1506
UNIVERSITÀ
DEGLI STUDI
DI URBINO
CARLO BO

Corso di Laurea in Informatica Applicata

Programmazione e Modellazione a Oggetti – *Progetto a.a. 2019/2020*

"Fuel Search – Prezzi Carburanti"

Sviluppatori:

Nicolò Santini, mat. 292404

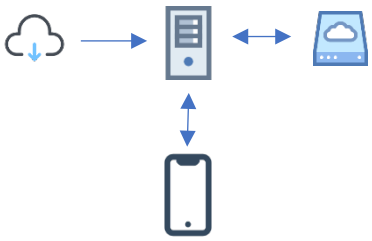
INDICE

PARTE PRIMA – Specifica del Software

Sviluppo dell'app "*Fuel Search – Prezzi Carburanti*" secondo il paradigma di OOP, utilizzando il linguaggio C# con il tool Xamarin per il supporto alla multiprogrammazione e alla futura eventuale distribuzione su sistemi operativi differenti da Android. L'app permette di risparmiare tempo e denaro cercando tra tutti i distributori di carburanti nella zona specificata quali sono i migliori. I dati utilizzati sono aperti e reperibili a questi indirizzi:

- Anagrafica Impianti: https://www.mise.gov.it/images/exportCSV/anagrafica_impianti_attivi.csv
- Rilevazioni: https://www.mise.gov.it/images/exportCSV/prezzo_alle_8.csv

PARTE SECONDA – Studio del Problema



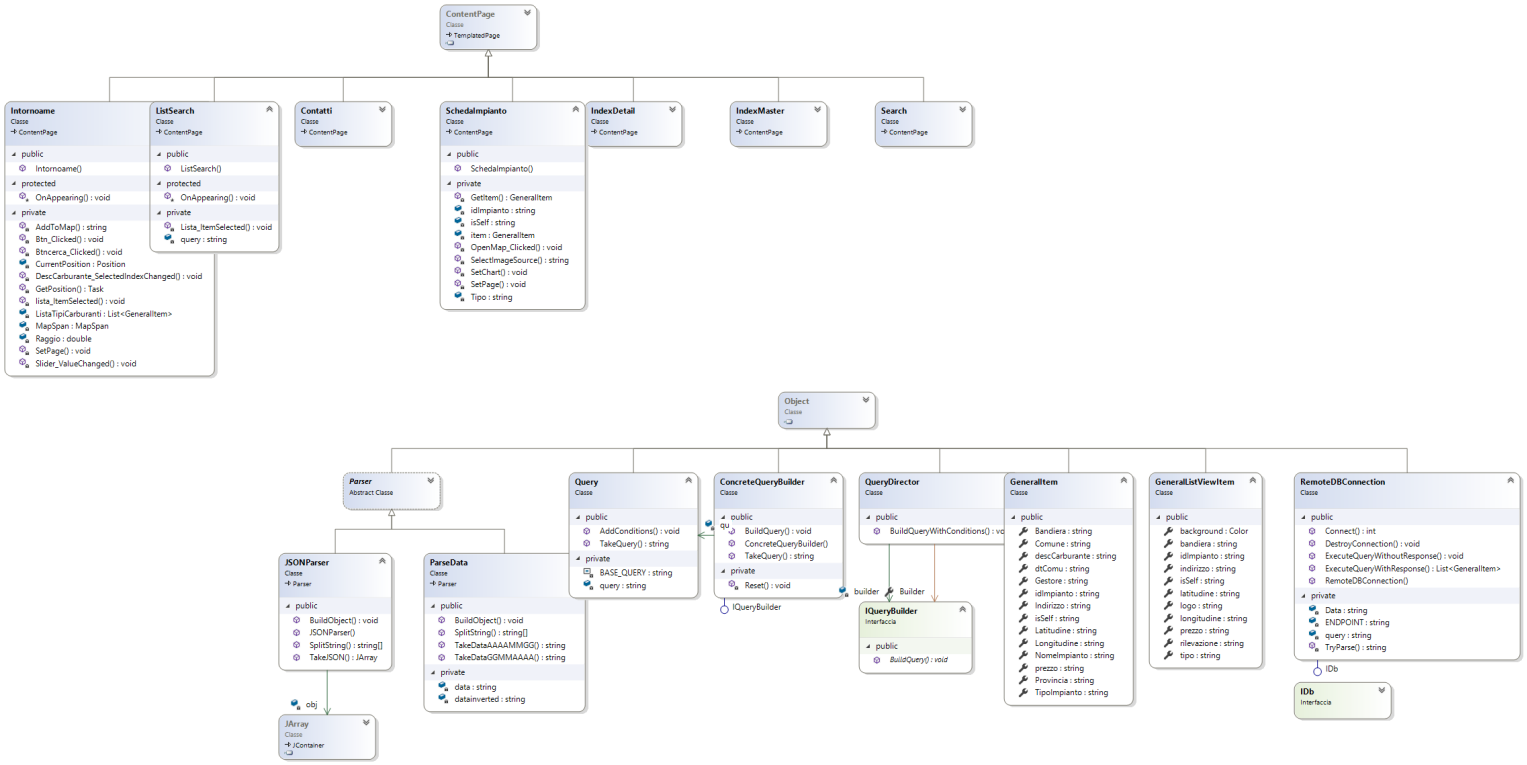
1. **Infrastruttura alla base del funzionamento:**
   Di fondamentale importanza è la scelta infrastrutturale che consente il funzionamento dell'app: i dati vengono scaricati dagli URL precedentemente nominati. Entrambi i file sono in formato .csv, i valori pertanto sono separati da virgole tra loro e alla fine di ogni record troviamo un carattere di terminazione della riga.
   I dati vengono scaricati sotto forma di stringhe da uno script php hostato in un server remoto. Un altro script si occupa di parsare le stringhe e interagire con il database (MySQL remoto) per l'inserimento dati nell'opportuna tabella. Il server fornisce inoltre una API che consente la comunicazione sicura con il database (anziché comunicare direttamente con il DB, viene fornito un endpoint che richiede una query come parametro). Il dispositivo su cui esegue l'applicazione si limita ad elaborare i dati che i vari endpoint ritornano.
2. **Organizzazione dati:**
   Visto che i dati sono organizzati in maniera tale da avere un file con la lista delle anagrafiche degli impianti e l'altro file con la lista delle rilevazioni, è necessario che una tabella contenga record aventi sia la rilevazione, sia l'anagrafica associata. Ciò garantisce una più semplice gestione dei dati da parte dell'applicazione.

