa Open Source, ed i sorgenti e documentazione caricati sul repository GitHub del progetto in corrispondenza della prima pubblicazione.

2 Pianificazione del lavoro

Il progetto originario prevedeva lo sviluppo dell'applicazione per dispositivi mobili, in particolare per i dispositivi dotati di sistema operativo Android, iOS e Windows Phone, dall'attività di analisi dei requisiti fino alla pubblicazione finale nei diversi *store* di applicazioni.

Si è predisposto quindi adeguato tempo alle attività di analisi dei requisiti, progettazione, sviluppo e verifica, puntando alla realizzazione dell'applicazione in ogni sua parte, compresa la componente opzionale. Quanto previsto si può osservare nel diagramma di Gantt in figura 2.1.

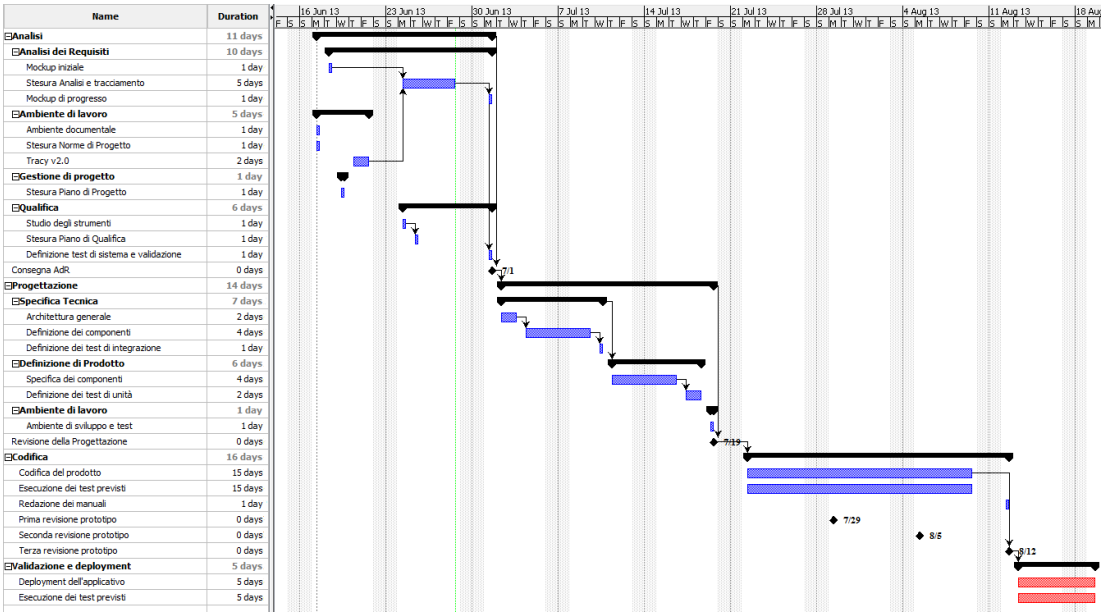

Figura 2.1: Diagramma di Gantt, pianificazione iniziale del lavoro

A seguito dell'attività di analisi dei requisiti, importante nel tempo perché comprensiva di varie riunioni, realizzazione di mockup grafici e successivamente mockup interattivi per illustrare i percorsi utente e l'interazione prevista con l'applicazione, si è rilevato che i dati in formato XML di qualità dell'aria erano insufficienti ai requisiti dell'applicazione, come si illustra in sezione 4.1.1. È emerso quindi il nuovo requisito di realizzazione di un adeguato sistema server, che potesse esportare i dati di qualità dell'aria dal database aziendale in modo adeguato alle esigenze di trasmissione dei dati dell'applicazione.

Questo ha portato ad un necessario sacrificio nelle ore di codifica, e conseguentemente di verifica del codice, dell'applicazione, per trovare il tempo necessario a progettare, implementare e testare adeguatamente tale sistema server.

Il nuovo diagramma di Gantt delle attività previste, che riflette le nuove necessità, è presentato in figura 2.2

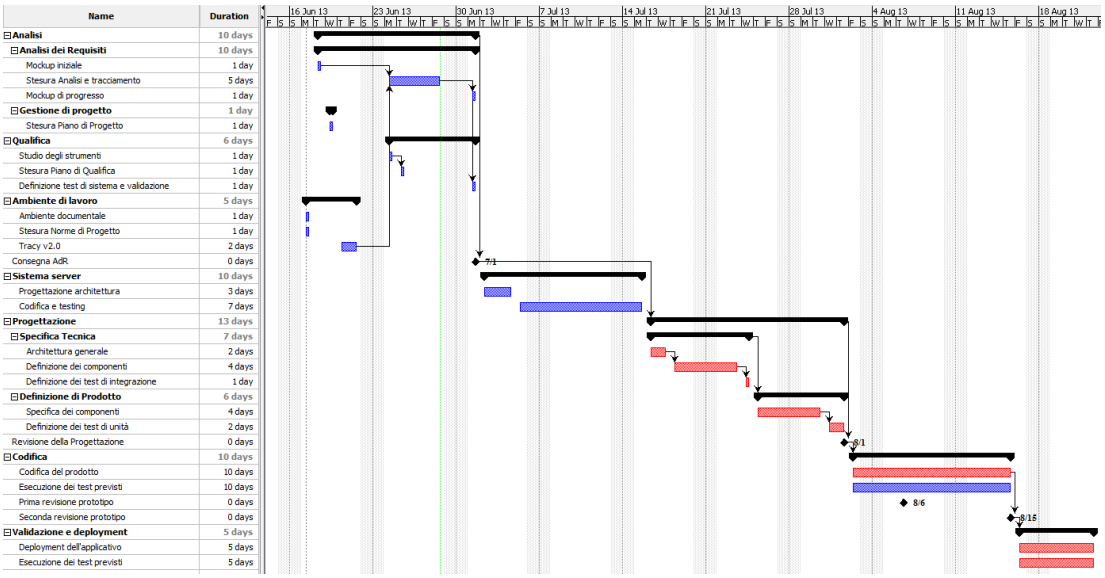

Figura 2.2: Diagramma di Gantt, pianificazione del lavoro con lo sviluppo del sistema server

3 Analisi dei requisiti

Si illustra di seguito l'attività di analisi dei requisiti svolta, procedendo ad una trattazione informale delle prospettive e funzionalità dell'applicazione dedotte dalle riunioni informative, per desumerne successivamente i casi d'uso e requisiti individuati. Il tracciamento dei requisiti conclude la sezione.

3.1 Prospettive dell'applicazione

L'applicazione vuole rendere disponibili agli utenti di smartphone i dati ambientali sulla qualità dell'aria, già presenti nelle banche dati ARPAV, per permetterne un più semplice monitoraggio mediante *app* rispetto all'attuale necessità di visita del sito web ARPAV.
Il prodotto vuole inoltre rendere più comodo e unificato l'accesso ai dati delle varie centraline sparse sul territorio, in modo da fornire all'utente il dato più rilevante rispetto alla sua posizione. Si vuole inoltre permettere all'utente di scegliere di essere avvisato nel caso la qualità dell'aria sia superiore alle soglie massime previste per legge nella zona in cui si trova, o nelle zone di interesse scelte.
Per funzionare, il prodotto necessiterà dei dati sugli inquinanti, che dovranno essere messi a disposizione dalle banche dati ARPAV e scaricati tramite internet sullo smartphone dell'utente.

3.2 Funzionalità offerte

Il programma dovrà fornire un'interfaccia ai dati in diretta e validati di qualità dell'aria rilevati da ARPAV. Sfrutterà la geolocalizzazione per fornire all'utente il dato in diretta più rilevante rispetto alla sua posizione, e permetterà di selezionare la stazione di interesse da una mappa oppure da una lista. Per la stazione selezionata mostrerà i dati rilevanti. In particolare sarà permesso:

- Visualizzare i valori di ozono e PM10 in diretta basati sulla propria posizione
- Visualizzare una mappa delle stazioni
- Visualizzare una lista di stazioni
  - Visualizzare le informazioni della stazione
  - Visualizzare grafici in diretta dei valori di ozono e PM10

  - Visualizzare i dati di legge validati relativi alla stazione
  - Iscriversi al servizio di allerta presso la stazione
- Attivare o disattivare il servizio di allerta ozono

3.3 Caratteristiche degli utenti

Il prodotto si rivolge ad utenti interessati al monitoraggio della qualità dell'aria, sia in tempo reale (con dati in arrivo con una frequenza oraria o giornaliera dalle centraline di controllo) che di consuntivo, con statistiche giornaliere e dati validati dagli operatori ARPA Veneto.
Non emerge quindi il requisito di avere alcuna gerarchia di utenti, o utenti con privilegi differenziati. Pertanto l'applicazione prevede una sola tipologia di utente, ovvero l'utilizzatore finale.
Vista la grande diffusione degli smartphone presso le fasce di utenza più diverse, si deve ritenere molto bassa la capacità tecnica dell'utente finale, che utilizzerà il prodotto occasionalmente, con eventuali notifiche di superamento delle soglie di inquinanti se abilitate, ed in mobilità, quindi utilizzando connessioni ad internet prevalentemente a consumo.

3.4 Vincoli generali

Per poter utilizzare l'applicazione è necessario uno smartphone che esegua uno tra i sistemi operativi Android, iOS oppure Windows Phone.
Tutte le funzioni legate alla geolocalizzazione dell'utente necessitano che lo smartphone di cui dispone sia dotato di sistemi di geolocalizzazione, e che questi vengano resi accessibili alle applicazioni installate. La mancanza di tali dispositivi non deve pregiudicare l'utilizzo delle altre funzioni del prodotto.

3.5 Casi d'uso

Si riportano i casi d'uso individuati durante l'attività di Analisi dei Requisiti, specificando per ognuno il codice gerarchico ed un titolo, seguito poi dalla specificazione degli attori coinvolti, una descrizione testuale dello stesso, eventuali precondizioni, il flusso degli eventi descritto dal caso d'uso, e la sua postcondizione.

Caso d'uso UC1: Operazioni qualità dell'aria

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente ha avviato correttamente il programma e questo è pronto all'uso. L'utente può scegliere l'operazione da svolgere riguardante la qualità dell'aria: può scegliere di visualizzare la mappa delle centraline, la lista delle centraline oppure le opzioni del sistema di notifica

di allerta ozono. Se la posizione è nota, visualizza i dati relativi alle stazioni più vicine per ogni tipo di inquinante;

- **Precondizione**: Il sistema è pronto a ricevere la scelta dell'utente;
- **Flusso principale degli eventi**:
  1. L'utente può visualizzare la mappa delle centraline (UC1.1);
  2. L'utente visualizza i dati per ogni inquinante relativi alla centralina più vicina (UC1.2);
  3. L'utente può visualizzare la lista testuale delle centraline (UC1.3);
  4. L'utente può visualizzare le preferenze relative al sistema di allerta ozono (UC1.4);
- **Postcondizione**: Il sistema ha ricevuto informazioni sull'operazione da svolgere desiderata dall'utente.

Caso d'uso UC1.1: Mappa qualità dell'aria

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente visualizza la mappa navigabile del Veneto, la sua posizione se disponibile e la posizione delle centraline che monitorano la qualità dell'aria. ;
- **Precondizione**: Il sistema ha caricato correttamente la visualizzazione della mappa;
- **Flusso principale degli eventi**:
  1. L'utente può modificare lo zoom della mappa (UC1.1.1);
  2. L'utente può navigare nella mappa (UC1.1.2);
  3. L'utente può effettuare operazioni sulle centraline (UC1.1.3);
- **Inclusioni**:
  1. Quando viene effettuata la modifica allo zoom, alla posizione della mappa o alla localizzazione dell'utente, il sistema aggiorna la visualizzazione della mappa (UC1.1.4);
- **Postcondizione**: Il sistema ha ricevuto informazioni sull'operazione che desidera effettuare l'utente.

Caso d'uso UC1.1.1: Modifica dello zoom

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente può modificare lo zoom della mappa delle centraline, effettuando zoom in (aumento del dettaglio della mappa) oppure zoom out (diminuzione del dettaglio della mappa);
- **Precondizione**: Il sistema è pronto a ricevere il comando di zoom;
- **Flusso principale degli eventi**:
  1. Zoom in (UC1.1.1.1);
  2. Zoom out (UC1.1.1.2);
- **Postcondizione**: Il sistema ha ricevuto correttamente il comando di zoom dell'utente.

Caso d'uso UC1.1.1.1: Zoom in

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente può modificare lo zoom della visualizzazione della scena riducendo l'angolo del campo visivo, ovvero la mappa viene vista a più grande livello di dettaglio;
- **Precondizione**: Il sistema ha caricato correttamente la mappa delle centraline e l'utente desidera modificare lo zoom ;
- **Postcondizione**: Il sistema modifica lo zoom della mappa come desiderato dall'utente.

Caso d'uso UC1.1.1.2: Zoom out

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente può modificare lo zoom della visualizzazione della scena aumentando l'angolo del campo visivo, ovvero la mappa viene vista ad un minore livello di dettaglio;
- **Precondizione**: Il sistema ha caricato correttamente la mappa delle centraline e l'utente desidera modificare lo zoom ;
- **Postcondizione**: Il sistema modifica lo zoom della mappa come desiderato dall'utente.

Caso d'uso UC1.1.2: Navigazione nella mappa

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente può spostare il centro della mappa (operazione chiamata panning);
- **Precondizione**: Il sistema ha caricato la mappa e l'utente desidera effettuare il panning;
- **Postcondizione**: La mappa del sistema ha subito lo spostamento desiderato dall'utente.

Caso d'uso UC1.1.3: Centralina qualità dell'aria

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente può selezionare una centralina tra quelle presenti sulla mappa, visualizzare le informazioni principali riguardanti la centralina selezionata, oppure visualizzare i dettagli della centralina selezionata;
- **Precondizione**: Il sistema ha caricato correttamente la mappa delle centraline e l'utente desidera interagire con le centraline presenti;
- **Flusso principale degli eventi**:
  1. L'utente può selezionare una centralina di rilevazione (UC1.1.3.1);
  2. L'utente può visualizzare le informazioni principali riguardanti la centralina (UC1.1.3.2);
  3. L'utente può visualizzare i dettagli della centralina selezionata (UC1.1.3.3);
- **Estensioni**:
  1. L'utente può iscriversi al sistema di allerta per la centralina selezionata (UC1.1.3.4);
- **Postcondizione**: Il sistema ha ricevuto l'informazione sull'interazione richiesta dall'utente.

Caso d'uso UC1.1.3.1: Selezione di una centralina

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente può selezionare una centralina tra quelle presenti sulla mappa;
- **Precondizione**: Il sistema ha caricato correttamente la mappa delle centraline e l'utente desidera selezionare una centralina;
- **Postcondizione**: Il sistema ha ricevuto l'informazione sulla centralina che l'utente desidera selezionare.

Caso d'uso UC1.1.3.2: Informazioni principali della centralina

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente può visualizzare le informazioni principali relative alla centralina:
  - Nome;
  - Indirizzo;
  - Tipologia;
  - Inquinante rilevato e concentrazione
- **Precondizione**: Il sistema presenta una centralina selezionata e l'utente desidera visualizzarne le informazioni principali;
- **Postcondizione**: Il sistema ha fornito all'utente le informazioni principali relative alla centralina selezionata.

Caso d'uso UC1.1.3.3: Informazioni dettagliate della centralina

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente può visualizzare le informazioni dettagliate della centralina selezionata, e vedere il grafico in tempo reale dei dati di concentrazione degli inquinanti rilevati. Può inoltre iscriversi al servizio di allerta della centralina, nel caso questa sia una centralina di rilevazione dell'ozono;
- **Precondizione**: Il sistema presenta una centralina selezionata e l'utente desidera visualizzarne le informazioni dettagliate;
- **Postcondizione**: Il sistema ha visualizzato le informazioni dettagliate relative alla centralina.

Caso d'uso UC1.1.3.4: Iscrizione al sistema di allerta

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente può iscriversi al sistema di allerta per la centralina selezionata;
- **Precondizione**: Il sistema ha selezionato correttamente una centralina che monitora la concentrazione di ozono;
- **Postcondizione**: Il sistema ha ricevuto la richiesta dell'utente di iscrizione al servizio di allerta per la centralina selezionata.

Caso d'uso UC1.1.4: Aggiornamento visualizzazione mappa

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente visualizza la mappa delle centraline aggiornata;
- **Precondizione**: Il sistema ha ricevuto delle modifiche da applicare alla mappa;
- **Postcondizione**: Il sistema ha applicato le modifiche alla mappa e le ha mostrate all'utente.

Caso d'uso UC1.2: Riassunto dei valori degli inquinanti

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente visualizza i dati relativi agli inquinanti presenti, mostrando per ogni inquinante il valore attuale registrato nella centralina più vicina;
- **Precondizione**: Il sistema visualizza la mappa e il dispositivo in uso permette l'accesso alle funzioni di geolocalizzazione;
- **Postcondizione**: Il sistema visualizza i valori relativi agli inquinanti delle centraline più vicine alla posizione dell'utente.

Caso d'uso UC1.3: Lista delle centraline

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente può visualizzare la lista testuale delle centraline, divise per provincia. È possibile selezionare una centralina tra quelle presenti, e visualizzarne le informazioni dettagliate;
- **Precondizione**: Il sistema ha caricato la lista delle centraline;
- **Flusso principale degli eventi**:
  1. L'utente può selezionare una centralina tra quelle presenti (UC1.3.1);
  2. L'utente può visualizzare le informazioni dettagliate della centralina selezionata (UC1.3.2);
- **Inclusioni**:
  1. L'utente visualizza le informazioni generali riguardanti la centralina (UC1.3.4);
- **Estensioni**:

1. L'utente può iscriversi al servizio di allerta ozono per la centralina selezionata (UC1.3.3);

- **Postcondizione**: Il sistema ha ricevuto informazioni sull'operazione che desidera fare l'utente.

Caso d'uso UC1.3.1: Selezione di una centralina

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente può selezionare una centralina tra le centraline proposte;
- **Precondizione**: Il sistema presenta la lista delle centraline presenti e l'utente desidera selezionarne una;
- **Postcondizione**: Il sistema ha ricevuto l'informazione sulla centralina che l'utente desidera selezionare.

Caso d'uso UC1.3.2: Visualizzazione informazioni dettagliate

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente visualizza le informazioni dettagliate relative alla centralina selezionata, quali le concentrazioni di inquinanti e i grafici di andamento. Può scegliere di iscriversi al servizio di allerta, oppure visualizzare i dati validati dagli operatori.;
- **Precondizione**: Il sistema presenta una centralina selezionata e l'utente desidera visualizzare le informazioni dettagliate;
- **Flusso principale degli eventi**:
  1. L'utente può visualizzare i dati di legge validati (UC1.3.2.1);
- **Postcondizione**: Il sistema ha visualizzato le informazioni dettagliate relative alla centralina.

Caso d'uso UC1.3.2.1: Visualizzazione dati validati

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente può scegliere di visualizzare i dati di legge validati dagli operatori ARPAV.;
- **Precondizione**: Il sistema presenta una centralina selezionata;
- **Postcondizione**: Il sistema ha ricevuto la richiesta e visualizzato i dati validati.

Caso d'uso UC1.3.3: Iscrizione al sistema di allerta

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente può iscriversi al sistema di allerta ozono della centralina selezionata, se questa rileva la presenza dell'inquinante;
- **Precondizione**: Il sistema presenta una centralina di rilevazione dell'ozono selezionata;
- **Postcondizione**: Il sistema ha ricevuto la richiesta di iscrizione al sistema di allerta ozono.

Caso d'uso UC1.3.4: Informazioni principali della centralina

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente visualizza le informazioni principali relative alla centralina, quali nome, indirizzo e concentrazione di inquinanti rilevata;
- **Precondizione**: Il sistema presenta una centralina selezionata;
- **Postcondizione**: Il sistema visualizza le informazioni principali relative alla centralina.

Caso d'uso UC1.4: Preferenze del sistema allerta ozono

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente visualizza le preferenze del sistema allerta ozono: può visualizzare la lista delle centraline presso le quali è registrato e attivare o disattivare il sistema di allerta;
- **Precondizione**: Il sistema ha recuperato le preferenze utente;
- **Flusso principale degli eventi**:
  1. L'utente può attivare o disattivare il sistema di allerta ozono (UC1.4.1);
  2. L'utente può visualizzare la lista delle centraline presso le quali è registrato (UC1.4.2);
- **Postcondizione**: Il sistema ha visualizzato le preferenze utente.

Caso d'uso UC1.4.1: Attivazione del sistema di allerta

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente può attivare e disattivare il sistema di allerta ozono per le centraline presso le quali si è registrato;

- **Precondizione**: Il sistema ha caricato correttamente lo stato del sistema di allerta ozono e l'utente desidera modificarne l'attivazione;
- **Postcondizione**: Il sistema ha modificato lo stato del sistema di allerta come selezionato dall'utente.

Caso d'uso UC1.4.2: Visualizzazione centraline registrate

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente può visualizzare le centraline presso le quali ha impostato il servizio di allerta. Se lo desidera, può disiscriversi da una centralina;
- **Precondizione**: Il sistema ha caricato la lista delle centraline presso le quali l'utente è registrato;
- **Flusso principale degli eventi**:
  1. L'utente può disiscriversi dal servizio di allerta di una centralina (UC1.4.2.1);
- **Postcondizione**: Il sistema visualizza la lista delle centraline presso le quali l'utente è registrato.

Caso d'uso UC1.4.2.1: Disiscrizione da una centralina

- **Attori**: Utente;
- **Scopo e descrizione**: L'utente può disiscriversi dal servizio di allerta ozono per una data centralina;
- **Precondizione**: Il sistema presenta la lista delle centraline presso le quali l'utente è iscritto e l'utente desidera cancellare l'iscrizione ad una di esse;
- **Postcondizione**: Il sistema ha disiscritto l'utente dal servizio allerta ozono per la centralina selezionata.

3.6 Specifica dei requisiti

I casi d'uso individuati e le riunioni informative effettuate durante lo stage hanno portato all'individuazione dei requisiti software elencati di seguito.

