**Documentazione Progetto  
Officina MeccanicaMilani  
Gianluca Milani**

Esame Laboratorio Web II

**1-Caso d’uso**

1.1 Obiettivi del progetto

1.2 Utenti destinatari

**2-Descrizione del progetto e funzionalità**

2.1 Panoramica delle funzionalità

2.2 Ruoli e permessi

2.2.1 Cliente

2.2.2 Meccanico

2.2.3 Amministratore

2.3 Flussi operativi principali

**3-Architettura del sistema**

3.1 Tipologia di Blazor utilizzata

3.2 Architettura a livelli (Presentation, API, Business, Data)

3.3 Integrazioni esterne (es. invio token reset password)

3.4 Sicurezza del sistema

3.5 Hosting e modalità di deployment

**4-Struttura del codice e organizzazione del progetto**

4.1 Struttura delle cartelle

4.2 Descrizione dei file principali

4.3 Mapping e DTO

**5-Difficoltà affrontate e soluzioni adottate**

5.1 Problemi tecnici rilevanti

5.2 Strategie di risoluzione

**6-Possibili estensioni future**

6.1 Funzionalità aggiuntive pianificate

6.2 Integrazioni tecnologiche potenziali

6.3 Adattamenti multi-officina o multi-tenant

**7-Diagrammi di flusso e architettura**

5.1 Diagramma dei livelli

5.2 Diagrammi di flusso per ruolo

5.2.1 Cliente

5.2.2 Meccanico

5.2.3 Admin

5.2.4 Login e autenticazione

**1. Caso d’uso**

Officina Meccanica Milani è un’applicazione web progettata per aiutare le officine meccaniche a gestire in modo più semplice, veloce e digitale tutte le attività legate ai veicoli dei clienti: dalla prenotazione degli appuntamenti alla gestione dei servizi offerti, dei ricambi, dei preventivi e delle riparazioni effettuate.

Ogni funzione è studiata per ridurre la gestione manuale (es. agende cartacee, fogli Excel o telefonate), centralizzare i dati e permettere a chi lavora nell’officina di avere tutto sotto controllo da un’unica interfaccia.

**1.1 Obiettivi del progetto**

L’applicativo nasce con lo scopo di:

-Ottimizzare la gestione di veicoli, clienti e interventi meccanici.

-Automatizzare le operazioni più ripetitive come la creazione di appuntamenti e preventivi.

-Dare un’interfaccia chiara e personalizzata in base al ruolo (cliente, meccanico o gestore).

-Rendere tracciabile ogni intervento effettuato, con storico sempre consultabile.

**1.2 Destinatari**

Il sistema è pensato per officine meccaniche che si occupano di:

-Revisione periodica dei veicoli

-Riparazioni meccaniche e interventi tecnici

-Manutenzione generale di automobili e altri mezzi

Il sito web si adatta bene a officine di piccole o medie dimensioni che vogliono digitalizzare i processi senza dover usare software complicati o costosi.

**2. Descrizione del progetto e funzionalità**

**2.1 Panoramica delle funzionalità**

Il progetto prevede una schermata di login, accessibile a tutti gli utenti. In base al ruolo con cui ci si autentica, vengono mostrate home page e pagine diverse. È anche possibile reimpostare la password in caso di smarrimento tramite un link dedicato.

Oltre alla parte gestionale, l’applicazione comprende anche un sito web pubblico, visibile senza registrazione, dove sono presenti pagine come:

**-Homepage**

**-Chi siamo**

**-Contatti**

**-Servizi offerti**

Questo permette ai clienti di conoscere l’officina, visualizzare i servizi disponibili e trovare facilmente le informazioni di contatto.

Nel sistema sono gestiti tre ruoli principali: Cliente, Meccanico e Amministratore, ognuno con funzionalità specifiche.

Tutte le operazioni presenti nell’applicazione sono automatizzate e associate al tipo di utente che accede alla piattaforma, grazie a un sistema di autenticazione e autorizzazione basato su ruoli.