A tal proposito l'applicazione ruota attorno ad una classe che contiene ha come attributi i vari campi della tabella contenente sia l'anagrafica che le rilevazioni. Importante è anche la classe che parsa i dati da JSON (formato dati ritornato dalle API quando interrogate) in stringhe e li inserisce nell'attributo corretto dell'oggetto.

PARTE TERZA – Scelte Architetturali

L'applicazione ha una netta distinzioni tra le classi che gestiscono la parte grafica nelle pagine XAML e classi che sono alla base degli oggetti che popolano le pagine.

Pertanto, andando ad analizzare l'UML del progetto, è possibile trovare due tipologie di classi: le classi che si occupano della parte grafica che derivano dalla superclasse ContentPage, mentre le altre classi che derivano direttamente da Object.



I punti cruciali dell'applicazione sono:

- **Creazione di un'interfaccia IDb:**

  Tale interfaccia fornisce 4 metodi per la comunicazione di base con un Database. Nel progetto viene implementata solamente dalla classe RemoteDBConnection, che permette di interrogare l'API in php passando come indirizzo la query da eseguire sul database remoto, permettendo all'app di interrogarlo indirettamente. L'interfaccia è stata inserita nel progetto con lo scopo di interrogare un eventuale database locale per effettuare il caching dei dati (velocizzando i tempi di reazione dell'app) in futuro.