3.6.1 Requisiti funzionali

Codice	Tipologia	Descrizione	Fonti
RF1	Obbligatorio	L'utente deve poter selezionare le operazioni relative alla qualità dell'aria	UC1
- RF1.1	Obbligatorio	L'utente deve poter scegliere di visualizzare la mappa relativa alla qualità dell'aria	UC1 UC1.1
- RF1.2	Desiderabile	L'utente deve poter scegliere di visualizzare la lista delle centraline relativa alla qualità dell'aria	UC1 UC1.3
- RF1.3	Desiderabile	Il programma deve fornire i dati rilevati nella centralina più vicina per ogni inquinante	UC1.2
RF2	Desiderabile	Il programma deve avvisare l'utente in caso di allerta relativa ai livelli di ozono	Capitolato
- RF2.1	Desiderabile	L'utente deve poter scegliere per quali centraline attivare il servizio allerta	UC1.3.3 UC1.1.3.4 UC1.4
– RF2.1.1	Obbligatorio	L'utente deve poter iscriversi al servizio allerta ozono per una data centralina	UC1.1.3.4
– RF2.1.2	Desiderabile	L'utente deve poter disiscriversi dal servizio allerta ozono per una data centralina	UC1.4.2.1
– RF2.1.3	Obbligatorio	L'utente deve poter visualizzare le centraline presso le quali è registrato al servizio di allerta	UC1.4.2
- RF2.2	Desiderabile	L'utente deve poter attivare o disattivare il servizio di allerta	UC1.4.1
RF3	Obbligatorio	Il programma deve fornire una mappa del Veneto	UC1

- RF3.1	Desiderabile	L'utente deve poter navigare all'interno della mappa	UC1.1.2 UC1.1.4
- RF3.2	Desiderabile	L'utente deve poter modificare lo zoom della mappa	UC1.1.1 UC1.1.1.1 UC1.1.1.2 UC1.1.4
- RF3.3	Desiderabile	L'utente deve poter visualizzare la propria posizione sulla mappa	UC1.1
- RF3.4	Obbligatorio	La mappa deve mostrare le centraline di rilevazione della qualità dell'aria	UC1.1 UC1.1.3
– RF3.4.1	Opzionale	Le centraline troppo vicine tra loro devono essere raggruppate con un unico simbolo	Interno
– RF3.4.2	Obbligatorio	L'utente deve poter selezionare una centralina	UC1.1.3.1 UC1.3.1
— RF3.4.2.1	Obbligatorio	Il programma deve presentare le informazioni di base della centralina selezionata	UC1.1.3.2
— RF3.4.2.2	Obbligatorio	L'utente deve poter visualizzare le informazioni dettagliate della centralina selezionata	UC1.1.3.3
—- RF3.4.2.2.1	Obbligatorio	L'utente deve poter visualizzare i dati validati dagli operatori per la centralina	UC1.3.2.1
—- RF3.4.2.2.2	Obbligatorio	Il programma deve visualizzare grafici di andamento degli inquinanti	UC1.1.3.3 UC1.3.2
—— RF3.4.2.2.2.1	Obbligatorio	Il programma deve visualizzare l'andamento dell'ozono nelle precedenti 48 ore	Capitolato
—— RF3.4.2.2.2.2	Obbligatorio	L'utente deve poter visualizzare l'andamento di PM10 dei 7 giorni precedenti	Capitolato

—— RF3.4.2.2.2.3	Desiderabile	Deve essere possibile visualizzare il valore dei punti rappresentati sui grafici	Capitolato
— RF3.4.2.3	Desiderabile	L'utente deve poter selezionare una centralina dalla lista delle centraline	UC1.3.1
— RF3.4.2.4	Obbligatorio	L'utente deve poter selezionare una centralina dalla mappa	UC1.1.3
RF4	Obbligatorio	Il programma deve recuperare i dati aggiornati all'avvio	Capitolato
- RF4.1	Obbligatorio	Il programma deve recuperare i dati di ozono relativi alle 48 ore precedenti al momento dell'avvio	Capitolato
– RF4.1.1	Opzionale	Se sono presenti dati al momento dell'avvio, il programma deve recuperare solo le informazioni mancanti	Interno
- RF4.2	Obbligatorio	Il programma deve recuperare i dati di PM10 relativi ai 7 giorni precedenti al momento dell'avvio	Capitolato

Tabella 3.1: Tabella dei requisiti funzionali

3.6.2 Requisiti qualitativi

Codice	Tipologia	Descrizione	Fonti
RQ9	Obbligatorio	Devono essere rispettate tutte le metriche sulla stesura di codice stabilite	Interno
RQ10	Obbligatorio	Deve essere prodotta documentazione del codice sorgente del software	Interno

Tabella 3.2: Tabella dei requisiti qualitativi

3.6.3 Requisiti di vincolo

Codice	Tipologia	Descrizione	Fonti
RV5	Obbligatorio	Il programma deve funzionare su Android 2.3	Capitolato
RV6	Obbligatorio	Il programma deve funzionare su Android 4.2	Capitolato
RV7	Obbligatorio	Il programma deve funzionare su iOS6	Capitolato
RV8	Obbligatorio	Il programma deve funzionare su Windows Phone 7	Capitolato

Tabella 3.3: Tabella dei requisiti di vincolo

3.7 Tracciamento requisiti

3.7.0.1 Tracciamento fonti-requisiti

Fonte		Requisiti
	Capitolato	RF2 RF3.4.2.2.2.1 RF4 RF4.1 RV5 RV6 RV7 RV8 RF3.4.2.2.2.2 RF4.2 RF3.4.2.2.2.3
	Interno	RF3.4.1 RF4.1.1 RQ9 RQ10
UC1	Operazioni qualità dell'aria	RF1 RF1.1 RF1.2 RF3
- UC1.1	Mappa qualità dell'aria	RF1.1 RF3.3 RF3.4
– UC1.1.1	Modifica dello zoom	RF3.2
— UC1.1.1.1	Zoom in	RF3.2
— UC1.1.1.2	Zoom out	RF3.2
– UC1.1.2	Navigazione nella mappa	RF3.1
– UC1.1.3	Centralina qualità dell'aria	RF3.4 RF3.4.2.4
— UC1.1.3.1	Selezione di una centralina	RF3.4.2

— UC1.1.3.2	Informazioni principali della centralina	RF3.4.2.1
— UC1.1.3.3	Informazioni dettagliate della centralina	RF3.4.2.2 RF3.4.2.2.2
— UC1.1.3.4	Iscrizione al sistema di allerta	RF2.1 RF2.1.1
– UC1.1.4	Aggiornamento visualizzazione mappa	RF3.1 RF3.2
- UC1.2	Riassunto dei valori degli inquinanti	RF1.3
- UC1.3	Lista delle centraline	RF1.2
– UC1.3.1	Selezione di una centralina	RF3.4.2 RF3.4.2.3
– UC1.3.2	Visualizzazione informazioni dettagliate	RF3.4.2.2.2
— UC1.3.2.1	Visualizzazione dati validati	RF3.4.2.2.1
– UC1.3.3	Iscrizione al sistema di allerta	RF2.1
– UC1.3.4	Informazioni principali della centralina	
- UC1.4	Preferenze del sistema allerta ozono	RF2.1
– UC1.4.1	Attivazione del sistema di allerta	RF2.2
– UC1.4.2	Visualizzazione centraline registrate	RF2.1.3
— UC1.4.2.1	Disiscrizione da una centralina	RF2.1.2

Tabella 3.4: Tabella di tracciamento fonti-requisiti

3.8 Test di Validazione

Si descrive di seguito la strategia di test di validazione finale del prodotto. Per ogni test si riporteranno in modo chiaro i vari passi necessari per la sua esecuzione. Ogni passo che risponde alla verifica di un requisito sarà identificato da un codice progressivo e gerarchico per identificarlo.

- **Test di validazione TV1**: Si verificheranno con il seguente test il buon funzionamento dell'applicazione e delle varie funzionalità offerte all'utente: mappa, lista delle stazioni e dettagli per una singola stazione. Per la sua esecuzione, al *beta tester* è richiesto:
  - Avviare l'applicazione e verificare la presenza delle opzioni di qualità dell'aria **(TV1.1)**:
    - Verificare la possibilità di selezione della mappa e della sua corretta visualizzazione **(TV1.1.1)**:
      - Verificare che la mappa mostri la regione Veneto **(TV1.1.1.1)**;
      - Verificare che la mappa permetta lo spostamento del punto di vista **(TV1.1.1.2)**;
      - Verificare che la mappa permetta lo zoom, sia per aumentare che per diminuire il livello di dettaglio **(TV1.1.1.3)**;
      - Verificare che la mappa visualizzi un segnaposto, se il dispositivo permette l'accesso alla posizione corrente da parte delle applicazioni, nella posizione dell'utente **(TV1.1.1.4)**;
      - Verificare che, data una centralina dalle coordinate note, il suo segnaposto si trovi nella posizione corretta **(TV1.1.1.5)**;
      - Verificare che la mappa al minor livello di dettaglio raggruppa le stazioni del centro di Padova in un unico marcatore **(TV1.1.1.6)**;
      - Verificare che la mappa permetta la selezione di un segnalino rappresentante una singola stazione **(TV1.1.1.7)**;
      - Verificare che la selezione del marcatore della stazione provochi la visualizzazione delle informazioni di base della stazione **(TV1.1.1.8)**;
      - Verificare che la sia possibile la selezione di visualizzazione delle informazioni dettagliate della stazione **(TV1.1.1.9)**;
      - Verificare che tra le informazioni dettagliate della stazione sia possibile visualizzare i dati validati dagli operatori **(TV1.1.1.10)**;
    - Verificare la possibilità di selezione della lista di stazioni e la sua corretta visualizzazione **(TV1.1.2)**:
      - Verificare la possibilità di selezione di una stazione dalla lista di stazione **(TV1.1.2.1)**;
  - Verificare la presenza dei dati di qualità dell'aria per la stazione conosciuta come la più vicina alla posizione rilevata per ogni inquinante **(TV1.2)**;

- **Test di validazione TV2**: Si verificherà con il seguente test il funzionamento del servizio di allerta ozono. Dopo l'avvio dell'applicazione, al *beta tester* è richiesto:
  - Verificare che per le stazioni che rilevano ozono sia possibile l'iscrizione al servizio di allerta utente **(TV2.1)**;
  - Iscriversi al servizio di allerta ozono per due delle stazioni di rilevamento dell'inquinante **(TV2.2)**;
  - Disiscriversi al servizio di allerta ozono per una delle stazioni di rilevamento dell'inquinante presso la quale ci si è precedentemente iscritti **(TV2.3)**;
  - Verificare presso le impostazioni di allerta che la stazione presso la quale ci si è iscritti sia presente, e che la stazione presso la quale ci si è iscritti e disiscritti non sia presente **(TV2.4)**;
  - Verificare presso le impostazioni di allerta che sia possibile iscriversi e disiscriversi al servizio di allerta **(TV2.5)**;
  - Verificare che una notifica di allerta relativa ad una stazione presso la quale ci si è iscritti, a servizio di allerta attivo, sia ricevuta e visualizzata correttamente dal dispositivo, e non visualizzata in caso di servizio allerta disattivato o stazione non presente tra le preferite **(TV2.6)**;
- **Test di validazione TV3**: Si verificherà il buon funzionamento della pagina relativa ai dettagli di una stazione selezionata. Dopo l'avvio dell'applicazione, al *beta tester* è richiesto:
  - Selezionare una stazione che rileva ozono e verificare la presenza del relativo grafico di andamento delle 48 ore precedenti **(TV3.1)**;
  - Selezionare una stazione che rileva PM10 e verificare la presenza del relativo grafico di andamento dei 7 giorni precedenti **(TV3.2)**;
  - Dato un grafico rappresentante l'andamento di ozono, verificare che sia possibile selezionare un punto, rappresentante una rilevazione oraria o giornaliera, e visualizzarne le informazioni. Eseguire la stessa operazione per un grafico di andamento dei PM10 **(TV3.3)**;
- **Test di validazione TV4**: Si verificherà il corretto recupero dei dati aggiornati dell'applicazione. Dopo l'avvio dell'applicazione, e a traffico dati attivo (tramite rete mobile o wireless) al *beta tester* è richiesto:
  - Selezionare una stazione che rileva ozono e verificare che l'ultimo dato rappresentato sia l'ultimo dato valido presente sul server **(TV4.1)**;
  - Selezionare una stazione che rileva PM10 e verificare che l'ultimo dato rappresentato sia l'ultimo dato valido presente sul server **(TV4.2)**;
  - Se l'applicazione è stata avviata precedentemente tra 7 giorni e 2 ore prima dell'orario di esecuzione del presente test, verificare che i dati scaricati dal server siano minori della dimensione del file presente sul server **(TV4.3)**;

3.9 Tracciamento requisiti-test di validazione

Codice	Descrizione del test	Codice del test
RF1	Si verifica il buon avvio del programma e la presenza delle opzioni di qualità dell'aria	TV1.1
- RF1.1	Si verifica la possibilità di selezione dell'opzione mappa	TV1.1.1
- RF1.2	Si verifica la possibilità di selezione dell'opzione elenco centraline	TV1.1.2
- RF1.3	Si verifica la presenza dei dati di qualità dell'aria relativi alla più vicina centralina per ogni inquinante	TV1.2
RF2	Si verifica la ricezione di una notifica di allerta per una data stazione	TV2.6
- RF2.1	Si verifica che ogni stazione che rileva ozono preveda la possibilità di iscrizione al servizio di allerta	TV2.1
– RF2.1.1	Si verifica la possibilità di iscrizione al servizio di allerta per una stazione campione	TV2.2
– RF2.1.2	Si verifica la possibilità di disiscrizione al servizio di allerta per una stazione campione	TV2.3
– RF2.1.3	Si verifica la presenza delle stazioni presso le quali ci si è iscritti al servizio di allerta	TV2.4
- RF2.2	Si verifica la possibilità di attivazione e disattivazione del servizio di allerta	TV2.5
RF3	Si verifica la presenza della mappa della regione veneto	TV1.1.1.1
- RF3.1	Si verifica che la navigazione all'interno della mappa sia permessa	TV1.1.1.2
- RF3.2	Si verifica che la modifica dello zoom della mappa sia permessa	TV1.1.1.3

- RF3.3	Si verifica che in corrispondenza della posizione presunta sia presente il segnalino della propria posizione	TV1.1.1.4
- RF3.4	Si verifica la presenza del segnalino in posizione corretta per un dato numero di stazioni campione	TV1.1.1.5
– RF3.4.1	Si verifica che a minor livello di dettaglio permesso le stazioni del centro di Padova siano raggruppate in un unico segnalino	TV1.1.1.6
– RF3.4.2	Si verifica la possibilità di selezionare un segnalino indicante una stazione campione	TV1.1.1.7
— RF3.4.2.1	Si verifica la comparsa e correttezza delle informazioni riguardanti la stazione selezionata	TV1.1.1.8
— RF3.4.2.2	Si verifica la possibilità di selezione della visualizzazione dettagliata della stazione	TV1.1.1.9
—- RF3.4.2.2.1	Si verifica la presenza dei dati validati dagli operatori per la stazione selezionata	TV1.1.1.10
—- RF3.4.2.2.2	Verificato tramite verifica dei requisiti figli	TV3
—— RF3.4.2.2.2.1	Si verifica la presenza del grafico di andamento dell'ozono nelle 48 ore precedenti, se la stazione selezionata lo rileva	TV3.1
—— RF3.4.2.2.2.2	Si verifica la presenza del grafico di andamento del PM10 per i 7 giorni precedenti, se la stazione selezionata lo rileva	TV3.2
—— RF3.4.2.2.2.3	Si verifica la possibilità di selezione di un punto del grafico e la comparsa di informazioni relative al punto selezionato	TV3.3

— RF3.4.2.3	Si verifica la possibilità di selezione di una centralina dalla lista	TV1.1.2.1
— RF3.4.2.4	Si verifica la possibilità di selezione di una centralina dalla mappa	TV1.1.1.7
RF4	Verificato tramite verifica dei requisiti figli	TV4
- RF4.1	Si verifica che gli ultimi dati di ozono per una data stazione siano gli ultimi pubblicati dal server	TV4.1
– RF4.1.1	Si verifica che il consumo di dati in download all'avvio sia minore della dimensione dell'attuale file JSON, se il programma è stato avviato almeno una volta durante i precedenti 7 giorni	TV4.3
- RF4.2	Si verifica che gli ultimi dati di PM10 per una data stazione siano gli ultimi pubblicati dal server	TV4.2

Tabella 3.5: Tabella di tracciamento requisiti-test di validazione

3.10 Mockup

A seguito delle riunioni svolte e dell'attività di analisi dei requisiti, per verificare la buona comprensione delle funzionalità richieste al prodotto e per presentare l'interazione utente immaginata in questo stadio preliminare sono stati realizzati dei mockup, prima grafici e poi interattivi, dei quali si riportano ed illustrano alcuni *screenshot*.

Figura 3.1: Primi mockup dell'applicazione, schermata iniziale

L'immagine 3.1 mostra un primo layout pensato, adatto a piccoli schermi grazie agli elementi quali pulsanti e voci del menù a dimensione di blocco, e che presenta un menu dal quale accedere a tutte le funzionalità previste. Si può rilevare come questo prototipo sia stato realizzato durante l'attività di analisi dei requisiti, perché prevedeva la divisione delle stazioni per inquinante rilevato, requisito

poi abbandonato da ARPAV a favore di un'unica presentazione delle stazioni di rilevamento di ogni inquinante rilevante.

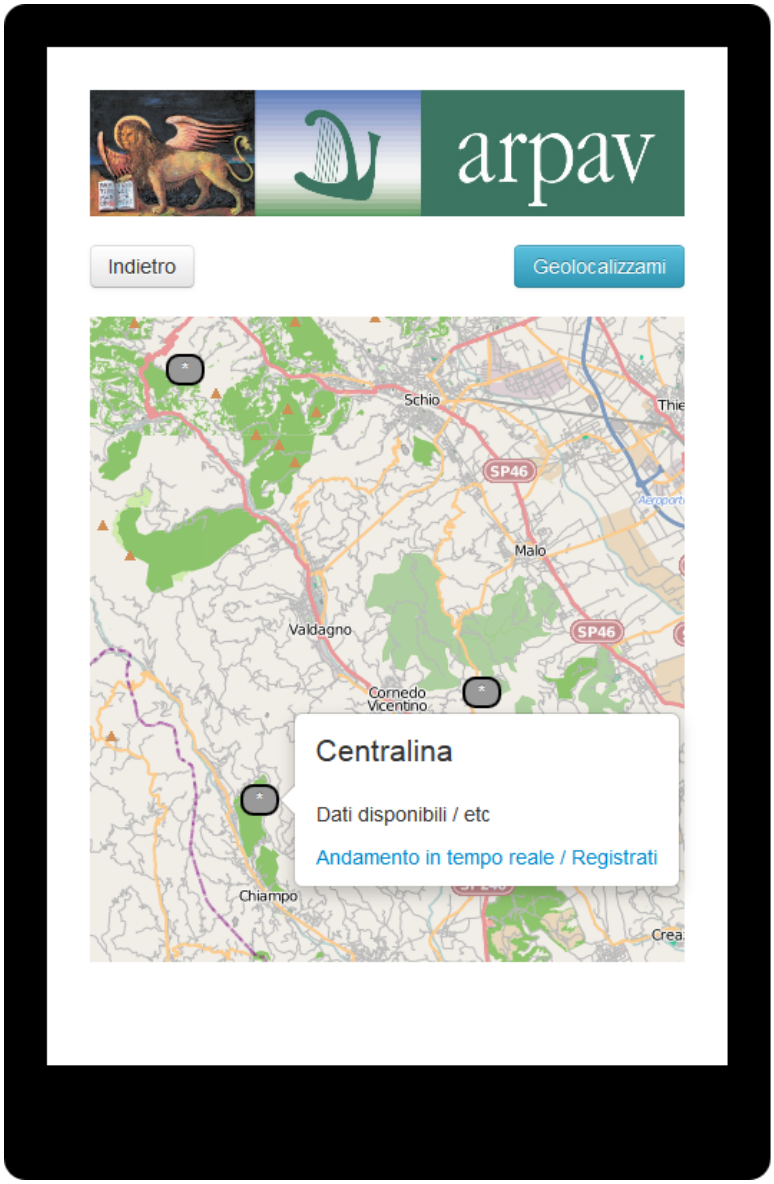

Figura 3.2: Primi mockup dell'applicazione, schermata di mappa

L'immagine 3.2 mostra l'integrazione della mappa all'interno dell'applicazione proposta, con un indicatore opportuno a segnalare le posizioni delle stazioni di rilevamento degli inquinanti dell'aria. Gli indicatori sono pensati come interattivi, in grado di mostrare un popup con le informazioni rilevanti a seguito dell'azione dell'utente.

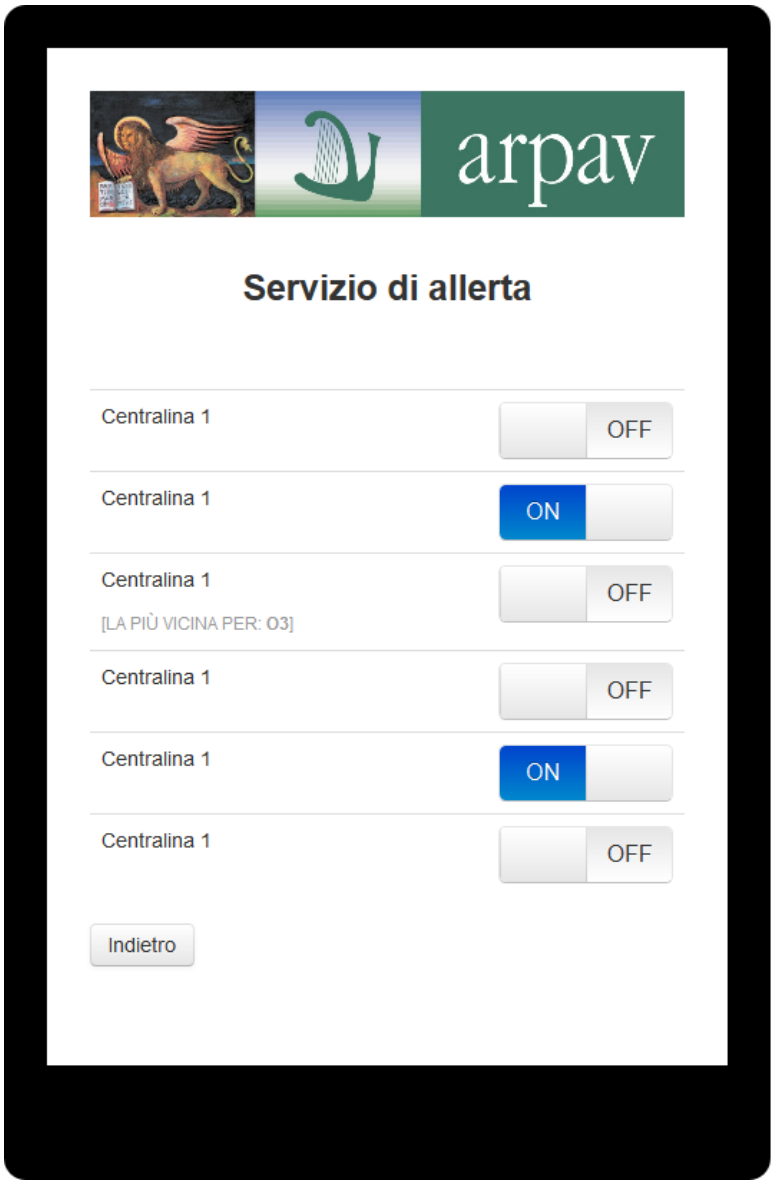

Figura 3.3: Primi mockup dell'applicazione, sistema di allerta

L'immagine 3.3 mostra il riepilogo delle stazioni e l'eventuale iscrizione utente al sistema di allerta ozono per la data centralina. L'interazione utente qui presentata è a scopo puramente prototipale, e verrà studiata nel dettaglio in sede di progettazione dell'applicazione.

4 Progettazione del sistema server

L'applicazione richiede il recupero orario dei dati tramite internet, per poter visualizzare i livelli di ozono e particolato nell'aria. Per questo, è necessario progettare un sistema in grado di recuperare i dati dal database e di fornirli all'applicazione nel momento in cui, all'avvio, essa ne faccia richiesta.

4.1 Esigenze di trasmissione dei dati

Una richiesta molto specifica da parte di ARPAV è che non ci fossero interrogazione del database e generazione dei dati in seguito ad ogni richiesta remota. Un simile approccio infatti, combinato con le grandi dimensioni del set di dati da interrogare, sarebbe stato troppo costoso in termini di risorse.

Si è quindi deciso di far generare al sistema server i file contenenti i dati delle centraline e degli inquinanti rilevati in corrispondenza con l'aggiornamento orario dei valori nel database. Questo approccio garantisce il numero minimo di interrogazioni al database, e permette un buon controllo sull'output del server, che sarà uniforme per ogni istanza dell'applicazione.

Visto che l'applicazione verrà usata prevalentemente con connessioni ad internet a consumo, è stato necessario minimizzare le dimensioni dei file, studiando il formato dei dati e la loro struttura.

4.1.1 XML attualmente in uso

ARPAV utilizza, per l'esportazione veloce dei dati dal database, un applicativo chiamato Abilitatore XML che, data una query al database, è in grado di generare un file XML contenente il risultato della stessa.