**2.2 Ruoli e permessi**

**2.2.1 Cliente**

Il cliente è l’utente che porta il proprio veicolo in officina per farlo revisionare, riparare o sottoporre a manutenzione. Può:

**-Registrare uno o più veicoli** con marca, modello e targa.

**-Prenotare un appuntamento** scegliendo giorno, ora e tipo di servizio.

**-Visualizzare lo storico** degli appuntamenti passati e futuri.

**-Visualizzare i preventivi** ricevuti per gli interventi richiesti e prenotarli.

**-Visualizzare le riparazioni** effettuate sul proprio veicolo.

**2.2.2 Meccanico**

Il meccanico è l’operatore dell’officina incaricato di eseguire fisicamente gli interventi sui veicoli. Può:

**-Visualizzare la propria agenda** con gli appuntamenti assegnati.

**-Registrare una nuova riparazione** indicando la descrizione dell’intervento e il costo.

**-Consultare lo storico** degli interventi che ha eseguito.

**-Accedere alle informazioni** dei veicoli sui quali ha lavorato.

**2.2.3 Amministratore**

L’amministratore è la figura che gestisce l’officina a livello operativo e organizzativo. Ha accesso completo al sistema e può:

**-Gestire gli utenti registrati** (visualizzare, modificare, eliminare).

**-Gestire i servizi offerti:** aggiungere, modificare o rimuovere servizi (es. tagliando, revisione, ecc.).

**-Visualizzare statistiche** generali (numero di appuntamenti, veicoli registrati, preventivi generati).

**-Monitorare l’attività** di clienti e meccanici.

**2.3 Flussi operativi principali**

**Prenotazione appuntamento(cliente):**

Il cliente seleziona un veicolo ,sceglie un servizio , il sistema mostra le fasce orarie disponibili , conferma l’appuntamento.

**Generazione preventivo(cliente,meccanico) :**

Viene selezionato un veicolo e un servizio ,si scelgono i ricambi , il sistema calcola il totale e crea un preventivo che verrà confermato o meno dal meccanico

**Registrazione riparazione (meccanico):**

Il meccanico seleziona il veicolo , inserisce descrizione e costo dell’intervento , la riparazione viene salvata nello storico.

**Gestione contenuti (admin):**

L’amministratore accede al pannello di controllo ,può modificare servizi, assegnare i meccanici all’appuntamento o gestire gli account utente.

**3. Architettura del sistema**

**3.1 Tipologia di Blazor utilizzata**

L’applicazione è sviluppata usando Blazor Server, una tecnologia di .NET che permette di costruire interfacce web interattive.

Con Blazor Server, tutta la logica dell’applicazione gira sul server e il browser comunica in tempo reale.

Questa scelta è stata fatta perché:

-non serve scaricare grandi quantità di codice nel browser (più veloce al caricamento),

-la logica rimane sicura e centralizzata sul server,

-è più facile da integrare con Entity Framework per la gestione del database,

-è più adatta per ambienti aziendali come quello di un’officina.

**3.2 Architettura a livelli**

Il progetto segue un’architettura a più strati, per tenere separati i vari aspetti dell’applicazione e renderla più chiara da mantenere.

Ecco i livelli principali:

**-Interfaccia utente (Presentation Layer)**

Tutte le pagine visibili all’utente sono realizzate con Blazor Server usando file .razor.

Le pagine sono divise per ruolo (cliente, meccanico, admin).

I componenti riutilizzabili (menu, layout, ecc.) si trovano nella cartella Shared/.

**-Controller API (API Layer)**

Le chiamate tra l’interfaccia e il server avvengono tramite controller REST.

I controller ricevono le richieste (es. crea veicolo, prendi appuntamenti) e le passano ai servizi.

Si trovano nella cartella Controllers/.

**-Logica applicativa (Business Layer)**

Tutta la logica principale è contenuta nei servizi, ad esempio:

-controllare se un appuntamento è disponibile,

-creare un preventivo,

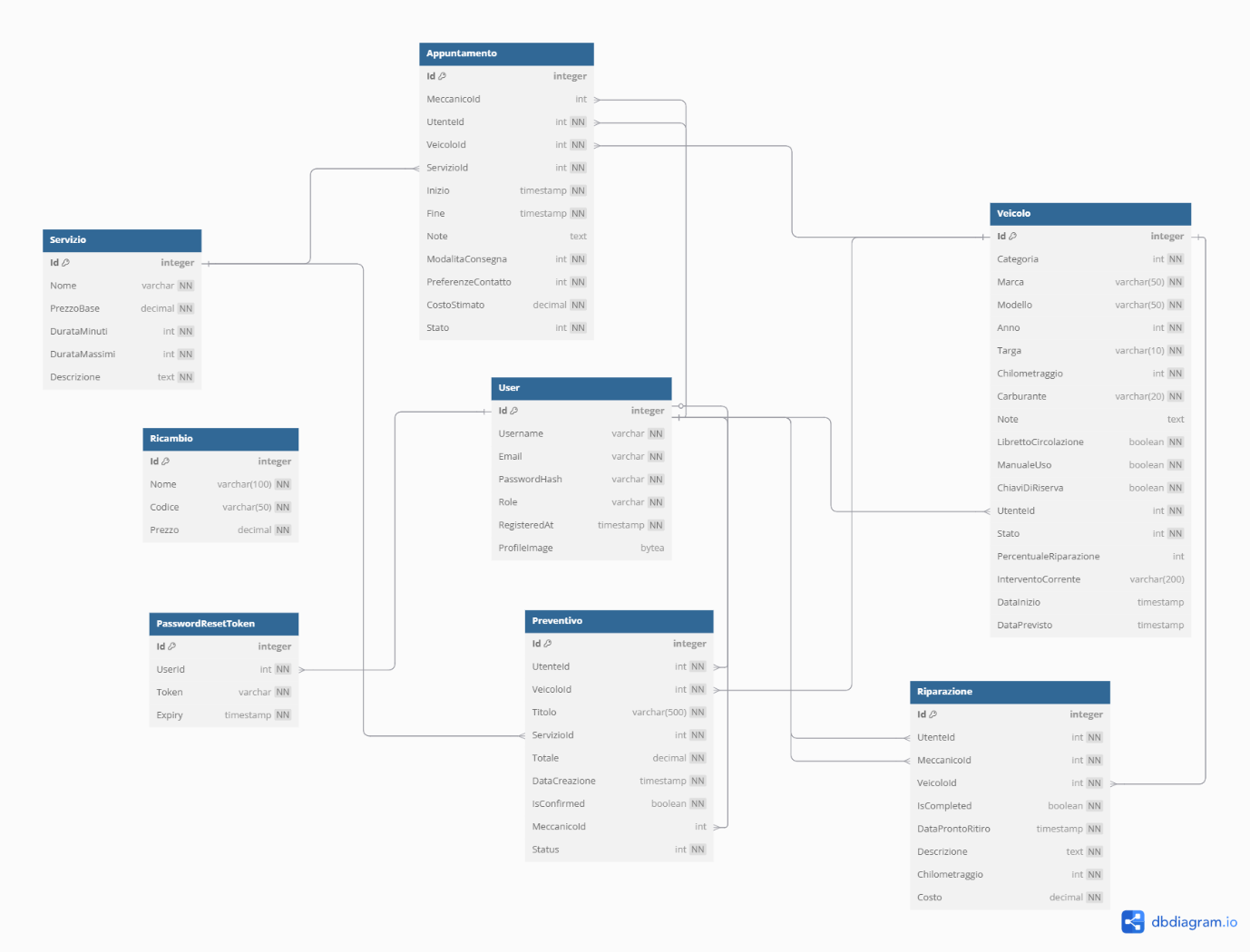
-gestire le riparazioni.

I servizi si trovano nella cartella Services/ e sono separati per ambito (UserService, VeicoloService, ecc.).

**-Accesso ai dati (Data Layer)**

I dati vengono gestiti tramite Entity Framework Core.

Le entità principali sono User, Veicolo, Appuntamento, Servizio, Ricambio, Preventivo, Riparazione.



Il file ApplicationDbContext.cs gestisce tutte le tabelle e le relazioni tra di esse.

In Data/ ci sono anche i DTO (oggetti usati per scambiare dati in modo sicuro tra front-end e back-end).

**3.3 Integrazioni esterne**

Il sistema integra un servizio di recupero password tramite invio email.  
Per questa funzionalità è stato utilizzato **MailTrap**, configurato tramite protocollo **SMTP**, che permette di simulare l’invio delle email in ambiente di sviluppo e test.

L’utente che ha dimenticato la password può richiedere un nuovo token, che viene generato automaticamente, salvato nel database e inviato tramite email di MailTrap. Tramite il link ricevuto, può accedere alla pagina di reset e impostare una nuova password in modo sicuro.

**3.4 Sicurezza del sistema**

Il sistema include diverse misure per proteggere i dati:

**-Autenticazione con cookie:** dopo il login, viene creato un cookie che mantiene la sessione attiva.

**-Ruoli e permessi:** ogni utente ha un ruolo (cliente, meccanico o admin) e vede solo le sezioni abilitate.

**-Controlli lato server e lato client:** anche se un utente prova ad accedere manualmente a una pagina che non gli spetta, viene bloccato.