- **Uso del Builder Pattern:**

  Poiché il funzionamento dell'applicazione si basa sulla continua interrogazione al database remoto per ottenere informazioni sui distributori da mostrare sullo schermo, si è reso necessario l'utilizzo del Builder Pattern per automatizzare la costruzione delle query che permettono di dialogare con il database. In

particolare, in alcune pagine dell'applicazione, come ad esempio "Search.xaml", era necessario costruire $2^5 = 32$ query diverse, a seconda che l'utente scegliesse uno piuttosto che l'altro filtro.
Grazie al builder pattern viene creato un oggetto query, che ha una base comune e viene popolato di condizioni a seconda che i vari picker presenti nella pagina siano o meno selezionati.

Degna di nota è anche la parte back-end: sostanzialmente il tutto è basato su delle classi in php: in particolare una classe astratta "Parser", utilizzata da "ParseAnagrafica" e "ParseRilevazioni" che in base al contenuto letto dal file agli indirizzi forniti nella PARTE PRIMA della relazione, crea nuovi oggetti "Anagrafica" e "Rilevazioni". Tali oggetti, memorizzati in degli array, vengono successivamente inseriti all'interno del DB remoto mediante il Task chiamato "DownloadAndFillDB.php" riempie periodicamente tutte le tabelle del DB e elimina i dati più vecchi di 15 giorni.
Tale task viene eseguito ogni 6 ore dal sistema operativo in cui è hostato il server.

PARTE QUARTA – Use Cases con UML

Use Case: Caricamento Home Page
Id: UC1
Actor: A1 User, A2 Db Remoto
Preconditions: • App appena avviata • Nessun dato è ancora presente nel dispositivo
Basic course of events: 1. L'app viene avviata da A1 2. Vengono create 8 query rappresentanti gli 8 valori presenti nella Home Page 3. Viene contattato il A2 4. Vengono ritornati i valori corrispondenti
Postconditions: • I dati tornati vengono mostrati nella schermata • La mappa viene popolata di pin nei punti in cui i distributori forniti si trovano
Alternative paths: • Se 4. Non ritorna valori sufficienti, viene mostrato un messaggio di errore

Use Case: Pagina Ricerca Impianti
Id: UC2
Actor: A1 User, A2 Db Remoto
Preconditions: • A1 si sposta, tramite il menu di selezione, nella pagina “Ricerca Impianti”
Basic course of events: 1. Creata query che ritorna i valori di tutte le provincie 2. Creata query che ritorna i valori del tipo carburante 3. Creata query che ritorna i nomi della compagnia 4. Contattato A2, che ritorna valori 5. Picker riempiti con valori ritornati 6. A1 seleziona valore da Picker Provincia 7. Creata query che ritorna i valori di comuni appartenenti a provincia selezionata 8. Contattato A2, che ritorna valori 9. Picker comune rimepito con valori ritornati 10. A1 seleziona uno o più picker, filtrando i risultati a suo piacimento 11. Viene creata la query per interagire con A2, inserendo condizioni a seconda dei picker selezionati 12. A1 seleziona il bottone di ricerca
Postconditions: • Viene aperta una nuova pagina con la lista dei distributori trovati con i filtri selezionati • Vengono mostrati in cima prezzo migliore e prezzo peggiore
Alternative paths: • Se 10. Non va a buon fine (A1 non seleziona nulla dal picker provincia), viene comunicato un errore

Use Case: Apertura Scheda Impianto
Id: UC3
Actor: A1 User, A2 Db Remoto
Preconditions: • A1 si trova nella Home Page o in una pagina con la lista degli impianti
Basic course of events: 1. A1 seleziona un oggetto dalla lista 2. Scheda impianto viene inizializzata con alcuni valori dell’oggetto selezionato 3. Creata query per ritornare tutti i valori dell’impianto selezionato 4. Contattato A2, che ritorna valori
Postconditions: • Dati anagrafici ordinati e mostrati a video • Dati su storico rilevazioni ordinati e mostrati nel grafico
Alternative paths: -

PARTE QUINTA – Link

Fuel Search è disponibile gratuitamente per tutti i dispositivi Android nel Google Play Store a questo link:

https://play.google.com/store/apps/details?id=com.FuelSearch.FuelSearch