4.1.1.0.1 Struttura

Il file XML generato dall'applicativo ricalca da vicino la struttura a record di un database relazionale, presentando una sequenza di righe del tipo:

```
<row>
  <VALORE>18.39672</VALORE>
  <STATCD>8</STATCD>
  <STATNM>PD Mandria</STATNM>
  <CODSEQ>500003731</CODSEQ>
  <ISTANZACD>1</ISTANZACD>
  <COORDCD>7</COORDCD>
  <PARAMETRO>O3</PARAMETRO>
  <DATAORA>2013-05-18 02:00:00</DATAORA>
  <GIORNO>SATURDAY </GIORNO>
  <DATAORA2>2013-05-18 02:00:00</DATAORA2>
  <GIORNO2>SAT</GIORNO2>
  <ORA>02:00</ORA>
  <DESCUM>mg/m3 293K</DESCUM>
</row>
```

Il documento è quindi valido rispetto ad uno schema che si ipotizza essere il seguente:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="data">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="row" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="VALORE" type="xsd:decimal" />
            <xsd:element name="STATCD" type="xsd:integer" />
            <xsd:element name="STATNM" type="xsd:string" />
            <xsd:element name="CODSEQ" type="xsd:string" />
            <xsd:element name="ISTANZACD" type="xsd:integer" />
            <xsd:element name="COORDCD" type="xsd:integer" />
            <xsd:element name="PARAMETRO" type="xsd:string" />
            <xsd:element name="DATAORA" type="xsd:string" />
            <xsd:element name="GIORNO" type="xsd:string" />
            <xsd:element name="DATAORA2" type="xsd:string" />
            <xsd:element name="GIORNO2" type="xsd:string" />
            <xsd:element name="ORA" type="xsd:string" />
            <xsd:element name="DESCUM" type="xsd:string" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

4.1.1.0.2 Criticità

Come emerge dall'analisi dell'XMLSchema fornito, il file presenta molte informazioni ridondanti, mancando di sfruttamento della gerarchia permessa da XML per ricalcare invece la struttura del database relazionale dal quale i dati sono esportati: sono presenti una serie di righe, ognuna delle quali contenente tutte le informazioni relative alla stazione che ha effettuato la misurazione, ed infine i dati relativi alla misurazione stessa.
In forma tabellare, questo si potrebbe rappresentare con una struttura simile alla seguente:

statcd	parametro	valore	dataora
8	O3	18.39	val1
8	O3	19.25	val2
8	O3	24.56	val3
8	O3	28.43	val4
8	O3	40.91	val5

e si consideri che la ripetizione di dati è maggiore di quanto rappresentato: ogni misurazione (formata da valore e *timestamp*) porterebbe con se più di dieci campi dati relativi alla stazione, considerando che alla struttura rappresentata andrebbero aggiunte informazioni sulle coordinate, sulla tipologia di stazione e sulla posizione nel territorio.

Una struttura migliore si potrebbe ottenere rappresentando invece la stazione con i suoi dati ed assegnandole come attributo di tipo complesso la misurazione di un dato inquinante, che a sua volta sarebbe specificata dalle effettive misurazioni. Ad esempio, supponendo di voler rappresentare cinque misurazioni orarie di ozono con la struttura descritta sopra (si suppongano rilevanti i campi dati denominati STATCD, PARAMETRO, VALORE e DATAORA ai fini dell'esempio), avremmo la ripetizione di cinque tag ROW, ognuno dei quali differente solo per *valore* e *dataora*.
Una struttura gerarchica ci permetterebbe invece di raggruppare le informazioni per stazione e per inquinante:

```
<STAZIONE>
  <STATCD>8</STATCD>
  <MISURAZIONI>
    <OZONO>
      <MISURA>
        <VALORE>18.39</VALORE>
        <DATAORA>val1</DATAORA>
      </MISURA>
      ...
    </OZONO>
  </MISURAZIONI>
</STAZIONE>
```

Questo ci permette di non dover ripetere il nome del parametro misurato e, soprattutto, i dati relativi alla stazione.

4.1.2 Progettazione di uno schema JSON

Per il nuovo file che risolve le criticità presentate si è deciso di adottare il formato JSON (*JavaScript Object Notation*).
Questa scelta si motiva con la più semplice lettura in JavaScript dei file risultati, e con le dimensioni mediamente leggermente minori dei file JSON rispetto ad equivalenti XML. La perdita di ricchezza semantica, ad esempio nei contenuti misti, e di capacità di manipolazione del documento non rappresentano un problema, vista l'esigenza di trasmissione di dati semplici e in sola lettura, dove invece la dimensione dei file è critica per garantire una buona esperienza utente (sia nella velocità che nelle dimensioni del download dei dati aggiornati).

Per maggior compattezza, si separeranno quindi i dati relativi alle stazioni e i dati relativi alle misure in due file distinti:

- **data.json** conterrà le misurazioni effettuate dalle stazioni;
- **stazioni.json** conterrà i dati relativi alle stazioni.

Entrambi i file *data.json* e *stazioni.json* riferiranno alle stazioni tramite il loro identificativo univoco, chiamato **CODSEQST** nella base di dati, che agirà quindi da chiave esterna. Tale denominazione, così come i nomi degli altri campi, verrà mantenuta per garantire una facile corrispondenza tra i file esportati e il database aziendale.

4.1.2.0.3 data.json

Il file conterrà un array di stazioni, ognuna qualificata dal proprio codice univoco (che funzionerà come chiave esterna verso il file contenente i dati delle stazioni) e da un array di misurazioni: questo conterrà un array contenente le misurazioni di ozono se presenti, ed un array contenente le misurazioni di PM10 se presenti.

Questi saranno degli array di oggetti contenenti come proprietà la data ed il valore della misura.

Formalmente, il file sarà valido secondo il seguente JSON Schema:

```
{
  "type":"object",
  "$schema": "http://json-schema.org/draft-04/schema",
  "properties":{
    "stazioni": {
      "type":"array",
      "items":
        {
          "type":"object",
          "properties":{
            "codseqst": {
              "type":"string",
            },
            "misurazioni": {
              "type":"array",
              "items":[
                {
                  "type":"object",
                  "properties":{
                    "ozono": {
                      "type":"array",
                      "items":
                        {
                          "type":"object",
                          "properties":{
                            "data": {
                              "type":"string",
                            },
                            "mis": {
                              "type":"string",
                            }
                          }
                        }


                    }
                  }
                },
                {
                  "type":"object",
                  "properties":{
                    "pm10": {
                      "type":"array",
                      "items":
```

```
                    {
                      "type":"object",
                      "properties":{
                        "data": {
                          "type":"string",
                        },
                        "mis": {
                          "type":"string",
                        }
                      }
                    }


                  }
                }
              }
            ]
          }
        }
      }


    }
  }
}
```

4.1.2.0.4 stazioni.json

Il file, molto più semplice, conterrà un array di stazioni, ognuna descritta con un oggetto con i dati essenziali come proprietà.

Il JSON Schema del file è il seguente:

```
{
  "type":"object",
  "$schema": "http://json-schema.org/draft-03/schema",
  "properties":{
    "stazioni": {
      "type":"array",
      "items":
        {
          "type":"object",
          "properties":{
            "codseqst": {
              "type":"string",
            },
            "comune": {
              "type":"string",
            },
```

```
            "lat": {
              "type":"string",
            },
            "localita": {
              "type":"string",
            },
            "lon": {
              "type":"string",
            },
            "nome": {
              "type":"string",
            },
            "provincia": {
              "type":"string",
            },
            "tipozona": {
              "type":"string",
            }
          }
        }


    }
  }
}
```

4.2 Strumenti e tecnologie

Per la realizzazione del sistema server si sono utilizzate le seguenti tecnologie:

- **PHP: Hypertext Processor**: PHP è un linguaggio di programmazione (o di scripting, come lo definisce l'homepage del progetto) interpretato, molto utilizzato per la realizzazione di applicazioni web; dalla release 5 ha migliorato il supporto agli oggetti introducendo tutte le caratteristiche comuni della programmazione OOP, comprese le interfacce. Viene qui utilizzato per la realizzazione del server in un normale stack XAMPP (Apache - MySql - Perl - PHP).

- **CodeIgniter**: un framework opensource realizzato in PHP mantenuto da EllisLab e dalla comunità. È pensato per essere leggero, molto veloce ed aderire al paradigma MVC (Model-View-Controller), oltre a fornire un buon supporto ai database grazie a classi *adapter* capaci di supportare tutti i principali sistemi.
  CodeIgniter inoltre fornisce una libreria di routing, capace di associare i metodi delle classi controller agli URL, permettendo così di richiamare funzioni dell'applicazione tramite un semplice wget: ad esempio il metodo *airdata()* definito in Aria_Controller verrà invocato tramite l'URL server-path/aria/airdata. Questo permetterà di aggiornare i dati dell'applicazione

tramite cronjob che, ad intervalli orari definiti, visiti tramite istruzione *wget* gli URL appropriati.

4.3 Progettazione del sistema

Per progettare il sistema, tenuto conto delle esigenze applicative, si è deciso di utlizzare il pattern architetturale MVC (Model-View-Controller), realizzando poi dei modelli costruiti a partire dal database, al quale si accede tramite pattern DAO (Data Access Object).

Si descrivono brevemente i pattern usati:

- **MVC**: Il pattern *Model-View-Controller* è un paradigma architetturale che prevede il disaccoppiamento del sistema in tre componenti: il Model (destinato ai dati e alle loro regole d'accesso), View (La loro rappresentazione grafica; nel nostro caso la stampa su file) e Controller (che gestisce le richieste utente). Questo permette buona modularità dell'applicazione, grazie all'indipendenza delle componenti, e favorisce estendibilità e riuso di tutto o parte del sistema.

- **DAO**: il pattern DAO, *Data Access Object*, favorisce l'indipendenza del modello di dati di un'applicazione dalla sua rappresentazione tabellare in una base di dati, oltre a garantire un'interfaccia uniforme per l'accesso al database indipendentemente dalla tecnologia di persistenza usata, grazie alla definizione di oggetti che fungano da tramite tra il modello di dati e lo strato di persistenza degli stessi. In particolare, si sfrutterà il DAO già fornito da CodeIgniter.

- **Strategy**: lo *Strategy pattern* permette ad una famiglia di algoritmi, rappresentati come classi, di esporre un'interfaccia comune, permettendo quindi all'algoritmo di essere selezionato a runtime senza che il chiamante debba fare asserzioni sull'algoritmo istanziato. Nell'applicazione si utilizzerà il pattern per creare un'interfaccia *Writer*, estesa dagli algoritmi concreti di scrittura dei dati forniti in un dato formato. In questo modo, tramite una medesima interfaccia, è possibile esportare dati in JSON oppure in XML tramite la differente istanziazione di una classe, senza dover modificare il codice.

Vista la semplicità dei modelli che richiedono la sola lettura delle informazioni, si è deciso di non estendere i modelli di CodeIgniter, che utilizzano il pattern Active-Record, ma di creare un proprio modello base da estendere, mantenendo i corretti riferimenti agli oggetti del framework, che viene così utilizzato principalmente per i controller, il routing ed il caricamento delle viste.

4.4 Architettura e specifica delle classi

In figura 4.1 si mostra l'architettura ad alto livello del sistema server. Di seguito, si motivano le principali scelte progettuali, prima di illustrare ruolo e funzione di ogni classe individuata.

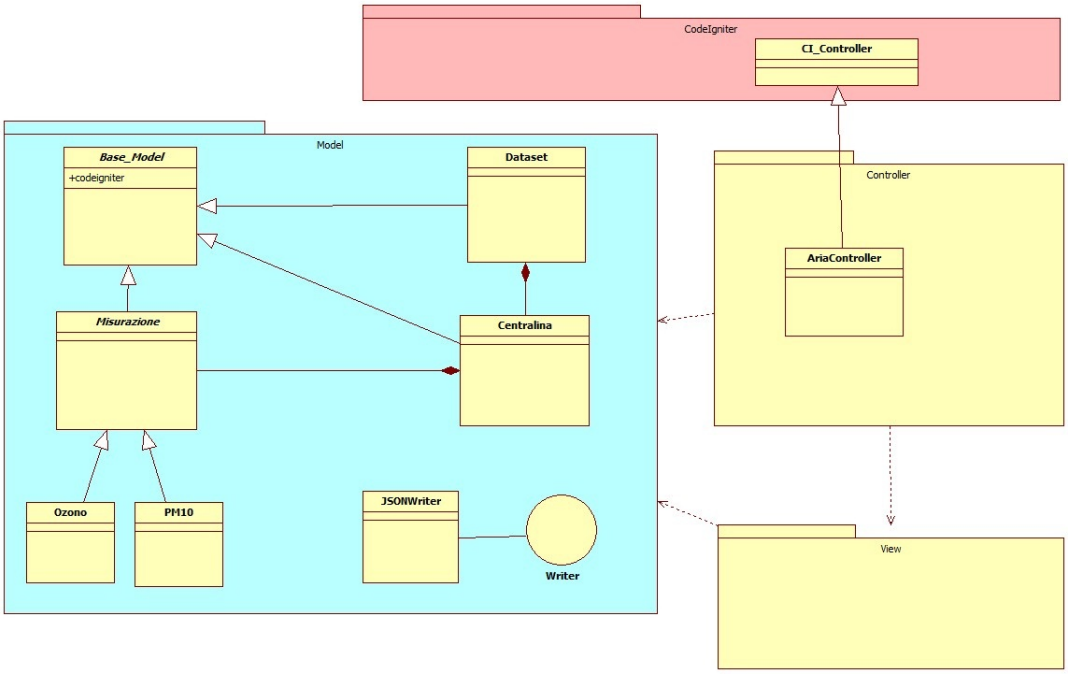

Figura 4.1: Architettura ad alto livello del server

Si vuole creare un sistema server che sia estendibile, ovvero che non si limiti ad esportare solo i dati di qualità dell'aria in JSON, ma che sia in grado di esportare le misurazioni di ogni inquinante rilevato da ARPAV, il tutto in diversi formati. È quindi necessario progettare un sistema modulare, facilmente estendibile ed in grado di prelevare i dati dalle varie tabelle del database.

In figura 4.2 si mostra, senza rispettare il formalismo UML, la struttura ad alto livello della parte di database ARPAV che si occupa di rappresentare i dati principali della stazione (tabella TESTAT), tutti gli inquinanti che può rilevare (tabella TEPNT) e, per ogni possibile inquinante, le misure rilevate, in tabella TDDAY, TDHOUR o TDMINUTE a seconda della frequenza giornaliera, oraria o al minuto di campionamento dello stesso.

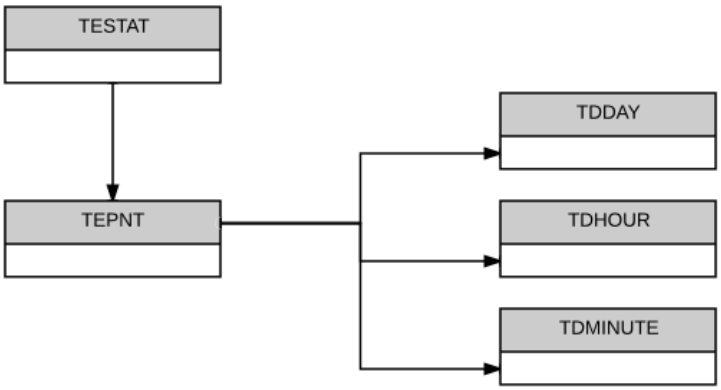

Figura 4.2: Schematizzazione di parte del database Oracle utilizzato

È quindi necessario realizzare una classe astratta per le misure, chiamata *Misurazione*, che possa essere estesa dalle misurazioni concrete. Queste ultime dovranno semplicemente specificare il codice dell'inquinante desiderato e la sua frequenza di campionamento come campi dati, sfruttando i metodi e le logiche di accesso alle tabelle TDMINUTE, TDHOUR e TDDAY specificate ed ereditate da Misurazione. Nel caso la loro logica di accesso ai dati sia differente (ad esempio, misurazioni su tabelle diverse) sarà sufficiente effettuare l'overriding dell'opportuno metodo. Questo è positivo, perché permette di estendere il sistema per creare nuove classi rappresentanti misurazioni in modo molto semplice, con la semplice specifica di due parametri, mantenendo al contempo un'interfaccia comune per le misurazioni.

Per permettere l'esportazione dei dati in modo estendibile si è deciso di creare un'interfaccia *Writer*, che rappresenta un generico scrittore di dati del modello. Le classi che vogliono aderire a tale contratto dovranno specificare i metodi per scrivere collezioni, elementi, attributi e campi dati; i controller che utilizzeranno i metodi dell'interfaccia Writer per stampare dati potranno cambiare output semplicemente istanziando una diversa classe concreta. In questo modo, l'aggiunta di nuovi formati di stampa ed esportazione dei dati è resa semplice e modulare.

La componente View è realizzata, come da convenzione del framework CodeIgniter, attraverso file PHP simili a template ai quali il controller possa passare determinati dati ed oggetti. La componente View del sistema risulta quindi sprovvista di vere e proprie classi, ma consiste in diversi file caricati dal controller, dei quali si omette rappresentazione in figura 4.1.

4.4.1 Componente Model

Model::Base_Model

Descrizione e utilizzo Rappresenta la classe astratta base, che ogni modello definito deve estendere. All'istanziazione, salva in un campo dati protetto

un riferimento all'istanza di CodeIgniter, che permette l'accesso al database, ai file di configurazione ed eventualmente agli altri modelli.

Tipologia Classe astratta

Model::Dataset_Model

Descrizione e utilizzo Costruisce e rappresenta insiemi di dati, creati costruendo ed aggregando altri modelli. Ha la responsabilità di fornire ai controller un punto di accesso agli insiemi di dati, nascondendone le implementazioni.

Classi ereditate
- Model::Base_Model

Model::Centralina_Model

Descrizione e utilizzo Rappresenta una stazione di rilevamento ed i suoi dati. Viene istanziata a partire dall'identificativo univoco di una stazione nel database, e si occupa di popolare i campi dati rappresentanti i dati della stazione. Successivamente, costruisce degli insiemi di misurazioni per gli inquinanti indicati, a partire dalle date indicate.

Classi ereditate
- Model::Base_Model

Model::Misurazione_Model

Descrizione e utilizzo Classe astratta per rappresentare un rilevamento di un inquinante da parte di una centralina ad un dato momento. Definisce query generiche per la costruzione di oggetti misurazione concreti, gestendo l'eventuale invalidità del dato recuperato. Sottoclassi concrete che ereditano da questa possono essere costruite in modo semplice, specificando semplicemente frequenza di campionamento, codice dell'inquinante e della stazione e orario desiderato, delegando alla classe corrente il compito di accedere al database tramite DAO e rappresentare il risultato. Una chiamata al metodo ereditato *View()* della sottoclasse restituirà l'oggetto popolato con i dati recuperati.

Tipologia Classe astratta

Classi ereditate
- Model::Base_Model

Model::Ozono_Model

Descrizione e utilizzo Rappresenta un rilevamento di ozono da parte di una stazione ad un dato tempo. All'istanziazione costruisce la superclasse Misurazione_Model con i giusti parametri (Codice dell'ozono, recuperato da file di configurazione, codice della stazione d'interesse e orario). Eredita il metodo View() della superclasse, che si occupa del recupero da database dei dati e del popolamento dei campi dati.

Classi ereditate
- Model::Misurazione_Model

Model::PM10_Model

Descrizione e utilizzo Rappresenta un rilevamento di PM10 da parte di una stazione ad un dato tempo. All'istanziazione costruisce la superclasse Misurazione_Model con i giusti parametri (Codice del PM10, recuperato da file di configurazione, codice della stazione d'interesse e orario). Eredita il metodo View() della superclasse, che si occupa del recupero da database dei dati e del popolamento dei campi dati.

Classi ereditate
- Model::Misurazione_Model

Model::Writer_Helper

Descrizione e utilizzo Interfaccia definita come classe *Helper* del framework CodeIgniter, rappresenta un generico scrittore di dati su stream, fornendo appropriati metodi generici per la stampa di collezioni e di elementi e attributi degli stessi, rimanendo però agnostica relativamente alla struttura dei dati. Classi che aderiscono a tale contratto potranno essere utilizzate intercambiabilmente dai Controller e dalle viste per scrivere oggetti e loro proprietà su stream definibili. Al momento è presente solo uno scrittore JSON, ma uno scrittore XML (o testo semplice, o HTML, ad esempio), vista la popolarità del formato per l'interscambio di dati presso ARPAV, può essere facilmente realizzato ed utilizzato implementando questa interfaccia.

Tipologia Interfaccia

Model::JSON_Writer_Helper

Descrizione e utilizzo Helper che permette la stampa di collezioni di dati e di dati in formato JSON, implementando i metodi definiti dall'interfaccia Writer_Helper.

Interfacce implementate
- Model::Writer_Helper

4.4.2 Componente Controller

Controller::Aria_Controller

Descrizione e utilizzo Si occupa dell'istanziazione ed esportazione dei dati di qualità dell'aria: istanziazione tramite le classi del Model, in particolare la classe Dataset; esportazione tramite file di vista che, dati i dati ed il Writer concreto passati dal controller, scrivano su disco i file JSON necessari.

Classi ereditate
- CodeIgniter::CI_Controller

4.4.3 Componente View

View::Aria::Data_File.php

Descrizione e utilizzo File di vista dell'applicazione. Attende dal controller i dati di qualità dell'aria ed un'istanza di una classe che implementi l'interfaccia Writer, e scrive su disco il file data.JSON.

View::Aria::Stations_File.php

Descrizione e utilizzo File di vista dell'applicazione. Attende dal controller i dati relativi alle stazioni di qualità dell'aria ed un'istanza di una classe che implementi l'interfaccia Writer, e scrive su disco il file stations.JSON.

4.5 Diagrammi di sequenza

Per illustrare meglio l'architettura del server, si mostra in figura 4.3 il diagramma di flusso delle operazioni causate dalla visita dell'indirizzo *server-path/aria/airdata*, concentrandosi sulle componenti dell'architettura interne al sistema. Verranno quindi omesse le operazioni effettuate dal framework, il cui funzionamento sarà quindi trasparente (si suppone ad esempio che la visita all'URL chiami direttamente il controller, ignorando l'instanziazione del framework e l'effettiva chiamata del controller causata dal router).

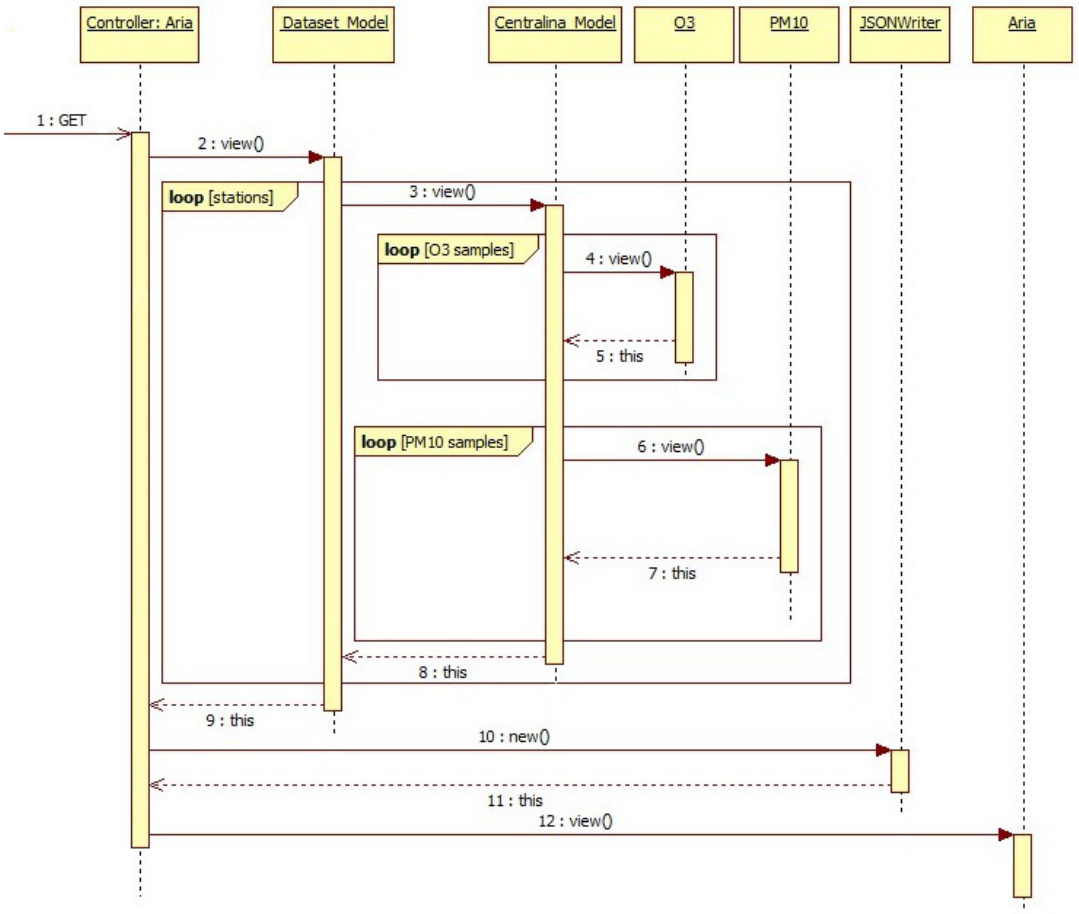

Figura 4.3: Diagramma di sequenza, esportazione dei dati da parte del server

5 Implementazione e testing del sistema server

Successivamente alla progettazione del sistema server, si è proceduto alla sua implementazione, realizzando l'architettura software ed i metodi previsti. Durante lo sviluppo, si è adottata una rigida strategia di test di unità per garantire la conformità alle attese del prodotto.

5.1 Il sistema di coordinate

Il sistema di coordinate utilizzato per salvare la posizione delle stazioni di rilevamento nel database si basa sulla proiezione di Gauss-Boaga, molto utilizzata nella cartografia ufficiale italiana. L'esigenza, però, è di esportare le coordinate in WGS84 (World Geodetic System 1984), comunemente utilizzato dai GPS e dalle mappe online.
Questo era imprevisto, perché non documentato nel database, ed è stato scoperto tramite test di unità e successivi approfondimenti. L'esigenza ideale sarebbe quella di convertire dinamicamente le coordinate tra i due sistemi di riferimento, in modo da poter prelevare ed esportare i dati richiesti in ogni momento con le coordinate WGS84.
Sfortunatamente, la precisa trasformazione di coordinate da un sistema all'altro richiede algoritmi specifici e dei file, denominati *grigliati*, non disponibili ad ARPAV, rendendo così impossibile la trasformazione su server. Metodi approssimati basati su semplici traslazioni, a seguito di prove, restituivano risultati errati anche di 500 metri, precisione giudicata insufficiente.
È obbligo di legge per il Geoportale Nazionale, un servizio ministeriale, fornire un convertitore online di coordinate; questo però non forniva API o altri sistemi per poter essere interrogato dinamicamente dal nostro sistema server.

Si è quindi deciso di mantenere un nuovo file, denominato *coordinate.json*, dove si associa l'identificativo della stazione alle sue coordinate, convertite manualmente. Il file andrà aggiornato a mano in caso di accensione di nuove stazioni di qualità dell'aria.

5.2 Il problema di DateTime::sub()

Altro intoppo allo sviluppo rilevato tramite test di unità è stato il manifestarsi di un bug presente nella classe DateTime di PHP, risolto a partire dalla versione 5.3.3, che portava gli oggetti data sui quali fosse stata chiamata la sottrazione di un intervallo temporale (funzione di cui si fa frequente uso nel codice, per recuperare i rilevamenti di inquinanti precedenti) a ripetere la sottrazione ad ogni ulteriore operazione effettuata. Vista l'impossibilità di aggiornamento della versione di PHP sul server utilizzato, questo ha portato al salvataggio di oggetti data come timestamp, quindi numeri interi, piuttosto che come oggetti DateTime già istanziati.

5.3 Testing

Lo sviluppo di ogni componente del sistema server è stato preceduto dalla scrittura di adeguati test di unità, perché comportamento dei metodi e output fossero sempre quelli attesi. Si riportano di seguito gli strumenti utilizzati e si consuntivano brevemente i test effettuati.

Per i test di unità si è deciso di utilizzare la classe Unit_Test fornita dal framework CodeIgniter, perché già presente ma al tempo stesso capace di eseguire i test necessari producendo adeguati report. La classe non ha la completezza di altre suite di test, trattandosi soprattutto di un sistema capace di controllare asserzioni semplici e tipi di dato e stampare i risultati ed eventuali discrepanze, ma è stata giudicata sufficiente allo scopo.

I test di unità del sistema server consistono principalmente della verifica della buona lettura dal database e del corretto popolamento delle classi, richiedendo ai moduli software di caricare dei dati noti in partenza, e confrontando i campi dati popolati con i dati conosciuti.

Vista l'implementazione di tutte le funzionalità necessarie del sistema, tutti i test di unità sono stati implementati e risultano superati.

6 Progettazione dell'applicazione

Nella progettazione dell'applicazione, ad un primo periodo di scelta di tecnologie e strumenti per lo sviluppo e per la verifica, oltre che di opportune metriche software, sono seguite attività di progettazione dell'architettura e di design dell'interfaccia e dello schema di colori.
Quanto segue illustra le attività svolte.

6.1 Strategia di verifica

Perché il processo di verifica, tramite analisi statica e test, sia utile, è necessario renderlo quantificabile tramite la definizione di opportune metriche. Si definiscono di seguito le metriche scelte, assieme ai loro range di accettazione ed ottimali.

Indice di manutenibilità

Questa metrica definisce un indice di manutenibilità di una classe come un numero compreso tra 0 (codice poco manutenibile) e 100 (codice ben manutenibile), calcolandolo a partire dalla metrica denominata Volume del Software da Halstead, dalla complessità ciclomatica e dal numero di linee di codice con la formula seguente:

$MAX(0, (171 - 5.2 * ln(HalsteadVolume) - 0.23 * (CyclomaticComplexity) - 16.2 * ln(LinesofCode)) * 100/171)$

Parametri utilizzati:

- Range-accettazione: [20-100];
- Range-ottimale: [50-100].

Numero di errori stimato

La metrica *Delivered Bugs* è una delle misure di complessità di Halstead, calcolata a partire dalla metrica chiamata Effort, che mette in relazione Difficulty (parametro calcolato dal numero di operatori ed operandi) e Volume. Vuole essere indice della difficoltà, e quindi della probabilità di insorgenza di errori nel

codice.
Parametri utilizzati:

- Range-accettazione: [0-5];
- Range-ottimale: [0-3].

Complessità ciclomatica

La metrica di complessità ciclomatica indica il numero di cammini linearmente indipendenti possibili durante l'esecuzione di un particolare metodo. Malgrado questa metrica sia già inclusa all'interno dell'Indice di Manutenibilità, la si vuole analizzare anche separatamente, vista l'importanza della stessa nel calcolo dei cammini (e quindi della copertura del codice).
Parametri utilizzati:

- Range-accettazione: [0-10];
- Range-ottimale: [0-6].

Si noti che parametri quali il massimo numero di metodi per classe e massimo numero di argomenti per metodo, indici di qualità della progettazione, risultano mancanti dalla precedente lista di metriche perché adottati già durante la progettazione, e quindi sicuramente rispettati durante la stesura del codice. Si riportano ugualmente le metriche ed i relativi parametri:

Massimo numero di metodi per classe

Il massimo numero di metodi per classe è una metrica che, se superata, può essere sintomo di una classe che riunisce le responsabilità di più classi e che quindi potrebbe essere utilmente divisa. Un eventuale superamento di questa metrica può essere tollerato se adeguatamente giustificato.
Parametri utilizzati:

- Range-accettazione: [0-8];
- Range-ottimale: [0-4].

Massimo numero di argomenti per metodo

Il massimo numero di argomento per metodo è una metrica che, se superata, può essere indice della necessità di dividere il metodo in più metodi, o che le strutture dati usate potrebbero essere astratte maggiormente. Un eventuale superamento di questa metrica può essere tollerato se adeguatamente giustificato.
Parametri utilizzati:

- Range-accettazione: [0-6];
- Range-ottimale: [0-4].

Accessibilità

Si vuole garantire l'accessibilità dell'applicazione e dei dati presentati dalla stessa.
Per fare questo, si sfruttano le linee guida WCAG (Web Content Accessibility Guidelines) 2.0, che misurano l'accessibilità delle pagine web con un indice misurato da zero a tre "A", a seconda dell'aderenza del prodotto con le linee guida. Il W3C provvede con chiare spiegazioni di ogni linea guida e con checklist per misurare il livello di aderenza alle stesse del prodotto. Si vuole raggiungere un minimo di "AA" per tutti i punti della lista di controllo applicabili all'applicazione ARPAV Aria.
Parametri utilizzati:

- Range-accettazione: [AA];
- Range-ottimale: [AAA].

Le linee guida WCAG sono state inoltre integrate con le linee guida fornite da Google per lo sviluppo di applicazioni Android che, sebbene molto generali, possono dare indicazioni utili riguardanti l'interfaccia ed il suo utilizzo.

6.2 Strumenti e tecnologie

6.2.1 Strumenti per lo sviluppo

- **Javascript e Node.js**: linguaggio di scripting interpretato utilizzato soprattutto lato client, supporta i paradigmi di programmazione ad oggetti e funzionale. Grazie anche alle tecniche AJAX (*Asynchronous JavaScript and XML*), che permettono invio e recupero di informazioni senza dover ricaricare la pagina web è possibile utilizzarlo per la costruzione di applicazioni client-side. Node.js è un framework per l'esecuzione locale di codice JavaScript, utilizzato da moltissimi strumenti, tra cui gli strumenti scelti per la verifica del codice prodotto e per l'esecuzione di task automatici. La popolarità di JavaScript fa si che siano disponibili moltissime librerie e framework; si elencano di seguito le librerie utilizzate:
  - **Backbone.js**: libreria JavaScript basata sui paradigmi MVP (*Model-View-Presenter*) ed MVC, che incoraggia un approccio strutturato alla stesura del codice, permettendo di definire modelli, collezioni e viste ed occupandosi della gestione del routing e degli eventi. Fornisce sistemi per il *rendering* di template HTML a partire da modelli, e si occupa dell'aggiornamento automatico delle viste al cambiare delle proprietà dei modelli. Inoltre, permette una sorta di ereditarietà, definita estensione, tra prototipi alternativa all'ereditarietà prototipale prevista da JavaScript, ed è flessibile, permettendo al programmatore di interpretare il pattern architetturale MV* come preferisce: considerando i template HTML come viste e le classi View come dei controller

specializzati, oppure creando una classe controller responsabile dell'istanziazione delle diverse classi View, relegando queste ultime al solo scopo di presentazione.

  - **Backbone.Marionette**: estensione di Backbone.js, pensata per raccogliere le *best practices* di utilizzo del framework in soluzioni pronte all'uso, che riducano la necessità di copia-incolla e di *boilerplate code*, ad esempio semplificando il rendering delle viste;
  - **jQuery**: libreria JavaScript pensata per semplificare l'interazione con il DOM (Document Object Model, utilizzato per rappresentare le entità presenti nel documento) e le tecniche AJAX;
  - **Modernizr**: libreria JavaScript che permette di eseguire test al caricamento della pagina per verificare l'eventuale supporto del browser a date tecnologie, ad esempio SVG. Permette di specificare quali tecnologie si desidera testare, e ne riporta il nome come classe dell'elemento <html> in caso di esito positivo;
  - **Leaflet.js**: libreria per la creazione di una mappa interattiva e compatibile con dispositivi touchscreen (grazie al supporto delle gesture) basata sui dati di OpenStreetMap, scelto come provider delle mappe, e sui *tiles* (il rendering effettivo della mappa a partire dai dati) forniti da vari provider;
  - **Highcharts e Charts.js**: librerie per la creazione di grafici a partire da serie di dati. Highcharts si basa su elementi SVG (Scalable Vector Graphics), mentre Charts.js disegna all'interno di un elemento <canvas> di HTML5;

- **Bootstrap**: framework per lo sviluppo web realizzato da Twitter, fornisce template HTML e CSS, oltre che componenti JavaScript opzionali, per la creazione di interfacce. Permette la creazione di design cosiddetti responsivi, che si adattano alle dimensioni della pagina e del dispositivo che la visualizza;

- **Apache Cordova**: framework per lo sviluppo mobile che permette la creazione di applicazioni utilizzando le tecnologie web (HTML, CSS e JavaScript). Permette l'accesso alle funzionalità del dispositivo da parte dell'applicazione, la quale ci accederà tramite le API standard definite dal W3C nella specifica HTML5, e si occupa della distribuzione ed installazione dell'applicazione nei dispositivi. L'applicazione funzionerà poi su una WebView, sorta di browser distribuito con essa, che assicura consistenza di accesso all'hardware indipendentemente dal sistema operativo sottostante;

- **W3C API**: Il W3C definisce delle API standard che il browser può fornire per svolgere vari compiti. Nel progetto, sarà Cordova a fornirne il supporto, e ad occuparsi quindi di garantire il funzionamento di quelle che richiedono accesso all'hardware. Di seguito si elencano le API utilizzate:

  - **Geolocation API**: set di oggetti JavaScript che, dopo autorizzazione utente, cercano di geolocalizzarlo tramite vari sistemi, ad esempio segnale WiFi, segnale telefonico e satelliti GPS. Questa operazione viene svolta in maniera trasparente all'utente dell'interfaccia, che dispone di un metodo esposto per la richiesta di geolocalizzazione, metodo che ritorna un oggetto con i dati oppure un errore in caso di fallimento;
  - **History API**: permettono la manipolazione della cronologia del browser tramite script, in modo da fornire un URL a date risorse anche se il cambiamento della pagina web non è stato provocato da un refresh della stessa o dalla navigazione verso un nuovo URL, ma dalla modifica di parte del documento con tecniche AJAX. Questo è molto utilizzato dalle applicazioni web a pagina singola, che riescono così a dare all'utente l'illusione della navigazione tra più pagine, fornendo al contempo URL per il salvataggio nei preferiti. Questi URL non rappresentano vere pagine, ma determinati stati dell'applicazione;
  - **WebStorage**: insieme di metodi e protocolli per il salvataggio di informazioni nel browser lato client in modo persistente oppure per la durata della sessione in corso. Si comportano come cookies di sessione o persistenti, con il vantaggio della standardizzazione, e permettono il salvataggio presso il client di coppie chiave-valore. Nel progetto verrà utilizzato LocalStorage, che garantisce la persistenza delle informazioni.

6.2.2 Strumenti per la verifica

- Verifica di processi
  - **Tracy v2**: software interno per la gestione di progetto, permette di definire e tracciare requisiti, casi d'uso, test (di sistema, di validazione, di integrazione e di unità), e l'architettura a vari livelli (dai package fino ai singoli metodi). Provvede anche ad una conveniente esportazione di documentazione formattata e consente di verificare il buon tracciamento di ogni componente del sistema ed ogni requisito, grazie ad un cruscotto preposto;
- Codice prodotto
  - **JSHint**: strumento per la rilevazione di errori comuni del codice Javascript e la misurazione di alcune metriche;
  - **Plato**: strumento per l'analisi statica del codice Javascript. Calcola le metriche di complessità, manutenibilità e complessità ciclomatica del codice definite in 6.1, e permette di analizzare report chiari ed informativi a vari livelli di dettaglio, dall'applicazione intera (come mostrato in figura 6.1) fino alle classi e metodi, direttamente nel codice sorgente (come mostrato in figura 6.2);
  - **Jasmine**: strumento per l'esecuzione automatica di test di unità ed integrazione del codice Javascript;

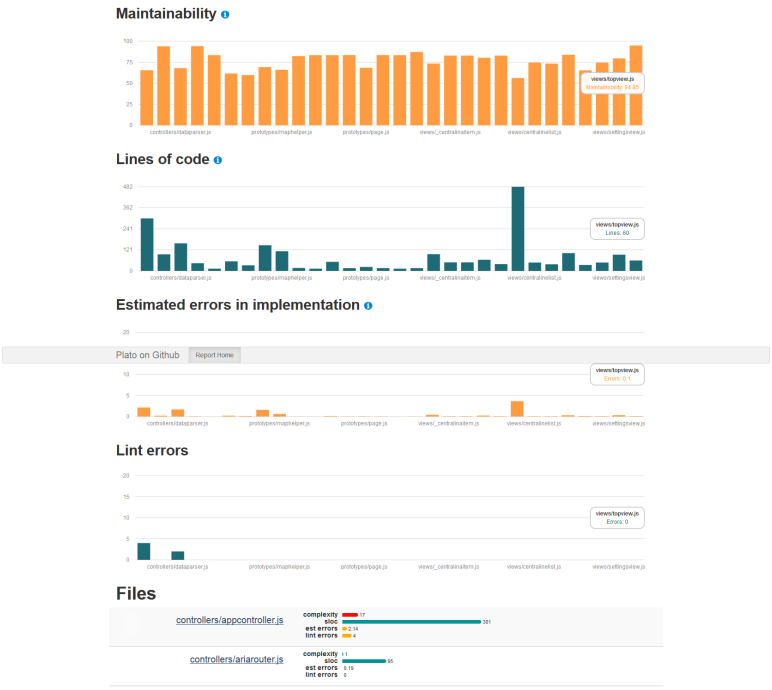

Figura 6.1: Lo strumento Plato, report di applicazione

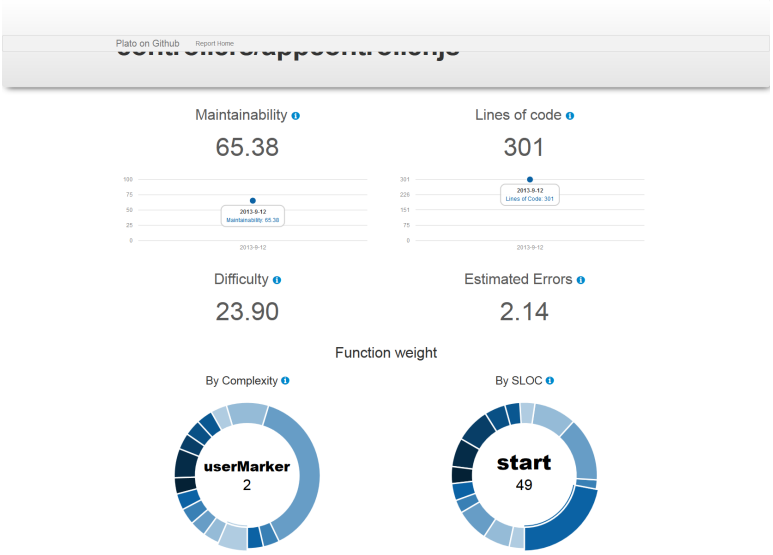

Figura 6.2: Lo strumento Plato, report riguardante una singola classe

  - **Istanbul**: strumento per la misura di line e branch coverage del codice Javascript. Può essere collegato a Jasmine per la misura della code coverage durante l'esecuzione dei test;
  - **YUIDoc**: un generatore di documentazione a partire dal codice sorgente e suoi commenti, utilizza una sintassi simile a Javadoc e produce una documentazione HTML navigabile e ben leggibile;
  - **Grunt**: Grunt è un *task runner*, un esecutore automatico di azioni, in maniera simile ad un *Makefile*, ma pensato per JavaScript ed estendibile grazie ad un sistema di plugin;
  - **Firebug**: strumento di sviluppo avanzato per il browser Firefox, fornisce una console JavaScript e permette il *debugging* del codice prodotto tramite breakpoint nel codice e monitoraggio del DOM e degli elementi JavaScript. Possiede anche strumenti atti a misurare le richieste di rete effettuate dall'applicazione, e strumenti per la modifica *live* di HTML e CSS;
  - **PhantomJS**: strumento basato su WebKit capace di automatizzare numerose operazioni su pagine web, simulando l'interazione di un utente, e di eseguire i test di unità definiti con altri framework, ad esempio Jasmine;
  - **Apache Ripple**: utilizzato come estensione per i browser Chrome e Chromium, permette di simulare un ambiente mobile grazie all'emulazione degli eventi lanciati dai dispositivi, delle dimensioni della schermata, degli eventi di movimento, rotazione, connettività internet e geolocalizzazione;

- HTML, CSS e accessibilità

  - **Validatori W3C**: il W3 Consortium fornisce dei validatori automatici per HTML5 e CSS, capaci di rilevare l'aderenza agli standard del prodotto;
  - **Total Validator**: strumento in grado di validare HTML e CSS in modo molto preciso. Fornisce anche indicazioni circa l'accessibilità secondo le linee guida definite dallo standard WCAG 2.0, effettuando i più comuni ed automatizzabili controlli;
  - **WCAG Contrast Checker**: strumento aggiuntivo per il browser Firefox per la misurazione dei contrasti di colore secondo le linee guida WCAG, sia in caso di visione normale che di discromatopsia;
  - **Fangs**: strumento aggiuntivo per il browser Firefox per la simulazione di screenreader;
  - **Android Talkback**: strumento di accessibilità del sistema operativo Android capace di leggere lo schermo dello smartphone.

6.2.3 Automatizzazione del processo di verifica

Grazie al *task runner* Grunt, è possibile creare una lista di task da eseguire quando desiderato. Grazie alla sua popolarità, possiede plugin per tutti gli strumenti di verifica utilizzati: JSHint, Plato, Jasmine, YUIDoc. Ci permette inoltre di concatenare i file contenenti le dichiarazioni dei prototipi e di minificare i file risultanti.
Un'esecuzione del *Gruntfile* creato permette quindi di eseguire tutti i test di analisi statica e i test di unità specificati, ottenendo report dettagliati ed errori specifici segnalati da JSHint, oltre che di avere una documentazione sempre aggiornata. Vista la semplicità di invocazione, è adatto per l'inserimento in un apposito *hook before-commit* di Git, il sistema di versioning utilizzato, permettendoci il controllo del codice sorgente prima che questo possa essere inserito nel repository.

6.3 Interfaccia utente

Particolare attenzione è stata posta nello studio dell'interfaccia, volendo garantire un'ottima esperienza utente indipendentemente dalle dimensioni e proporzioni dello schermo utilizzato, fosse esso uno schermo QVGA (320x240 pixel) oppure il monitor di un normale computer. L'interfaccia, inoltre, sebbene non fosse richiesto il *look&feel* nativo di ogni sistema operativo mobile, è stata oggetto di studio perché risultasse quanto più possibile intuitiva secondo le diverse abitudini utente.

6.3.1 I layout responsivi

Per la realizzazione di un'interfaccia capace di scalare al meglio dal grande schermo *Retina* di un iPad ad uno schermo QVGA di un dispositivo di fascia bassa, si è sfruttato l'approccio di Bootstrap denominato *Web design responsivo*.

I layout responsivi si adattano allo schermo dell'utente utilizzando una griglia fluida, basata sulle percentuali e sulle proporzioni dello schermo piuttosto che su pixel fissati. Bootstrap in particolare divide la pagina in una griglia di dodici colonne, la cui dimensione è fluida ed espressa in percentuale e varia quindi con le dimensioni del browser, e fornisce adeguate classi CSS per permettere ad elementi di blocco di occupare un certo numero di colonne della griglia. Grazie alle *media query*, le regole possono essere dinamicamente modificate, permettendo ad esempio di impilare gli elementi presenti sulla stessa riga piuttosto che affiancarli quando le dimensioni dello schermo risultino essere inferiori ad una certa soglia.

Questa strategia, che combina le *media query* ad un più facile utilizzo dei principi del design fluido, grazie alla griglia, permette la creazione di un'interfaccia utente capace di adattarsi ad ogni schermo, evitando asserzioni sulle grandezze dei dispositivi o sui dispositivi sottostanti.

6.3.2 Prototipi di interfaccia

La progettazione del layout generale deve decidere la posizione e le modalità di accesso ai diversi elementi previsti:

- Un elemento, prevedibilmente l'header, che contenga il titolo della schermata visualizzata dall'utente;
- I controlli di navigazione:
  - Eventuali menu di accesso alle funzionalità previste;
  - il tasto Indietro (denominato backbutton) per i dispositivi iOS che sono sprovvisti di un backbutton fisico o su schermo. Questo dovrà preferibilmente essere posto in alto a sinistra, per incontrare le aspettative degli utenti iOS (e le linee guida sullo sviluppo di interfacce definite da Apple)
  - Eventuali controlli contestuali alla pagina visualizzata;
- Il contenuto principale della pagina.

Tenuto conto di questi elementi, si sono elaborate le proposte riportate e descritte di seguito.

6.3.2.1 Layout ispirato ad Android

Una prima possibilità è ispirata allo stile distintivo di Android, ed è schematizzata in figura 6.3: presenta un header distintivo diviso in tre parti, la centrale dedicata al titolo della schermata e le due laterali rispettivamente per il *backbutton* e per un eventuale pulsante di azione a seconda del contesto (ad esempio, per la geolocalizzazione quando si visualizza la mappa), l'area centrale per i contenuti ed un footer contenente i controlli di navigazione, come se le diverse schermate fossero delle schede dell'applicazione.

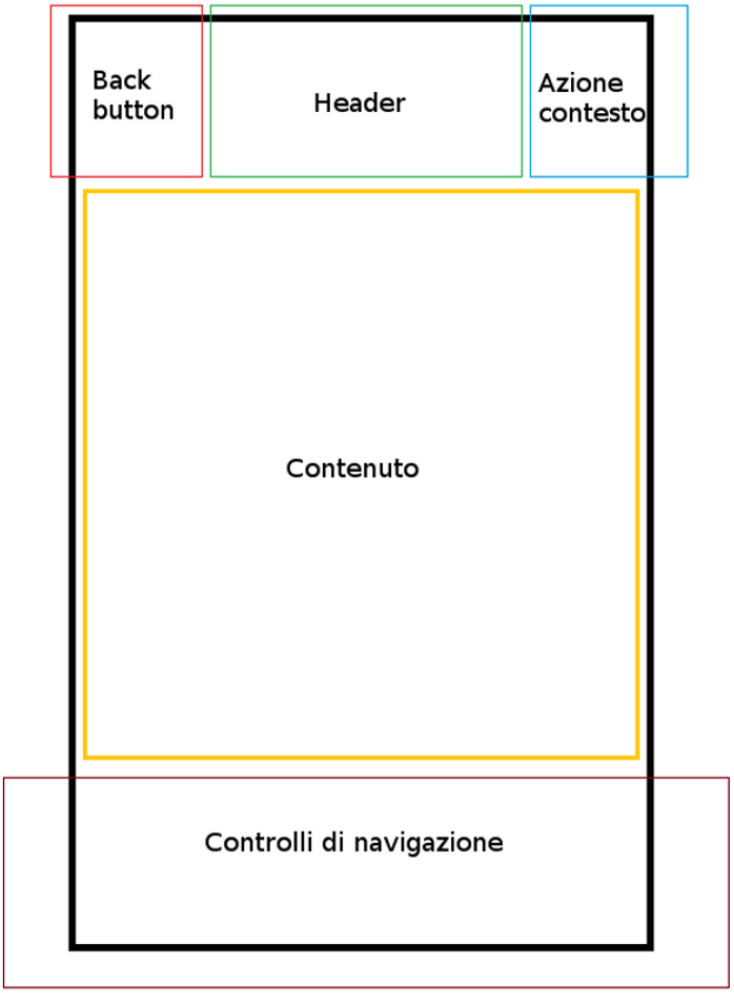

Figura 6.3: Layout ispirato ad Android

Quanto illustrato si può comprendere meglio osservando dei mockup di interfaccia che utilizzi questo layout, in figura 6.4 e 6.5.

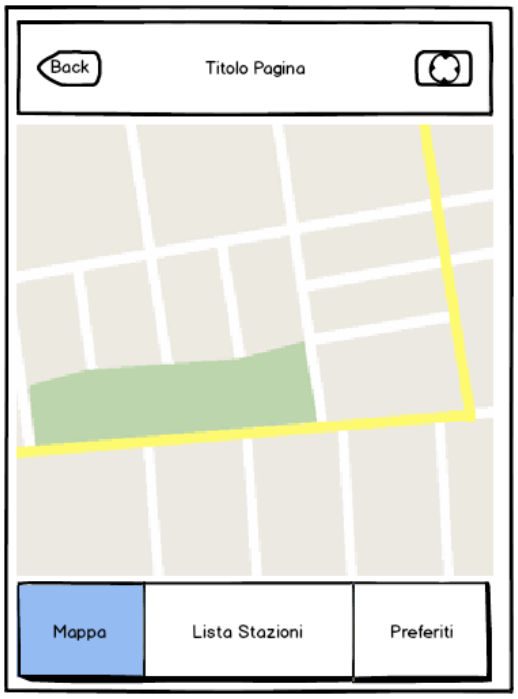

Figura 6.4: Layout ispirato ad Android, mockup della mappa

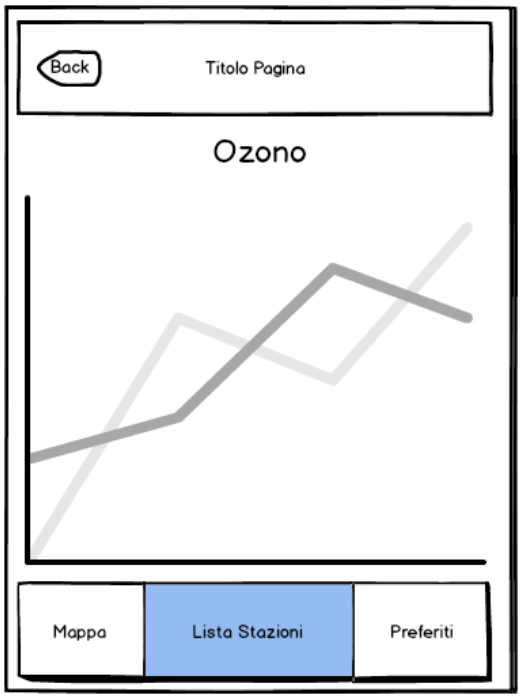

Figura 6.5: Layout ispirato ad Android, mockup del grafico di ozono

La proposta ha il vantaggio di rendere semplice la navigazione utente all'interno dell'applicazione, permettendogli di passare tra le varie funzionalità (mappa, lista

di stazioni, sistema di allerta) in modo semplice, e senza costringerlo a tornare ad un menu ogniqualvolta desideri passare da una funzionalità all'altra. Lo svantaggio è rappresentato dal grande utilizzo di spazio verticale dovuto alla presenza costante sia di header che di footer su schermo, che su dispositivi dallo schermo molto piccolo renderebbe difficilmente fruibili i grafici di andamento degli inquinanti, funzione primaria dell'applicazione.

6.3.2.2 Layout con menu iniziale

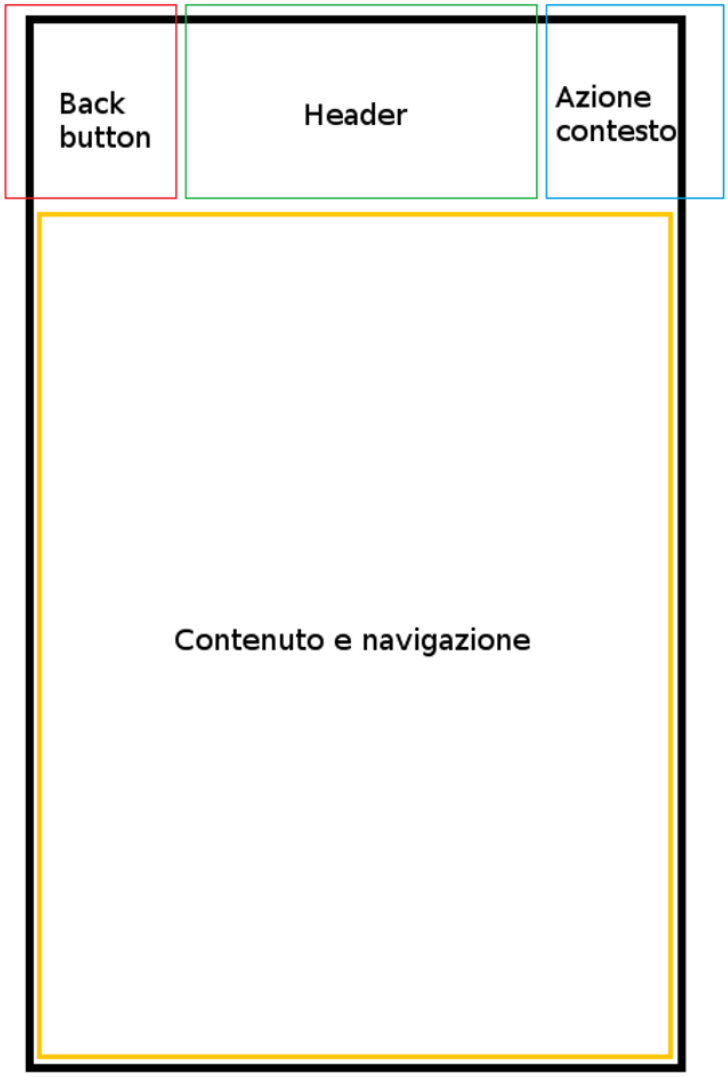

Figura 6.6: Layout con menu iniziale

Mantenendo inalterata la struttura della componente header, è possibile organizzare un layout che preveda una più ampia regione dedicata ai contenuti, abbandonando il footer di navigazione ed impostando la struttura dell'applicazione attorno ad un menu iniziale. Il layout viene presentato in figura 6.6, e se ne può

apprezzare il maggiore spazio dedicato ai grafici in figura 6.8, mentre un mockup del menu iniziale dal quale accedere alle varie funzionalità è riportato in figura 6.7.

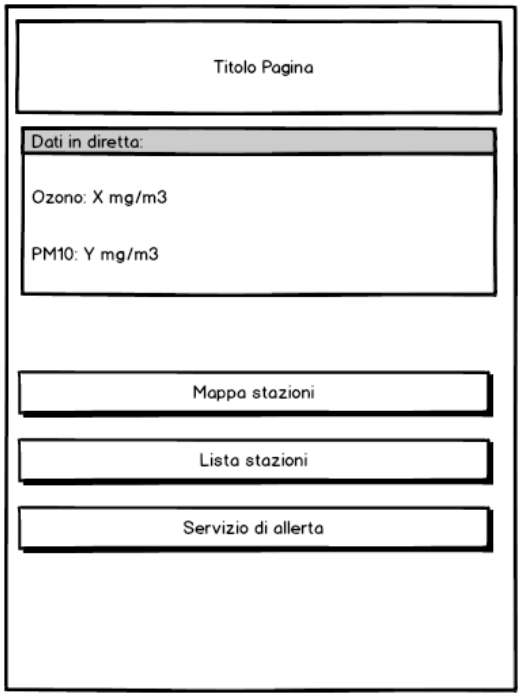

Figura 6.7: Layout con menu iniziale, mockup della mappa

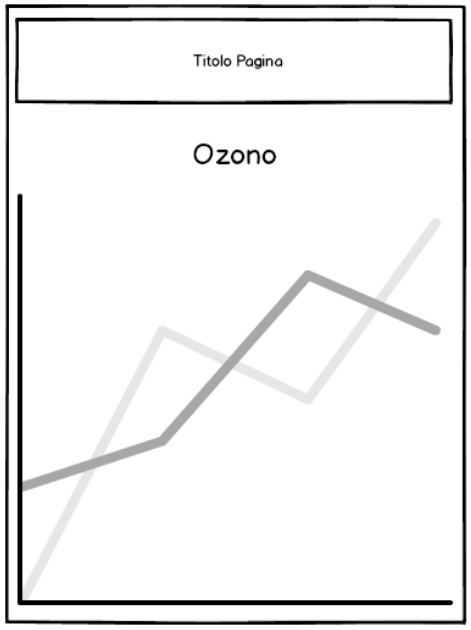

Figura 6.8: Layout con menu iniziale, mockup del grafico di ozono

Questo layout non prevede nessun menu di navigazione riportato in ogni pagina, ma si può immaginare comunque possa risultare usabile grazie alla poca profondità di navigazione utente possibile: si raggiungono quattro livelli massimi seguendo il percorso che dal menu porta alla lista delle province e successivamente delle stazioni, scegliendo una stazione e visualizzandone i dati validati da operatori.
A causa della maggior diffusione di dispositivi dallo schermo piccolo, e volendo riservare quanto più spazio possibile ai contenuti, si è optato per la realizzazione di questo secondo layout.

6.3.3 Schema di colori

Figura 6.9: Il logo ARPAV

Le applicazioni del progetto AppArpav non possiedono colori distintivi, è quindi possibile scegliere uno schema di colori liberamente. L'unico elemento comune a tutte le applicazioni già esistenti è l'utilizzo del logo ARPAV, riportato in figura 6.9 sia nello *splashscreen* iniziale all'avvio sia sopra i contenuti. Per creare uno stile coerente, è possibile utilizzare i colori distintivi del logo, in particolare il verde di sfondo alla scritta ARPAV (il colore *#2d7167* in notazione esadecimale), per personalizzare i colori propri dei fogli di stile del framework Bootstrap.
Si mostrano quindi in figura 6.10 gli elementi pricipali dell'interfaccia grafica e lo schema di colori scelto. Oltre alla scelta del colore per l'elemento header, notevoli differenze rispetto a Bootstrap sono i colori dei font utilizzati per i pulsanti e le voci dei menu: sono stati resi più scuri per migliorare il contrasto ed incontrare le linee guida WCAG 2.0.

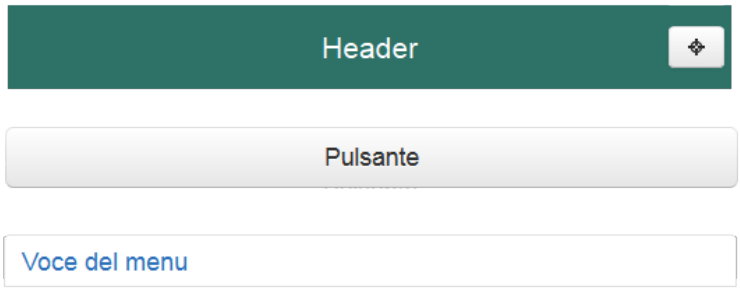

Figura 6.10: Elementi grafici dell'interfaccia e loro schema di colori

6.3.4 Il sistema a regioni

La libreria Backbone.Marionette che si è deciso di utilizzare per gestire le viste ed il loro rendering prevede la definizione di regioni, ovvero di elementi HTML già

presenti nel DOM all'interno dei quali effettuare il rendering delle viste. Lo stesso Marionette si occupa poi della gestione della memoria al cambio di viste, occupandosi della deallocazione degli elementi renderizzati e delle classi JavaScript non più necessarie.

Per adottare tale sistema si definiscono le regioni di navigazione e di contenuto in corrispondenza delle aree identificate in sezione 6.3.2.2, identificate da elementi con adeguato identificativo univoco. Il rendering degli elementi potrà essere effettuato entro tali regioni; fosse necessario estendere l'applicazione ed aggiungerne di nuove, basterebbe definire presso il controller dell'applicazione nuove regioni, ed effettuare il rendering delle viste appropriate entro di esse, senza che il resto del sistema richieda modifiche.

6.4 Progettazione del sistema

Per la realizzazione del sistema, si è scelto di interpretare il paradigma MVC proposto da Backbone con la creazione di una classe controller responsabile dell'istanziazione delle diverse classi View, relegando queste ultime al solo scopo di presentazione. Si è fatto inoltre uso dei design pattern Adapter per rendere il sistema il più possibile indipendente dalle API di geolocalizzazione o del provider delle mappe.
Di seguito si illustrano i design pattern utilizzati:

- **MVC**: come per il sistema server, anche per l'applicazione si è scelto di utilizzare il pattern Model-View-Controller per disaccoppiare le componenti e mantenere una rigida separazione di ruoli tra dati e presentazione. Come illustrato in precedenza, si è scelto di utilizzare un approccio standard utilizzando un controller che fungesse da punto di accesso all'applicazione, rendendo quindi le classi View responsabili solamente della presentazione e del rendering dei template HTML. Questo approccio, grazie al singolo punto di accesso all'applicazione, rende più semplice l'instanziazione dei dati e delle classi che aderiscono al pattern Singleton durante l'avvio;

- **Adapter**: il pattern Adapter ci permette di definire un'interfaccia alle librerie esterne, in modo da renderle compatibili con l'architettura e ridurre al minimo la dipendenza del sistema nei loro confronti, facilitando anche futuri cambi di librerie. Il sistema potrà accedere alle funzioni offerte dalla libreria adattata tramite l'interfaccia della classe adapter, ed un cambio nella libreria porterà a modifiche solo nella classe adattatore, e non nel resto del sistema. Nel progetto si utilizzano adapter in più punti: per il provider delle mappe, in modo da rendere trasparenti eventuali passaggi ad altri sistemi, per le librerie di creazione dei grafici, e per le API di geolocalizzazione;

- **Singleton**: il pattern Singleton impedisce la presenza di più di un'istanza di una data classe. Nei linguaggi di programmazione basati su prototipi, come JavaScript, questa definizione perde di significato ma, necessitando nel sistema di classi dal comportamento affine al Singleton, si è deciso di simularne

il comportamento tramite convenzioni: istanze di tali classi possono essere create solamente dal controller durante l'avvio dell'applicazione, e i loro riferimenti devono essere adeguatamente assegnati all'oggetto globale dell'applicazione. Ogni altra classe potrà accedere alle istanze simil-Singleton tramite opportuno campo dati dell'oggetto globale;

- **Accesso al Database**: per favorire l'indipendenza dalla base di dati, che consta di file JSON e XML scaricati dal server remoto, si sono definite due classi Singleton dal comportamento affine a quello di un DAO (Data Access Object): la prima ha il compito di scaricare i file remoti, e la seconda è in grado di parsare questi file ed instanziare i modelli di dati, popolandoli con gli opportuni campi dati.

6.4.1 Architettura e specifica delle classi

In figura 6.11 si mostra l'architettura ad alto livello dell'applicazione, divisa per componenti. Di seguito, si motivano le principali scelte progettuali, prima di illustrare ruolo e funzione di ogni classe individuata.

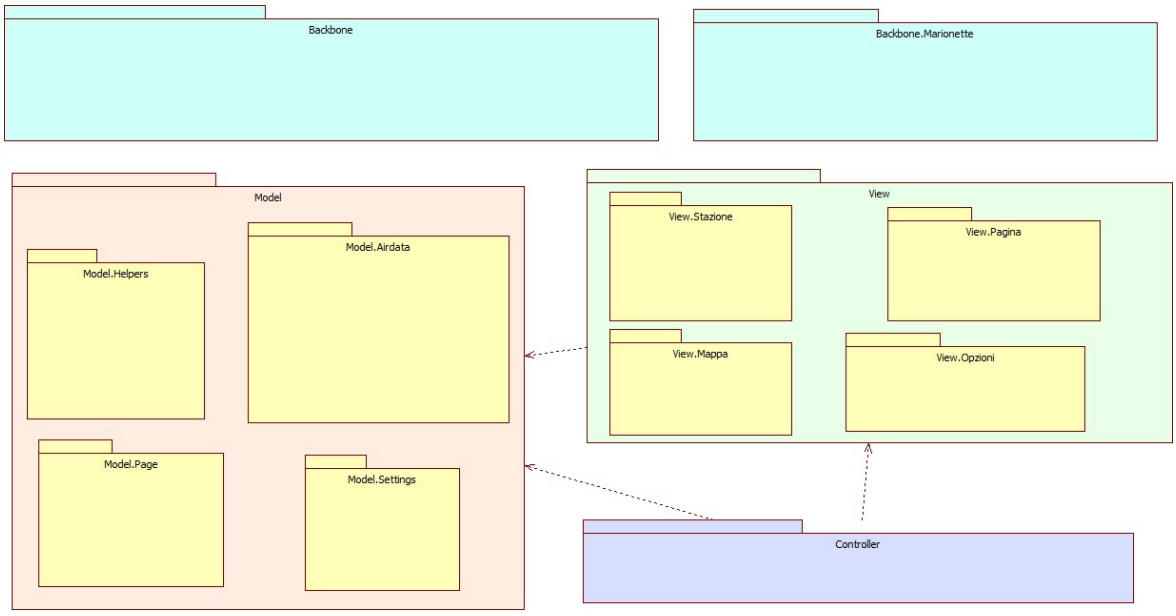

Figura 6.11: Architettura ad alto livello, per componenti, dell'applicazione

Si vuole individuare una gerarchia di stazioni di rilevamento che aggreghino generiche misure, le quali possono essere poi specializzate in misure specifiche. Questo permette al sistema di essere estendibile, nel caso si desiderasse presentare ulteriori dati (ad esempio la temperatura esterna, parametro forse oggetto di future estensioni del prodotto) propri della stazione e della sua zona. Ulteriori dati non rilevanti alla qualità dell'aria possono essere aggiunti alle stazioni di rilevamento opportune, le quali possono essere aggregate da ulteriori classi oltre ad Airdata.

Le restanti componenti, visto lo scopo primario di presentazione e stampa di dati scaricati dal server ed elaborati, non risultano particolarmente complesse. Si notino le componenti DataParser e DataProvider, che cooperano al recupero e formattazione dei dati prelevati da remoto, che come illustrato nella specifica lavorano a stretto contatto (e le loro dichiarazioni devono quindi essere considerate dei contratti, al pari delle interfacce), ma con divisione delle responsabilità: DataParser si limita a richiamare i metodi esposti da DataProvider, il quale si occupa di recuperare i dati in modo del tutto trasparente al parser.

GeoHelper, MapHelper e ModalHelper, inoltre, fungono da adapter nei confronti dei sistemi esterni di *Geolocation API*, *Leaflet.js* e *Bootstrap*.

6.4.1.1 Componente Model

Alla trattazione dei modelli si premette che Backbone permette di definirne due tipologie differenti: le collezioni, denominate Collection, che rappresentano degli aggregati di modelli, e i Model, responsabili di contenere i dati. Grazie ad Underscore.js, le collezioni forniscono metodi di utilità per scorrere e filtrare le collezioni, aiutando il recupero dei modelli di dati necessari.

6.4.1.1.1 Package Model::Airdata

Il package contiene le classi responsabili della gestione dei dati di qualità dell'aria, dalle stazioni di rilevamento fino ai singoli campionamenti.

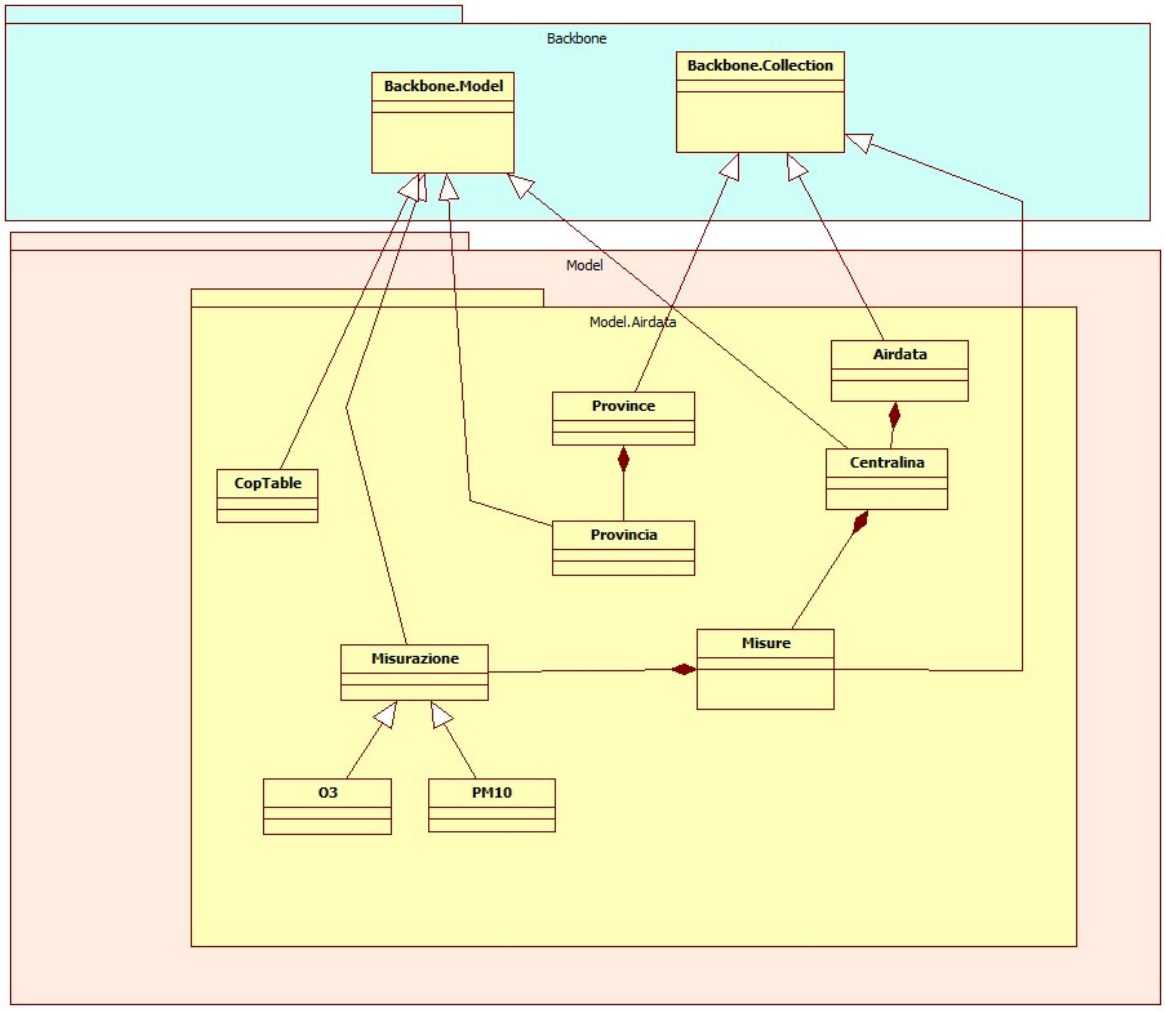

Figura 6.12: Architettura dell'applicazione, modulo Airdata

- **Model::Airdata::Airdata**

  Descrizione e utilizzo Classe collezione per i modelli della classe Centralina. Come aggregato rappresenta l'insieme dei dati di qualità dell'aria;

Classi ereditate – Backbone.Collection

- **Model::Airdata::Centralina**

  Descrizione e utilizzo Rappresenta una stazione di misurazione, contenendo l'insieme dei dati della stazione e le collezioni di Misurazioni per l'ozono e i PM10, oltre a metodi di utilità per la stampa di alcuni campi dati;

  Classi ereditate – Backbone.Model

- **Model::Airdata::CopTable**

  Descrizione e utilizzo Rappresenta l'insieme dei dati validati per una data stazione, contenendo i campi dati opportuni;

  Classi ereditate – Backbone.Model

- **Model::Airdata::Misurazione**

  Descrizione e utilizzo Rappresenta un rilevamento di un inquinante, caratterizzato dalla data e dal valore del rilevamento, funge da padre per le effettive misurazioni di ossigeno e PM10, che erediteranno da questa classe. La si definisce per permettere la creazione di una sola classe collezione Misure in grado di aggregare sia misurazioni di ozono che di PM10, senza dover creare una classe collezione per i modelli di ozono ed una per i modelli di PM10;

  Classi ereditate – Backbone.Model

- **Model::Airdata::Misure**

  Descrizione e utilizzo Classe collezione di oggetti Misurazione o classi da essa derivate, come ad esempio le classi O3 e PM10;

  Classi ereditate – Backbone.Collection

- **Model::Airdata::O3**

  Descrizione e utilizzo Rappresenta un rilevamento di ozono da parte di una data stazione, oltre a fornire metodi di utilità per la stampa;

  Classi ereditate – Model::Misurazione

- **Model::Airdata::PM10**

  Descrizione e utilizzo Rappresenta un rilevamento di PM10 da parte di una data stazione, oltre a fornire metodi di utilità per la stampa;

  Classi ereditate – Model::Misurazione

- **Model::Airdata::Province**

  Descrizione e utilizzo Classe collezione per gli oggetti di tipo Provincia;

  Classi ereditate – Backbone.Collection

- **Model::Airdata::Provincia**

  Descrizione e utilizzo Rappresenta una singola provincia con le sue stazioni di rilevamento della qualità dell'aria;

  Classi ereditate – Backbone.Model

6.4.1.1.2 Package Model::Helpers

Il package contiene gli adapter per le API di geolocalizzazione, di rappresentazione della mappa e di dialogo modale bloccante e non bloccante.

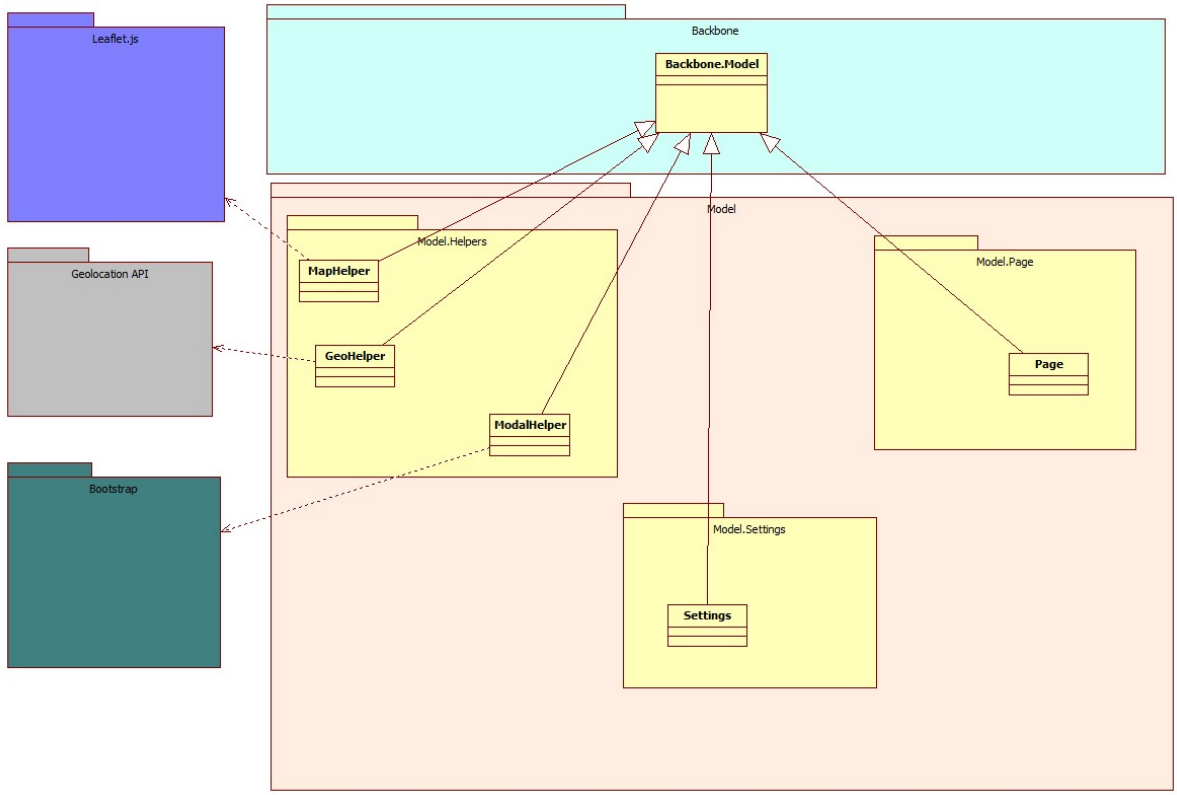

Figura 6.13: Architettura dell'applicazione, moduli Helper, Page e Settings

- **Model::Helpers::GeoHelper**

  Descrizione e utilizzo Classe adapter per le API di geolocalizzazione definite da HTML5, si occupa di fornire un'interfaccia alle funzionalità di geolocalizzazione, con metodi per individuare la posizione utente, con alta e bassa precisione a seconda delle risorse disponibili, e per individuare le distanze tra la posizione utente e la posizione delle stazioni. Ha quindi il compito di individuare all'avvio le stazioni più vicine all'utente per il rilevamento di ozono e PM10 (possono essere differenti, perché non tutte le stazioni rilevano entrambi gli inquinanti in diretta), in modo da fornire l'istantanea della qualità dell'aria. Si comporta come una classe Singleton, deve essere instanziata solo dal

controller all'avvio dell'applicazione ed un suo riferimento è presente come campo dati dell'oggetto globale dell'applicazione AA;

Classi ereditate – Backbone.Model

- **Model::Helpers::MapHelper**

  Descrizione e utilizzo Classe adapter per la libreria Leaflet.js, che si occupa della mappa interattiva. Ci permette quindi di definire delle funzioni relative alla stampa della mappa, al posizionamento del centro della mappa, al posizionamento dei segnalini per le stazioni e dei segnalini per la posizione utente. Il sistema non deve così sapere nulla sull'effettivo provider di mappe, ma si limita ad interagire con la classe definita quando necessario. Si comporta come una classe Singleton, deve essere instanziata solo dal controller all'avvio dell'applicazione ed un suo riferimento è presente come campo dati dell'oggetto globale dell'applicazione AA;

  Classi ereditate – Backbone.Model

- **Model::Helpers::ModalHelper**

  Descrizione e utilizzo Classe adapter nei confronti della libreria JavaScript del framework Bootstrap, per visualizzare dialoghi modali bloccanti o non bloccanti in caso di errori del sistema;

  Classi ereditate – Backbone.Model

6.4.1.1.3 Package Model::Page

Il package contiene le classi responsabili della gestione della pagina visualizzata all'utente.

- **Model::Page::Page**

  Descrizione e utilizzo Rappresenta una pagina visualizzata dall'applicazione, con i dati relativi al titolo e alla presenza o meno di pulsanti nelle due aree di controllo definite nell'header;

  Classi ereditate – Backbone.Model

6.4.1.1.4 Package Model::Settings

Il package contiene le classi responsabili della persistenza delle preferenze utente e della loro modifica.

- **Model::Settings::Settings**

  Descrizione e utilizzo Rappresenta le impostazioni utente, e si occupa della loro modifica e persistenza grazie alle API LocalStorage di HTML5 descritte in 6.2. Le informazioni utente salvate sono la lista di stazioni alle quali l'utente è interessato, per il servizio di allerta, e le preferenze

di attività o inattività del servizio di allerta stesso. Si comporta come una classe Singleton, deve essere instanziata solo dal controller all'avvio dell'applicazione ed un suo riferimento è presente come campo dati dell'oggetto globale dell'applicazione AA;

Classi ereditate – Backbone.Model

6.4.1.2 Componente Controller

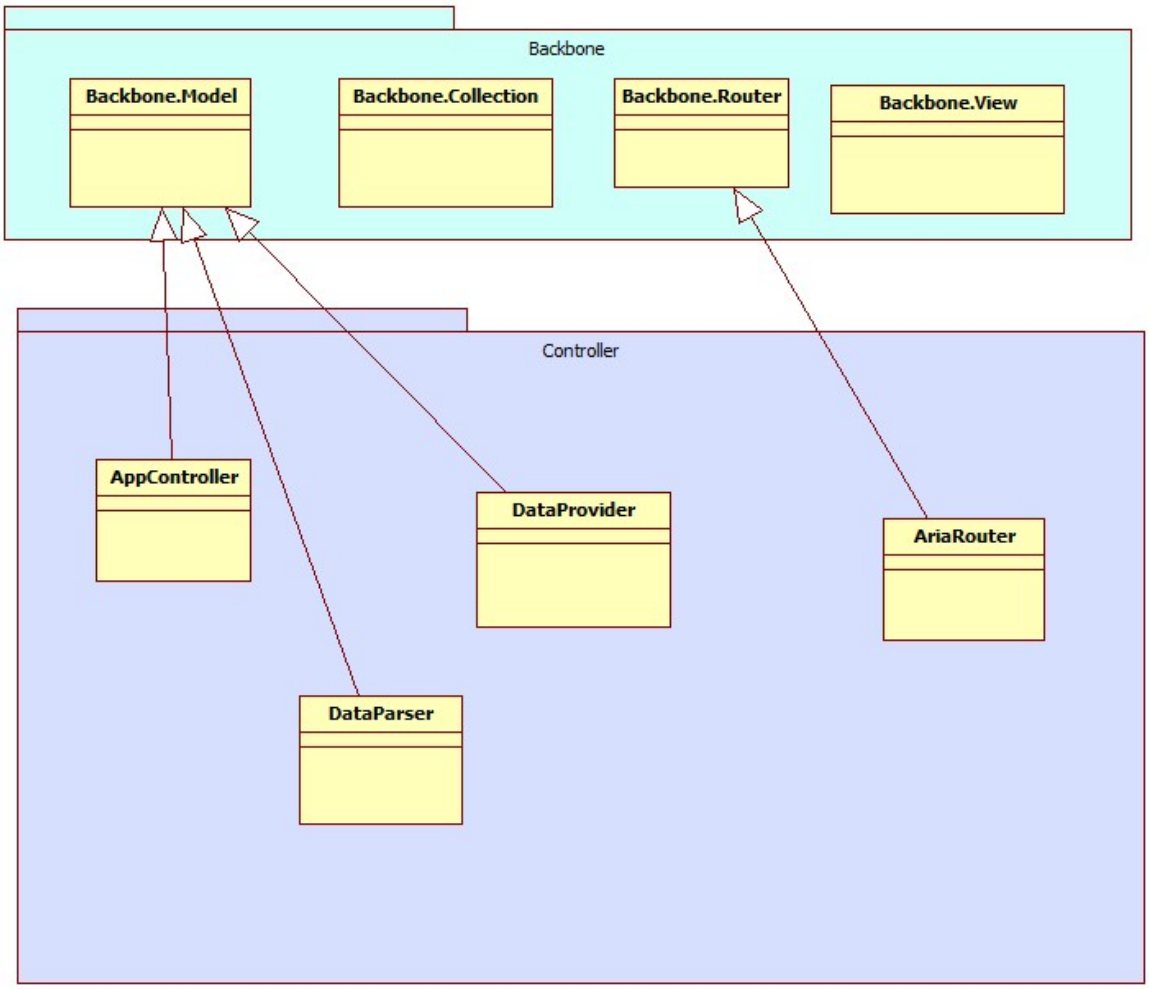

Figura 6.14: Architettura dell'applicazione, modulo Controller

- **Controller::AriaRouter**

  Descrizione e utilizzo Mappa di corrispondenza tra URL e stati del programma, grazie alla History API illustrata in 6.2. I percorsi disponibili sono pattern che Backbone monitora ogniqualvolta un nuovo stato dell'applicazione viene aggiunto alla cronologia tramite il metodo *navigate()*, ereditato dal router di Backbone. Quando rileva un match tra pattern definiti e stato dell'applicazione aggiunto alla cronologia, il

router chiama il proprio metodo associato che, per buona separazione delle responsabilità, dovrebbe semplicemente inoltrare la chiamata ad un metodo del controller che si occupi della gestione dell'evento. Si comporta come una classe Singleton, deve essere instanziata solo dal controller all'avvio dell'applicazione ed un suo riferimento è presente come campo dati dell'oggetto globale dell'applicazione AA;

Classi ereditate – Backbone.Router

- **Controller::DataParser**

  Descrizione e utilizzo Utilizza il DataProvider passatogli dal controller per parsare i file scaricati e popolare i modelli con i dati corretti. Si comporta come una classe Singleton, deve essere instanziata solo dal controller all'avvio dell'applicazione ed un suo riferimento è presente come campo dati dell'oggetto globale dell'applicazione AA;

  Classi ereditate – Backbone.Model

- **Controller::DataProvider**

  Descrizione e utilizzo Scarica i file JSON e XML contenenti i dati necessari all'applicazione. Questi file sono poi letti e parsati dalla componente DataParser del sistema. Si comporta come una classe Singleton, deve essere instanziata solo dal controller all'avvio dell'applicazione ed un suo riferimento è presente come campo dati dell'oggetto globale dell'applicazione AA;

  Classi ereditate – Backbone.Model

- **Controller::AriaRouter**

  Descrizione e utilizzo Controller dell'applicazione, fornisce un punto di accesso all'applicazione e si occupa dell'inizializzazione dei componenti del sistema all'avvio. Gestisce il passaggio delle informazioni tra modelli e viste, ed è responsabile dell'invio di segnali di modifica ai modelli da parte delle viste a seguito di azioni utente. Si comporta come una classe Singleton, viene instanziata all'avvio dell'applicazione ed un suo riferimento è presente come campo dati dell'oggetto globale dell'applicazione AA;

  Classi ereditate – Backbone.Model

6.4.1.3 Componente View

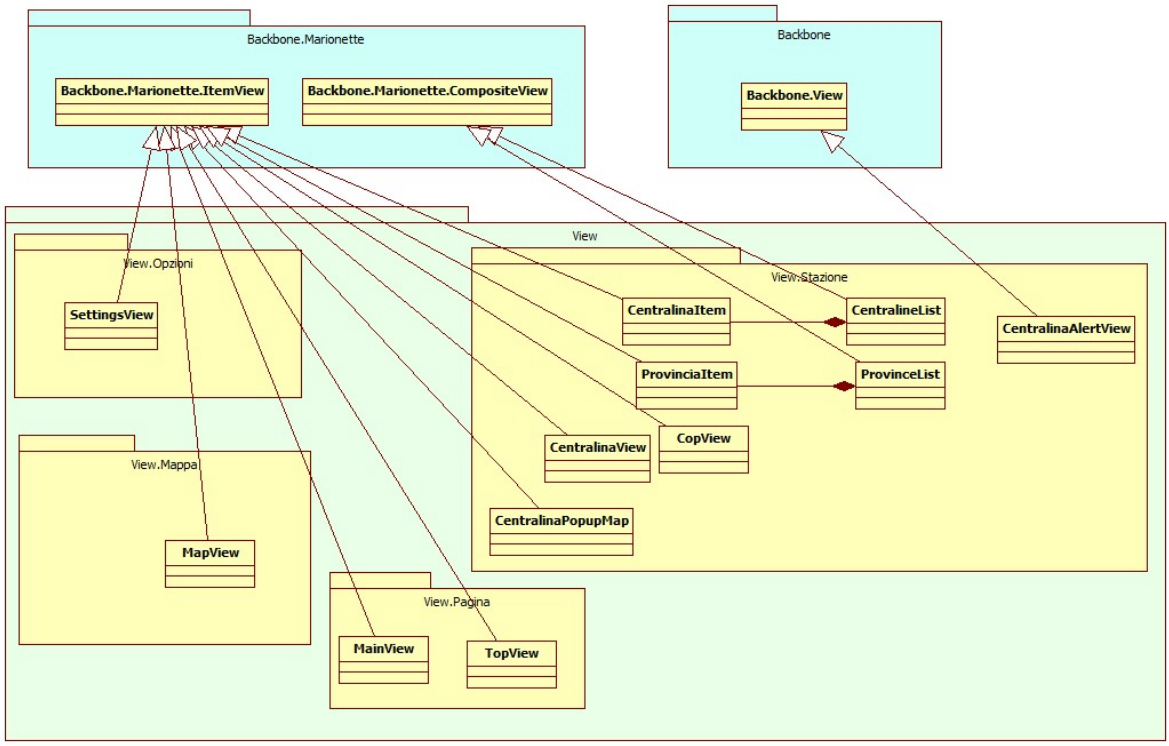

Figura 6.15: Architettura dell'applicazione, modulo View

Alla trattazione delle viste si premette che Backbone.Marionette, la libreria che aiuta ad applicare le *best practice* di Backbone, permette tre tipologie di viste distinte: quelle per renderizzare collezioni di modelli, quelle per renderizzare modelli ed infine delle viste che renderizzano collezioni di modelli ed informazioni aggiuntive. Le viste che permettono il rendering di collezioni constano della sola indicazione della vista da utilizzare per il rendering dei modelli in esse contenuti, mentre le cosiddette *CompositeView* oltre alla collezione permettono il rendering di informazioni aggiuntive e la gestione di eventi. Marionette fornisce infine viste che permettono il rendering dei modelli, le quali solitamente possiedono più informazioni, specialmente legate alle azioni da intraprendere a seguito dell'interazione utente.

Tutte le classi riportate hanno la responsabilità, nel caso provvedano al rendering di elementi con i quali l'utente può interagire, di monitorare tali eventi ed associarli ad appropriati metodi del controller. Ogni classe è associata ad un template HTML, non c'è cioè alcun markup HTML presente all'interno delle classi View, ma semplicemente un *selector* CSS che specifichi il nome del template da recuperare e popolare con i dati corretti. Questo aiuta a separare ancor di più la presentazione dei dati dal resto del sistema.

6.4.1.3.1 Package View::Stazione

Il seguente package contiene le classi che hanno la responsabilità di visualizzare dettagli relativi alle stazioni di qualità dell'aria, dalle pagine di dettagli fino agli elenchi di stazioni e province.

- **View::Stazione::CentralinaView**

  Descrizione e utilizzo Responsabile del rendering della pagina dei dettagli di una data stazione. Deve disegnare i grafici di andamento di ozono e PM10, e per questo motivo adatta le librerie Highcharts e Graph.js per l'utilizzo con i modelli di dati del sistema;

  Classi ereditate – Backbone.Marionette.ItemView

- **View::Stazione::CentralineList**

  Descrizione e utilizzo Responsabile del rendering della lista di centraline per una data provincia, passata come collezione. Delega il rendering delle singole centraline alla classe View::CentralinaItem;

  Classi ereditate – Backbone.Marionette.CompositeView

- **View::Stazione::CentralinaItem**

  Descrizione e utilizzo Responsabile del rendering della voce di una data centralina, passata come modello, in una lista di centraline;

  Classi ereditate – Backbone.Marionette.ItemView

- **View::Stazione::CopView**

  Descrizione e utilizzo Responsabile del rendering della tabella di dati validati dagli operatori per una data stazione, passata come modello;

  Classi ereditate – Backbone.Marionette.ItemView

- **View::Stazione::ProvinceList**

  Descrizione e utilizzo Responsabile del rendering della lista di province presenti, passata come collezione. Delega il rendering delle singole voci alla classe View::ProvinciaItem;

  Classi ereditate – Backbone.Marionette.CompositeView

- **View::Stazione::ProvinciaItem**

  Descrizione e utilizzo Responsabile del rendering della voce di una data provincia, passata come modello, in una lista di province;

  Classi ereditate – Backbone.Marionette.ItemView

- **View::Stazione::CentralinaAlertView**

Descrizione e utilizzo Responsabile del rendering di una singola voce della lista di stazioni per le quali sia stato attivato il servizio di allerta ozono, permettendo anche la disiscrizione utente dalla stessa. Non derivando da una delle viste automatiche di Marionette, deve essere fornita di metodo *render()*, da chiamare esplicitamente quando si desidera visualizzare l'elemento;

Classi ereditate – Backbone.View

- **View::Stazione::CentralinaPopupMap**

  Descrizione e utilizzo Responsabile del rendering del popup associato al segnaposto di una data stazione nella mappa interattiva;

  Classi ereditate – Backbone.Marionette.ItemView

6.4.1.3.2 Package View::Pagina

Il seguente package contiene le classi responsabili della gestione delle pagine dell'applicazione.

- **View::Pagina::MainView**

  Descrizione e utilizzo Responsabile del rendering della prima pagina dell'applicazione, contenente i dati in diretta se l'utente è geolocalizzato, del menu per l'accesso alle varie funzionalità disponibili, e della gestione dell'interazione utente con i menu;

  Classi ereditate – Backbone.Marionette.ItemView

- **View::Pagina::TopView**

  Descrizione e utilizzo Responsabile del rendering dell'header della pagina, passato come modello: dal titolo all'eventuale presenza (e azione prevista) per i pulsanti di navigazione;

  Classi ereditate – Backbone.Marionette.ItemView

6.4.1.3.3 Package View::Mappa

Il seguente package contiene le classi responsabili delle visualizzazioni della mappa.

- **View::Mappa::MapView**

  Descrizione e utilizzo Responsabile del rendering della mappa interattiva di OpenStreetMaps, e della gestione degli eventi di selezione delle stazioni da parte dell'utente;

  Classi ereditate – Backbone.Marionette.Itemview

6.4.1.3.4 Package View::Opzioni

Il seguente package contiene le classi responsabili di visualizzazione e modifica delle opzioni utente.

- **View::Opzioni::SettingsView**

  Descrizione e utilizzo Responsabile del rendering della pagina di configurazione utente, dalla quale è possibile modificare l'attività del servizio di allerta utente e visualizzare le centraline per le quali si è attivato lo stesso. È possibile anche la disiscrizione dalle singole stazioni;

  Classi ereditate – Backbone.Marionette.ItemView

6.5 Strategie per l'accessibilità del prodotto

Nell'ottica di alleggerire il processo di verifica, si cerca di progettare in anticipo l'accessibilità del prodotto. Alcune strategie per garantire la migliore accessibilità possibile sono già state adottate:

- Accessibilità della mappa interattiva: la componente di mappa risulta completamente inaccessibile a persone che non possono visualizzare le immagini di cui è composta, oppure che non conoscono formalismi e modalità di lettura delle mappe. Inoltre, non c'è controllo sulla qualità di colori e contrasto di colori della mappa generata da provider terzi, che potrebbe quindi risultare poco leggibile per determinate categorie di utenti con alterazioni della percezione dei colori. Questo è stato mitigato già in sede di Analisi dei Requisiti, prevedendo un elenco di stazioni di qualità dell'aria dal pari valore informativo, ma composto di solo testo. In questo modo, si provvede con un'alternativa testuale al contenuto per immagini;

- Contrasti di colori: l'aderenza a livello AA alle linee guida WCAG 2.0 relative al contrasto di colori è stata progettata ed illustrata nella sezione 6.3.3, e già oggetto di verifica tramite lo strumento ColorChecker.

Sarà necessario realizzare l'interfaccia utente ed il suo markup HTML e CSS seguendo rigidamente gli standard, ed adottando tutte le linee guida WGAC 2.0 possibili. Si rimanda illustrazione del lavoro svolto e dei risultati della verifica di accessibilità alla sezione 7.3.4, dove per ogni elemento applicabile (e non triviale) sarà data motivazione del *success criteria* e, se utilizzate, delle tecniche W3C impiegate per rispettare la linea guida.

7 Implementazione e testing dell'applicazione

Lo sviluppo dell'applicazione, avvenuto per incrementi seguiti da revisioni informali del prodotto assieme ai responsabili ARPAV per verificare il buon progresso del lavoro, è proceduto secondo i tempi previsti (malgrado la curva di apprendimento giudicata molto ripida del framework Backbone).

7.1 Il sistema opzionale di allerta

Notabile eccezione al buon esito dello sviluppo, la componente opzionale dell'applicazione, il sistema di allerta ozono, è stata sviluppata solamente in parte, ovvero nella componente che permette all'utente di scegliere le stazioni per le quali desidera essere informato. Manca cioè la componente di codice responsabile dell'invio al server dell'iscrizione al servizio di allerta, e la componente di codice responsabile della ricezione delle notifiche push e del conseguente avviso all'utente. Questo è dovuto all'esaurirsi del tempo programmato per lo sviluppo, a causa dello sviluppo imprevisto del sistema server illustrato in sezione 2, ed al non poter sottrarre tempo residuo da aggiungere alle attività di sviluppo e verifica alle attività di build e pubblicazione dell'applicazione, giudicate di primaria importanza da ARPAV.
Di concerto con ARPAV, si è quindi deciso di preservare la funzionalità di scelta dell'utente delle stazioni, già funzionante, trasformandola però in un sistema di preferiti che permetta all'utente di accedere velocemente alle stazioni preferite. Sfruttando la persistenza delle impostazioni grazie a LocalStorage e la pagina di riepilogo e disiscrizione dai preferiti, è stato possibile preservare componenti già sviluppate e ben testate aggiungendo una funzionalità al prodotto, mitigando parzialmente l'impatto sulla riuscita del progetto dovuto all'esaurirsi del tempo.

7.2 Esiti delle attività di sviluppo

Il prodotto è stato realizzato con strategia incrementale, realizzando le componenti del sistema critiche prima di tutto e poi procedendo per integrazione di componenti successive. Si riportano di seguito gli esiti delle attività di sviluppo del prodotto.

7.2.1 Requisiti soddisfatti

Di seguito, si elencano i requisiti soddisfatti tra i requisiti individuati nell'attività di analisi. Si noti che tutti i requisiti obbligatori sono stati soddisfatti, assieme ad una parte di desiderabili e opzionali.

7.2.1.0.5 Requisiti funzionali

Codice	Tipologia	Descrizione	Fonti
RF1	Obbligatorio	L'utente deve poter selezionare le operazioni relative alla qualità dell'aria	soddisfatto
- RF1.1	Obbligatorio	L'utente deve poter scegliere di visualizzare la mappa relativa alla qualità dell'aria	soddisfatto
- RF1.2	Desiderabile	L'utente deve poter scegliere di visualizzare la lista delle centraline relativa alla qualità dell'aria	soddisfatto
- RF1.3	Desiderabile	Il programma deve fornire i dati rilevati nella centralina più vicina per ogni inquinante	soddisfatto
RF2	Desiderabile	Il programma deve avvisare l'utente in caso di allerta relativa ai livelli di ozono	non soddisfatto
- RF2.1	Desiderabile	L'utente deve poter scegliere per quali centraline attivare il servizio allerta	soddisfatto
– RF2.1.1	Obbligatorio	L'utente deve poter iscriversi al servizio allerta ozono per una data centralina	soddisfatto
– RF2.1.2	Desiderabile	L'utente deve poter disiscriversi dal servizio allerta ozono per una data centralina	soddisfatto
– RF2.1.3	Obbligatorio	L'utente deve poter visualizzare le centraline presso le quali è registrato al servizio di allerta	soddisfatto
- RF2.2	Desiderabile	L'utente deve poter attivare o disattivare il servizio di allerta	non soddisfatto
RF3	Obbligatorio	Il programma deve fornire una mappa del Veneto	soddisfatto

- RF3.1	Desiderabile	L'utente deve poter navigare all'interno della mappa	soddisfatto
- RF3.2	Desiderabile	L'utente deve poter modificare lo zoom della mappa	soddisfatto
- RF3.3	Desiderabile	L'utente deve poter visualizzare la propria posizione sulla mappa	soddisfatto
- RF3.4	Obbligatorio	La mappa deve mostrare le centraline di rilevazione della qualità dell'aria	soddisfatto
– RF3.4.1	Opzionale	Le centraline troppo vicine tra loro devono essere raggruppate con un unico simbolo	non soddisfatto
– RF3.4.2	Obbligatorio	L'utente deve poter selezionare una centralina	soddisfatto
— RF3.4.2.1	Obbligatorio	Il programma deve presentare le informazioni di base della centralina selezionata	soddisfatto
— RF3.4.2.2	Obbligatorio	L'utente deve poter visualizzare le informazioni dettagliate della centralina selezionata	soddisfatto
—- RF3.4.2.2.1	Obbligatorio	L'utente deve poter visualizzare i dati validati dagli operatori per la centralina	soddisfatto
—- RF3.4.2.2.2	Obbligatorio	Il programma deve visualizzare grafici di andamento degli inquinanti	soddisfatto
—— RF3.4.2.2.2.1	Obbligatorio	Il programma deve visualizzare l'andamento dell'ozono nelle precedenti 48 ore	soddisfatto
—— RF3.4.2.2.2.2	Obbligatorio	L'utente deve poter visualizzare l'andamento di PM10 dei 7 giorni precedenti	soddisfatto
—— RF3.4.2.2.2.3	Desiderabile	Deve essere possibile visualizzare il valore dei punti rappresentati sui grafici	soddisfatto*

— RF3.4.2.3	Desiderabile	L'utente deve poter selezionare una centralina dalla lista delle centraline	soddisfatto
— RF3.4.2.4	Obbligatorio	L'utente deve poter selezionare una centralina dalla mappa	soddisfatto
RF4	Obbligatorio	Il programma deve recuperare i dati aggiornati all'avvio	soddisfatto
- RF4.1	Obbligatorio	Il programma deve recuperare i dati di ozono relativi alle 48 ore precedenti al momento dell'avvio	soddisfatto
– RF4.1.1	Opzionale	Se sono presenti dati al momento dell'avvio, il programma deve recuperare solo le informazioni mancanti	non soddisfatto
- RF4.2	Obbligatorio	Il programma deve recuperare i dati di PM10 relativi ai 7 giorni precedenti al momento dell'avvio	soddisfatto

Tabella 7.1: Tabella di soddisfacimento dei requisiti funzionali

* Soddisfatto per ogni piattaforma ≠ Android 2.3.X

7.2.1.0.6 Requisiti qualitativi

Codice	Tipologia	Descrizione	Fonti
RQ9	Obbligatorio	Devono essere rispettate tutte le metriche sulla stesura di codice stabilite	soddisfatto
RQ10	Obbligatorio	Deve essere prodotta documentazione del codice sorgente del software	soddisfatto

Tabella 7.2: Tabella di soddisfacimento dei requisiti qualitativi

7.2.1.0.7 Requisiti di vincolo

Codice	Tipologia	Descrizione	Fonti
RV5	Obbligatorio	Il programma deve funzionare su Android 2.3	soddisfatto
RV6	Obbligatorio	Il programma deve funzionare su Android 4.2	soddisfatto
RV7	Obbligatorio	Il programma deve funzionare su iOS6	soddisfatto
RV8	Obbligatorio	Il programma deve funzionare su Windows Phone 7	soddisfatto

Tabella 7.3: Tabella di soddisfacimento dei requisiti di vincolo

7.2.2 Grafici e supporto a SVG

La libreria scelta inizialmente per il disegno dei grafici, Highcharts, ne permette la creazione lato client in maniera dinamica, grazie al passaggio di dati e parametri tramite opportuni oggetti JSON. La visualizzazione avviene poi tramite oggetti SVG inseriti nel DOM.

Le versioni di Android inferiori alla versione 3.0.0, quindi tutte le versioni 2.3.X per le quali si vuole garantire il funzionamento del prodotto, hanno un supporto limitato o assente ad SVG. Gli autori della libreria Highcharts dichiarano di realizzare una sorta di retrocompatibilità tramite l'uso di un motore di rendering differente per tali versioni di Android, teoricamente in grado di leggere elementi SVG e ridisegnarlo in un elemento *canvas*.

Delle prime prove su alcuni dispositivi interessati dal problema hanno mostrato discreti risultati, con grafici interattivi e funzionanti, sebbene non eccessivamente performanti, ma comunque giudicati sufficienti per la realizzazione del progetto. Una successiva validazione del progetto durante i rilasci settimanali interni ha mostrato come, su un ulteriore dispositivo dotato di Android 2.3.6 la soluzione risultasse non più funzionante, a causa del mancato disegno dei grafici e quindi dell'apparente fallimento della modalità di compatibilità offerta dalla libreria.

La soluzione migliore è sembrata quella di scegliere una libreria che rappresentasse i grafici tramite disegno su elementi *canvas*, individuata in Chart.js, dal funzionamento ed API simili ad Highcharts ma senza alcun supporto per l'interattività a seguito di azioni utente quali il tocco sui punti rappresentati nel grafico. Tramite la libreria Modernizr, è facile riconoscere con un test a runtime di veloce esecuzione se il browser sottostante offre supporto ad SVG o meno, e grazie a questo dato è possibile instanziare la corretta libreria tramite i costruttori offerti dalle stesse richiamabili tramite adapter.
Questo è stato implementato con successo ed è risultato un approccio funzionante alla soluzione del problema, adottato nel rilascio 1.0.0 dell'applicazione.

7.3 Esiti delle attività di verifica

Di seguito si riportano gli esiti delle attività di verifica svolte durante e successivamente all'attività di codifica.

7.3.1 Analisi statica

Il prodotto è stato costantemente monitorato, tramite lo strumento Plato, per garantire il rispetto dei parametri fissati per le metriche di analisi statica. Si riportano i risultati finali ottenuti in forma tabulare. Con il colore rosso si evidenziano risultati che non rispettano i parametri fissati, mentre con il colore arancione si evidenziano risultati che rispettano solamente il range di accettazione definito, ma non il range ottimale.

Modulo	Complessità ciclomatica	Indice di manutenibilità	Numero di errori stimato
Controller:: appcontroller	17*	65	2.14
Controller:: ariarouter	1	93	0.19
Controller:: dataparser	8	68	1.69
Controller:: dataprovider	1	94	0.07
Model:: airdata	1	63	0.02
Model:: centralina	1	71	0.20
Model:: coptable	1	59	0.12
Model:: geohelper	7	69	1.58
Model:: maphelper	3	66	0.61
Model:: misurazione	1	82	0.03
Model:: misure	1	83	0.02
Model:: modalhelper	2	83	0.12
Model:: o3	1	83	0.04
Model:: page	1	68	0.06

Model:: pm10	1	83	0.04
Model:: province	1	83	0.02
Model:: provincia	1	87	0.03
Model:: settings	5	63	0.41
View:: centralinaitem	1	83	0.08
View:: provinciaitem	1	83	0.08
View:: centralinaalertview	2	80	0.24
View:: centralinapopupmap	1	83	0.08
View:: centralinaview	4	56	3.65
View:: centralinelist	1	74	0.08
View:: copview	1	73	0.08
View:: mainview	1	84	0.30
View:: mapview	1	65	0.12
View:: provincelist	1	74	0.08
View:: settingsview	1	79	0.31

View:: topview	1	95	0.10

Tabella 7.4: Risultati di analisi statica, applicazione mobile

* Il valore calcolato di complessità ciclomatica non rappresenta correttamente la complessità del modulo, essendo causato da assegnamenti di valori di default a variabili effettuati tramite operatore condizionale ternario (x ? y : z), che provoca un aumento di cammini che però si escludono vicendevolmente (similmente al problema del calcolo della complessità ciclomatica in presenza di operatori *switch*). Ignorando il metodo in questione, la complessità ciclomatica del modulo risulta essere 3, valore che ricade nel range ottimale stabilito.

7.3.2 Test di unità del prodotto

Per la verifica delle singole unità di codice, metodi e classi, si è proceduto con una strategia volta ad individuare le parti di codice meritevoli di test di unità specifici (ad esempio, per i modelli e collezioni, trattandosi solamente di contenitori di campi dati è stata giudicata sufficiente la verifica della loro buona integrazione con il restante sistema).
Una volta individuate le componenti sensibili, come ad esempio quelle preposte al *parsing* dei file o alla geolocalizzazione e manipolazione di distanze delle stazioni, esse sono state sviluppate con un approccio *test driven*: si sono definiti test di unità specifici con lo strumento Jasmine, e si è successivamente prodotto il codice sorgente dei moduli.
Le componenti sono poi state integrate tra di loro sempre mediante appositi test automatici.

Come aiuto alla modalità di test automatica permessa da Jasmine, si è sfruttato inoltre lo strumento Firebug, precedentemente descritto in sezione 6.2, per ottenere istantanee dello stato di esecuzione del software in dati momenti, e per poter accedere allo stato delle istanze degli oggetti definiti in ogni momento, permettendo a volte la più facile verifica del buon funzionamento di alcune classi definite.

7.3.3 Test di validazione

Di seguito si riporta l'esito dei test di validazione formale del prodotto, effettuato alla presenza dei responsabili di ARPA Veneto, sulla versione 1.0.0 dell'applicazione. La versione oggetto di test di validazione sarà successivamente pubblicata, a meno di fallimento dei test, per i dispositivi Android, iOS e Windows Phone.

Codice del test	Prospettive del test	Esito
TV1	Verificato dall'esito dei test figli	superato in tutte le componenti implementate
- TV1.1	Si verifica il buon avvio del programma e la presenza delle opzioni di qualità dell'aria	superato
– TV1.1.1	Si verifica la possibilità di selezione dell'opzione mappa	superato
— TV1.1.1.1	Si verifica la presenza della mappa della regione veneto	superato
— TV1.1.1.2	Si verifica che la navigazione all'interno della mappa sia permessa	superato
— TV1.1.1.3	Si verifica che la modifica dello zoom della mappa sia permessa	superato
— TV1.1.1.4	Si verifica che in corrispondenza della posizione presunta sia presente il segnalino della propria posizione	superato
— TV1.1.1.5	Si verifica la presenza del segnalino in posizione corretta per un dato numero di stazioni campione	superato
— TV1.1.1.6	Si verifica che a minor livello di dettaglio permesso le stazioni del centro di Padova siano raggruppate in un unico segnalino	non implementato
— TV1.1.1.7	Si verifica la possibilità di selezionare un segnalino indicante una stazione campione	superato
— TV1.1.1.8	Si verifica la comparsa e correttezza delle informazioni riguardanti la stazione selezionata	superato
— TV1.1.1.9	Si verifica la possibilità di selezione della visualizzazione dettagliata della stazione	superato
— TV1.1.1.10	Si verifica la presenza dei dati validati dagli operatori per la stazione selezionata	superato
– TV1.1.2	Si verifica la possibilità di selezione dell'opzione elenco centraline	superato
— TV1.1.2.1	Si verifica la possibilità di selezione di una centralina dalla lista	superato

- TV1.2	Si verifica la presenza dei dati di qualità dell'aria relativi alla più vicina centralina per ogni inquinante	superato
TV2	Verificato dall'esito dei test figli	superato in tutte le componenti implementate*
- TV2.1	Si verifica che ogni stazione che rileva ozono preveda la possibilità di iscrizione al servizio di allerta	superato*
- TV2.2	Si verifica la possibilità di iscrizione al servizio di allerta per una stazione campione	superato*
- TV2.3	Si verifica la possibilità di disiscrizione al servizio di allerta per una stazione campione	superato*
- TV2.4	Si verifica la presenza delle stazioni presso le quali ci si è iscritti al servizio di allerta	superato*
- TV2.5	Si verifica la possibilità di attivazione e disattivazione del servizio di allerta	non implementato
- TV2.6	Si verifica la ricezione di una notifica di allerta per una data stazione	non implementato
TV3	Verificato dall'esito dei test figli	superato**
- TV3.1	Si verifica la presenza del grafico di andamento dell'ozono nelle 48 ore precedenti, se la stazione selezionata lo rileva	superato
- TV3.2	Si verifica la presenza del grafico di andamento del PM10 per i 7 giorni precedenti, se la stazione selezionata lo rileva	superato
- TV3.3	Si verifica la possibilità di selezione di un punto del grafico e la comparsa di informazioni relative al punto selezionato	superato**

TV4	Verificato dall'esito dei test figli	superato in tutte le componenti implementate
- TV4.1	Si verifica che gli ultimi dati di ozono per una data stazione siano gli ultimi pubblicati dal server	superato
- TV4.2	Si verifica che gli ultimi dati di PM10 per una data stazione siano gli ultimi pubblicati dal server	superato
- TV4.3	Si verifica che il consumo di dati in download all'avvio sia minore della dimensione dell'attuale file JSON, se il programma è stato avviato almeno una volta durante i precedenti 7 giorni	non implementato

Tabella 7.5: Riepilogo dei test di validazione, applicazione mobile

* Si noti che il sistema di allerta ozono non è stato implementato in tutte le sue componenti, ed è quindi stato rimosso dalla versione 1.0.0 che si rilascia al pubblico. I test marcati come superato si riferiscono alle componenti comunque realizzate ed oggetto di validazione da parte di ARPAV, che ha deciso poi di rinominare la parte funzionante del servizio di allerta come servizio preferenze utente.

** A causa dei problemi dovuti ad SVG di cui si è trattato in sezione 7.2.2, il test è da considerarsi superato per dispositivi dotati di una versione di Android $\geq 4.0.0$, e fallito a causa di componente non realizzata per versioni di Android $< 4.0.0$.

7.3.4 Verifica dell'accessibilità del prodotto

Si dedica una sezione apposita agli esiti della verifica dell'accessibilità del prodotto, divisi per linea guida WCAG 2.0. Si riportano solo le linee guida applicabili al progetto, escludendo quindi ad esempio quelle relative al contenuto multimediale basato sul tempo, come i video.

7.3.4.1 Guideline 1.1: *Text Alternatives*

La linea guida 1.1 prevede che si provveda con alternative testuali a contenuti diversi dal testo, a meno che il contenuto non sia di pura decorazione.

In particolare, lo scenario B della linea guida 1.1.1 prevede che, per elementi quali i grafici, dove un'alternativa testuale non avrebbe pari valenza informativa, se ne produca una breve descrizione. Questo è stato applicato nel progetto in corrispondenza dei grafici di andamento di ozono e PM10, dove si fornisce assieme al grafico l'informazione giudicata più rilevante dello stesso, ovvero il valore massimo dell'inquinante misurato nel periodo di tempo.

Si noti che la funzione di mappa del Veneto dell'applicazione, secondo le linee guida, è quindi inaccessibile all'utente. Questo era previsto, ed è stata progettata in anticipo un'adeguata contromisura strutturale, come descritto in sezione 6.5, grazie alla funzionalità di lista delle stazioni. Inoltre, l'ulteriore informazione importante che la mappa riporta all'utente, ovvero una stima della distanza dalla stazione più vicina, è trasmessa come testo direttamente all'avvio dell'applicazione, grazie alla funzionalità di rilevamento della stazione più vicina per ogni inquinante e della sua comunicazione all'utente con una stima in chilometri della distanza.

7.3.4.2 Guideline 1.3: *Adaptable*

La linea guida 1.3 prevede la creazione di contenuto che possa essere presentato in modi diversi senza perdita di struttura od informazioni.

La linea guida 1.3.1 prevede che informazioni e relazioni comunicate attraverso la presentazione siano disponibili come testo oppure siano determinabili programmaticamente, ad esempio tramite apposito markup. Questa risulta rispettata da ogni elemento applicabile: non si usano mai tag di presentazione senza che siano abbinati ad apposito markup, a meno che non si tratti di pura presentazione (ad esempio la minore dimensione dell'informazione sugli inquinanti rilevati per ogni stazione). Si utilizza inoltre, per aggiungere ulteriore semantica all'applicazione, il tag *address* per le informazioni relative all'indirizzo delle stazioni.

Informazione e presentazione sono rigidamente separate tramite l'utilizzo di fogli di stile CSS, ed informazioni tabulari sono presentate con il corretto markup delle tabelle, ad esempio nel caso della tabella di dati validati di qualità dell'aria.

L'aderenza alle linee guida della tabella realizzata è stata aiutata dalle tecniche WCAG H39 e H73, che prevedono la presenza di elementi *summary* e *caption*, utilizzando però solamente l'elemento *caption* perché l'elemento *summary* per le tabelle non è supportato da HTML5 e produce quindi un documento invalido.

Per le restanti linee guida si verifica il corretto utilizzo del markup HTML, specialmente per quanto riguarda liste di elementi e per il corretto uso dell'ordine delle intestazioni all'interno della pagina.

7.3.4.3 Guideline 1.4: *Distinguishable*

La linea guida 1.4 prevede che per l'utente la visualizzazione dei contenuti sia semplice.

La linea guida 1.4.1 in particolare prevede che il colore non sia l'unico modo attraverso il quale comunicare un'informazione. Questo è stato rispettato nel prodotto per ogni elemento che utilizza il colore, come illustrato di seguito.
Si noti che la tabella dei dati validati dagli operatori, la cui realizzazione era richiesto fosse quanto più aderente possibile a quella del sito web aziendale, è stata resa conforme alla linea guida nel prodotto: la tabella presente nel sito utilizza il colore rosso oppure verde per indicare se ci sia stato rispetto dei parametri di legge per un dato inquinante, senza però provvedere con una descrizione dello stesso colore. Nel progetto, si è aggiunta invece esplicita descrizione testuale, eventualmente colorata di rosso, in caso di mancato rispetto dei valori di legge. La situazione di scarsa accessibilità delle informazioni nel sito web è stata inoltre segnalata ad ARPAV.
Altro elemento d'uso del colore è per identificare i link. Gli stessi però sono evidenziati tramite apposita sottolineatura, oppure con la rappresentazione come elemento di blocco simile ad un pulsante nel caso delle voci di menu.

la linea guida 1.4.3 individua dei rapporti consigliati di contrasto tra colore di sfondo e colore del testo per garantire buona leggibilità dello stesso. L'aderenza alla linea guida era stata progettata in anticipo, in corrispondenza dell'attività di progettazione dello schema di colori, come illustrato in sezione 6.5, e la conformità a livello AA per persone con corretta percezione dei colori e per persone affette da discromatopsia è verificata tramite lo strumento ColorChecker descritto in sezione 6.2.

Il testo può essere ridimensionato a piacimento da parte dall'utente, per rispettare la linea guida 1.4.4, e grazie al layout fluido l'ingradimento della pagina non causa perdita di funzionalità, al contrario scala molto bene, permettendo al prodotto di essere perfettamente fruibile anche al 200% di zoom, come raccomandato. Questo è verificabile con le funzioni di ingrandimento caratteri del browser.

7.3.4.4 Guideline 2.4: *Navigable*

La linea guida 2.4 prevede che l'utente sia facilitato nel navigare tra i contenuti, trovare i contenuti stessi e determinare la sua posizione.

Questo si è ottenuto nel prodotto tramite il rispetto della linea guida 2.4.2, ovvero fornendo un titolo esplicativo ad ogni pagina, rendendo esplicativi i link e le voci di menu (linea guida 2.4.4) ed utilizzando nel modo e ordine corretto le intestazioni di pagina (linea guida 2.4.6). Si è inoltre aggiunto un link per il ritorno all'inizio della pagina dopo lunghi blocchi di contenuti: questa non è pratica comune nelle applicazioni per dispositivi touchscreen, ma si ritiene possa favorire l'accessibilità dell'applicazione per persone che utilizzino sistemi di lettura dello schermo del dispositivo e per persone con difficoltà motorie, per i quali lo scroll della pagina tramite dita potrebbe essere difficoltosa.

7.3.4.5 Conclusioni

Quanto elencato precedentemente corrisponde alla riflessione e verifica di accessibilità svolta durante e al termine della codifica del prodotto. Si è giustificato il rispetto delle linee guida in tutti gli scenari e per tutte le linee guida giudicate applicabili sia al contenuto che allo scopo dell'applicazione, cercando di rispettare a livello AA quanto prescritto dal WCAG.

Si certifica inoltre, visto che non può esserci accessibilità senza aderenza agli standard, che il markup HTML e CSS risulta valido in ogni sua parte secondo il validatore per HTML5 e CSS3 fornito dal W3C.

8 Compilazione e pubblicazione

La tecnologia scelta per ottenere un supporto multipiattaforma, Apache Cordova (descritta in sezione 6.2), prevede che la *web application* prodotta sia poi preparata per la compilazione per ogni piattaforma desiderata.
La configurazione per le varie piattaforme è unica, ed avviene attraverso un opportuno file XML che permette di specificare nome dell'applicazione, versione, icona da utilizzare, permessi richiesti e domini da inserire nella *whitelist*, in modo che non si applichino restrizioni CORS (Cross Origin Resource Sharing). È il sistema di build ad occuparsi poi di generare le corrette configurazioni nel codice specifico generato per ogni piattaforma.

Per la preparazione del codice specifico per ogni piattaforma a partire dall'applicazione web, Cordova mette a disposizione uno strumento denominato Cordova Command Line Interface. Questo richiede che gli eseguibili specifici per gli SDK di ogni piattaforma bersaglio siano presenti nel *PATH* del terminale, e permette con un comando la preparazione del codice.
Dopo aver eseguito una prima volta il comando di aggiunta delle piattaforme desiderate, ad esempio per Android il comando:

cordova platform add android

è sufficiente il comando

cordova prepare

per la preparazione del codice specifico per ogni piattaforma. Lo stesso può essere compilato per ogni piattaforma in un pacchetto pronto per l'installazione su dispositivi mobili tramite il comando

cordova build

oppure, più convenientemente, se è necessario effettuare modifiche o accreditare l'applicativo con i file di licenza che ne permettano l'installazione e la distribuzione, aprire il codice prodotto con gli strumenti propri di ogni SDK e realizzare manualmente la compilazione del codice che però, grazie a Cordova, non richiede modifiche, se non sono richieste particolari personalizzazioni.

Per preparare il codice sorgente dell'applicazione, da compilarsi successivamente tramite gli strumenti propri di ogni piattaforma, è necessario che sul computer utilizzato sia installato l'SDK della piattaforma bersaglio. Questo porta ad alcune restrizioni, in quanto è necessario un computer Macintosh con il sistema

operativo iOS in esecuzione per poter scaricare l'SDK rivolto a dispositivi con iOS, mentre per l'SDK destinato a Windows Phone 7.5 e 8 è necessario sia in esecuzione il sistema operativo Windows 8 a 64 bit.

Per una compilazione manuale del prodotto per ogni piattaforma bersaglio, è necessario quindi disporre di almeno due sistemi operativi (fortunatamente, l'SDK di Android è disponibile sia per iOS che per Windows 8), oltre che delle chiavi di accreditamento per la pubblicazione dell'applicazione.

La macchina utilizzata durante lo sviluppo eseguiva il sistema operativo ArchLinux, pregiudicando quindi la compilazione per iOS e Windows Phone. Si è quindi proceduto a build di debug più frequenti per la piattaforma Android, sfruttando macchine con in esecuzione iOS e Windows 8 per la validazione del prodotto in corrispondenza di incrementi degni di nota nella stesura del codice. Differenti accortezze si sono adottate per le build di rilascio del prodotto.

8.1 Il sistema di *override*

Nel caso sia necessario del codice specifico per una o più piattaforme, invece di utilizzare all'interno dell'applicativo degli strumenti di rilevamento del dispositivo e, successivamente, delle istruzioni condizionali per le diverse piattaforme, soluzione che aumenterebbe inutilmente la complessità dell'applicazione e ne ridurrebbe la manutenibilità e la robustezza futura (non ci sono infatti garanzie che un dispositivo di un dato sistema operativo si dichiari come atteso), Cordova fornisce il sistema denominato di *override*.
Gli strumenti di creazione di progetto di Cordova realizzano una cartella chiamata *merges* nella cartella radice del progetto, che all'aggiunta di piattaforme al progetto si popola di cartelle omonime (Android, iOS...). I file presenti all'interno delle cartelle saranno copiati all'interno del codice sorgente prima della preparazione dell'applicazione per la data piattaforma, sovrascrivendo eventualmente file omonimi.

Questo ci permette di collocare eventuale codice specifico per una data piattaforma all'interno della sua cartella presente in *overrides*, e sarà lo strumento a linea di comando a provvedere alla copia di questi file prima delle operazioni di preparazione.

Il sistema di *override* è stato utilizzato nel progetto, ad esempio, per visualizzare o nascondere il back button visualizzato nella regione di navigazione dell'applicazione: il *back button* è presente in tutte le pagine, ma è nascosto da un'opportuna regola CSS collocata in un file CSS separato.
Per iOS, per il quale si vuole visualizzare il pulsante, è stato creato nella cartella *merges/ios/css* un file CSS che sovrascrivesse la regola di *visibility:hidden*, impostandola a *visible*. In questo modo si è evitato l'inserimento nel codice di

istruzioni condizionali che modificassero la visibilità dell'elemento a seconda della piattaforma rilevata.

8.2 Compilazione dell'applicazione

Si illustrano ora le metodologie per la compilazione dell'applicazione nell'ottica del suo rilascio al pubblico e dei suoi futuri aggiornamenti.

8.2.1 Build manuali

Una strategia di compilazione manuale dell'applicazione per le tre piattaforme prevede la preparazione dell'ambiente di compilazione (tramite l'installazione di Node.js, Cordova e dell'SDK desiderato) in almeno due sistemi operativi, iOS e Windows 8 64bit. Dopodiché, futuri aggiornamenti del prodotto che richiedano una nuova pubblicazione richiederebbero l'aggiornamento del codice, effettuabile tramite sistema di versionamento e GitHub, *repository* scelto per il progetto, l'esecuzione del comando di preparazione e la compilazione manuale finale firmando i pacchetti ottenuti con le diverse chiavi per sviluppatori, una per Android, una per iOS e una per Windows Phone.

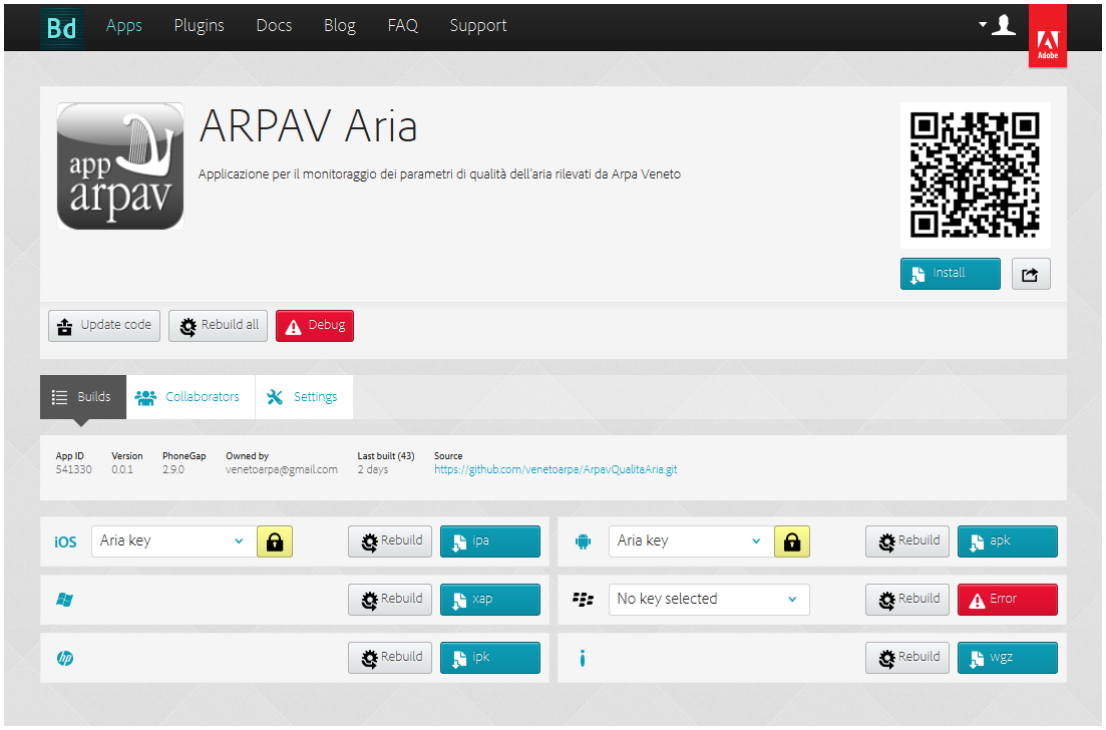

Figura 8.1: Sistema PhoneGap Build, riepilogo applicazione

8.2.2 Build automatiche tramite PhoneGap Build

Per rendere più comodo il processo di compilazione manuale descritto precedentemente, Adobe ha realizzato un sistema chiamato PhoneGap Build, gratuito per

progetti opensource ospitati su GitHub, che offre il recupero del codice sorgente dal repository e la compilazione e firma delle applicazioni per le piattaforme desiderate presso i loro server.
Il sistema supporta le ultime versioni di Cordova, ha recentemente introdotto il supporto a *merge/override* descritti prima e permette il caricamento delle chiavi di firma delle applicazioni, in modo da realizzare pacchetti già pronti per la pubblicazione presso i negozi di applicazioni dei diversi ecosistemi. Offre inoltre, per un più comodo debug direttamente sui dispositivi di test, una console remota basata sul software Weinre, dal funzionamento simile a quello della console JavaScript di Chrome/Chromium o Firebug, ma capace di eseguire remotamente sull'istanza dell'applicazione in esecuzione sul dispositivo.

Si è deciso di provare il sistema, mostrato di seguito in figura 8.1 nell'ottica di migliorare il macchinoso sistema manuale, che richiede l'accesso a macchine di diversa tipologia e con diversi sistemi operativi installati.
Il sistema è stato provato realizzando i pacchetti per Android e iOS, installandoli su dispositivi di test e verificando il buon funzionamento del prodotto, indice di compilazione andata a buon fine.
Una prima configurazione di chiavi di firma per le applicazioni e permessi richiesti all'utente dall'applicazione ha permesso di ottenere lo stesso risultato delle build manuali, ed il sistema risulta quindi utilizzabile per tutti i futuri aggiornamenti del prodotto, compreso l'eventuale debug grazie all'installazione fornita della console Weinre.

9 Conclusioni

Al termine del progetto di stage, e quindi al termine di circa 300 ore di analisi di problemi e di esigenze utente, studio del dominio, progettazione di un sistema che possa rispondere al meglio alle richieste, studio (e a volte scontro) di nuove tecnologie e strumenti, è possibile osservare quanto realizzato, sia in termini di prodotti che di processi, ed analizzarne il rapporto con i tre anni accademici di studio precedenti all'esperienza svolta.
È quindi possibile un'analisi immediata, basata sui risultati ottenuti e sui raggiungimenti degli obiettivi, ed una riflessione su quanto si è imparato di nuovo e quanto si è applicato delle conoscenze acquisite durante i corsi del percorso di laurea triennale.

9.1 Obiettivi raggiunti e risultati

Gli obiettivi del progetto di stage sono stati raggiunti in tutte le componenti obbligatorie, con buona soddisfazione del committente.

In particolare, si desiderava non solo la realizzazione di un'applicazione multipiattaforma, ma al tempo stesso la ricerca e l'eventuale dimostrazione che si possano realizzare applicazioni multipiattaforma per i diversi sistemi operativi mobile presenti sul mercato. Questo è stato dimostrato possibile con buon successo, visto il funzionamento dell'applicazione su tutte le piattaforme bersaglio e l'unica base di codice realizzata, ed è pienamente in linea con la *mission* del progetto appARPAV, che si propone di raggiungere quanti più utenti possibile.
È quindi possibile, per futuri progetti che vogliano ingrandire il parco applicazioni di ARPAV, sfruttare le basi poste (sia in termini di conoscenza, di studio e scelta delle tecnologie, ed eventualmente di moduli di codice) dal progetto, e ritengo che questo sia indice del successo dell'esperienza di stage.

La richiesta aggiuntiva, in corso d'opera, di realizzazione di un sistema server preposto all'esportazione dei dati di qualità dell'aria, è stata affrontata bene e con successo, eccedendo forse le aspettative del committente. Si è infatti realizzato un sistema estendibile per l'esportazione di qualsiasi tipo di dato ambientale misurato dalle stazioni ARPAV in vari formati specificati tramite plugin.
Questo ha superato le esigenze applicative specifiche del progetto di stage, ma lascia all'azienda qualcosa di concretamente utilizzabile, e ben documentato, per

applicazioni future che richiedano altri tipi di dati ambientali.

L'applicazione è stata pubblicata su Play Store, per i dispositivi Android, ed attende revisione e pubblicazione nell'App Store, per i dispositivi iOS, mentre una futura pubblicazione per Windows Phone è subordinata all'iscrizione alla rete di sviluppatori Microsoft da parte di ARPAV. Il progetto di stage ha quindi realizzato l'obiettivo di sviluppo multipiattaforma, obiettivo primario del progetto, ed ha aggiunto una nuova prospettiva allo sviluppo di applicazioni mobile che esulasse dalla programmazione nativa per dispositivi Android o Apple.

L'approccio alla progettazione di applicazioni considerando al contempo l'accessibilità delle stesse, facilitato dalla natura ibrida delle applicazioni web, è una modalità poco percorsa finora, e credo possa aver lasciato riflessioni positive presso l'azienda (individuando ad esempio anche alcune criticità del portale web). Lo stesso vale per lo studio dei formati di trasmissione dei dati, mostrando come si possano ottenere grandi miglioramenti in termini di dimensioni con la razionalizzazione degli schemi XML.

9.2 Conoscenze acquisite

Al termine dell'esperienza di stage, si può dire di aver approfondito ed acquisito competenze con nuovi strumenti e tecnologie, qui riassunti:

- **Apache Cordova / PhoneGap:** lo strumento principale per la realizzazione dell'applicazione. Era lo strumento meno conosciuto, sul quale si riponevano le maggiori aspettative, e che si è rivelato alla fine poco più di un browser con accesso all'hardware. Si può dire di averne compreso bene il funzionamento, che lo rende attualmente uno dei migliori progetti per scrivere applicazioni web quanto più agnostiche possibili da librerie e tecnologie terze (si sfruttano infatti solo API standard definite dal W3C), sebbene secondo me sia un progetto che può maturare ancora, specialmente nel comparto di installazione, realizzazione di scheletri di applicazioni e strumenti per lo sviluppo;

- **JavaScript:** l'utilizzo di JavaScript come linguaggio principale per la realizzazione della logica dell'applicazione era un'esperienza del tutto nuova, ed è stato un approfondimento molto apprezzato, specialmente nello studio dei diversi framework che ne estendono le possibilità. Avendolo conosciuto solo come linguaggio di scripting lato client per pagine web, è stato arricchente il suo utilizzo in un ambito del tutto nuovo, e ha lasciato un forte desiderio di approfondimento, sia per realizzare applicazioni che lato server grazie al progetto Node.js;

- **Strumenti di verifica:** Si è acquisita familiarità con un grande numero di strumenti di verifica, tramite analisi statica e test, visto l'ampio spettro delle tecnologie utilizzate: gli strumenti di analisi per JavaScript, i browser

headless come Phantom.js ma anche gli emulatori di dispositivi mobili, oltre che gli strumenti di analisi dei browser e gli strumenti per database Oracle;

- **Database Oracle:** la sintassi di interrogazione per i database Oracle e la loro struttura era relativamente familiare perché simile al conosciuto MySql, ma ha comunque portato ad approfondimento nelle differenze con le basi di dati conosciute ed ha lasciato l'esperienza di interazione con una banca dati molto grande, con milioni di record, e di approfondimento delle problematiche legate alla grande mole di dati;

9.3 Rapporto con la preparazione accademica

Durante lo svolgimento dello stage si sono potute applicare le conoscenze già acquisite per la gestione di progetto e per il miglior sviluppo possibile dell'applicazione, potendo provare sul campo l'importanza dell'analisi dei requisiti e della validazione, i benefici di una buona ed attenta progettazione dell'architettura e di una strategia formale ed estensiva di test del prodotto.

Sul piano di realizzazione del prodotto e dell'interfaccia utente, si sono potute applicare le conoscenze acquisite sullo sviluppo di interfacce web fluide, sui percorsi utente e, soprattutto, sull'accessibilità di prodotto e sull'importanza del corretto uso degli standard web, preferendo sempre la semantica alla presentazione.

In generale, credo che la preparazione accademica precedente all'attività di stage sia stata adeguata perché potessi affrontare al meglio il progetto, non tanto nelle conoscenze fornite quanto piuttosto nel metodo insegnato durante tutti i corsi del percorso di laurea triennale. Ho potuto informarmi ed imparare tutte le nuove tecnologie necessarie per svolgere il progetto autonomamente, grazie alle basi culturali fornite che mi hanno permesso di poter comprendere nuovi sistemi nel modo migliore. Credo questo, l'avere gli strumenti per la comprensione autonoma di nuove tecnologie e nuovi approcci a quanto studiato, sia il bagaglio più importante lasciato dal percorso di laurea triennale, più che lo studio di specifiche tecnologie e dettagli, che sono comunque strumenti per la crescita personale.

Ho potuto provare di persona tutti i processi necessari allo svolgimento di un progetto che dovesse comprendere tutte le attività, dalle prime analisi fino alla manutenzione finale del prodotto in opera, affrontando di nuovo l'attività di progettazione iniziale che si è confermata di massima importanza, ben più della codifica. Il prodotto stesso è stato poi sviluppato con una strategia chiara e formale di sviluppo per componenti incrementali da integrare e relativi test, e documentato in maniera chiara in ogni processo effettuato, dall'analisi alla codifica, per garantire il miglior esito possibile al progetto.

A | Bibliografia

Risorse riguardanti JavaScript

- Osmani, A. *Developing Backbone.js Applications* (O'Reilly Media, 2013). Disponibile anche online presso http://addyosmani.github.io/backbone-fundamentals/
- Osmani, A. *Architectural example of a Backbone to-do app*, consultato a luglio 2013 presso http://todomvc.com/architecture-examples/backbone/
- Osmani, A. *Learning JavaScript Design Patterns* (O'Reilly Media, 2013). Disponibile anche online presso https://github.com/addyosmani/essential-js-design-patterns
- Fogus, M. *Functional JavaScript: introducing Functional Programming with Underscore.js*, sezioni 2,3,8,9 (O'Reilly Media, 2013)
- DocumentCloud. *Annotated Backbone.js sources*, consultato a luglio 2013 presso http://backbonejs.org/
- Booth, P. *About code complexity in JavaScript*, consultato a giugno 2013 presso http://jscomplexity.org/complexity
- Butler, D. *Unit test JavaScript applications with Jasmine*, consultato a giugno 2013 presso http://www.adobe.com/devnet/html5/articles/unit-test-javascript-applications-with-jasmine.html

Risorse riguardanti HTML e accessibilità

- Cooper, Reid, Kirkpatrick, O Connor, Vanderheiden. *Understanding WCAG 2.0*, versione denominata *W3C Working Group Note 5 September 2013* disponibile presso http://www.w3.org/TR/2013/NOTE-UNDERSTANDING-WCAG20-20130905

Altre risorse

- Beta studio, s.r.l. *Specifiche tecniche del servizio di trasformazione di coordinate* (Ministero dell'ambiente e della tutela del territorio e del mare, rev. 2013-03-08). Disponibile presso http://www.pcn.minambiente.it/GN/leggi/MATTM-MU_SRC_PSCC_SPEC_TECNICHE-001-1.pdf