**-Password criptate:** le password sono salvate nel database in formato hash, quindi non leggibili.

**-Validazioni dei dati:** i dati inseriti dagli utenti vengono controllati per evitare errori o malintenzionati (es. campi obbligatori, email valide, formati corretti).

**3.6 Hosting e modalità di deployment**

Il progetto è stato pubblicato online tramite Azure App Service, che ospita l’intera applicazione Blazor Server.

Il database è collegato a un’istanza di Azure SQL Database, gestita tramite Entity Framework Core e migrazioni automatiche.

La configurazione della connessione e delle chiavi è gestita tramite appsettings.json e variabili di ambiente sul portale Azure.

**4. Struttura del codice e organizzazione del progetto**

L'applicazione è strutturata in modo ordinato, con cartelle separate per ogni area logica del progetto. Questo permette di mantenere il codice chiaro, modulare e facile da gestire.

**4.1 Struttura delle cartelle principali**

**-Pages/ –** Contiene tutte le pagine .razor visibili agli utenti (cliente, meccanico, admin).

**-Shared/ –** Include componenti riutilizzabili e layout generali (es. navbar, layout principale).

Controllers/ – I controller API che gestiscono le richieste REST tra front-end e back-end.

**-Services/ –** Contiene la logica applicativa vera e propria, divisa per ambito (utente, veicolo, appuntamento, ecc.).

**-Models/ –** Include tutte le entità del database (User, Veicolo, Appuntamento, ecc.).

**-Dtos/ –** I DTO (Data Transfer Object) usati per scambiare i dati in modo sicuro tra livelli.

**-Data/ –** Contiene il DbContext, la configurazione del database e le migrazioni EF Core.

**-Mapping/ –** Profili AutoMapper per convertire entità e DTO.

**-wwwroot/ –** File statici come CSS, immagini, script JavaScript.

**4.2 File principali e responsabilità**

Program.cs e Startup.cs – Gestione del ciclo di vita dell’app, configurazione dei servizi e della sicurezza.

**-App.razor –** Componente principale dell’app Blazor, gestisce il routing.

**-ApplicationDbContext.cs –** Il contesto di Entity Framework, con tutti i DbSet<> delle entità.

**-appsettings.json –** File di configurazione (stringhe di connessione, SMTP MailTrap, ecc.).

**-UserService.cs, AppuntamentoService.cs... –** Servizi specifici per ogni area funzionale.

**4.3 Mapping e DTO**

Per evitare di esporre direttamente le entità del database al front-end, vengono utilizzati DTO che contengono solo i dati strettamente necessari.

Le conversioni tra entità e DTO sono gestite tramite AutoMapper, in profili dedicati (es. VeicoloProfile, UserProfile).

**5. Difficoltà affrontate e soluzioni adottate**

**5.1 Problemi tecnici rilevanti**

Durante lo sviluppo, alcune delle difficoltà principali sono state:

**-Gestione delle fasce orarie libere per gli appuntamenti,** evitando sovrapposizioni e doppie prenotazioni.

**-Autenticazione con i cookies basata sui ruoli**, per far sì che ogni utente vedesse solo ciò che gli era permesso.

**-Collegamento tra entità complesse,** come preventivi con lista di ricambi o riparazioni legate sia al veicolo che all’utente.

**5.2 Strategie di risoluzione**

Per affrontare questi problemi sono state adottate le seguenti soluzioni:

-Le fasce orarie vengono generate dinamicamente e confrontate con gli appuntamenti già presenti nel database.

-L’autenticazione è stata implementata con cookie e autorizzazioni basate su Claims, sfruttando il sistema di ruoli.

-I servizi contengono controlli e filtri per caricare solo i dati legati all’utente attivo.

**6. Possibili estensioni future**

**6.1 Funzionalità aggiuntive pianificate**

In futuro si potrebbero aggiungere:

-Download dei preventivi in PDF, per offrire un documento stampabile al cliente.

-Dashboard più avanzata per tutti i meccanici ma soprattutto per l’admin con, grafici per monitorare l’andamento dell’officina (es. guadagni, appuntamenti mensili).

**6.2 Integrazioni tecnologiche potenziali**

-Integrazione con sistemi di pagamento online, per consentire il saldo diretto di interventi e ricambi.

-Collegamento con gestionali fiscali per emettere fatture in automatico.

-Sistema di messaggistica interna tra clienti e meccanici.

**6.3 Adattamenti multi-officina o multi-tenant**

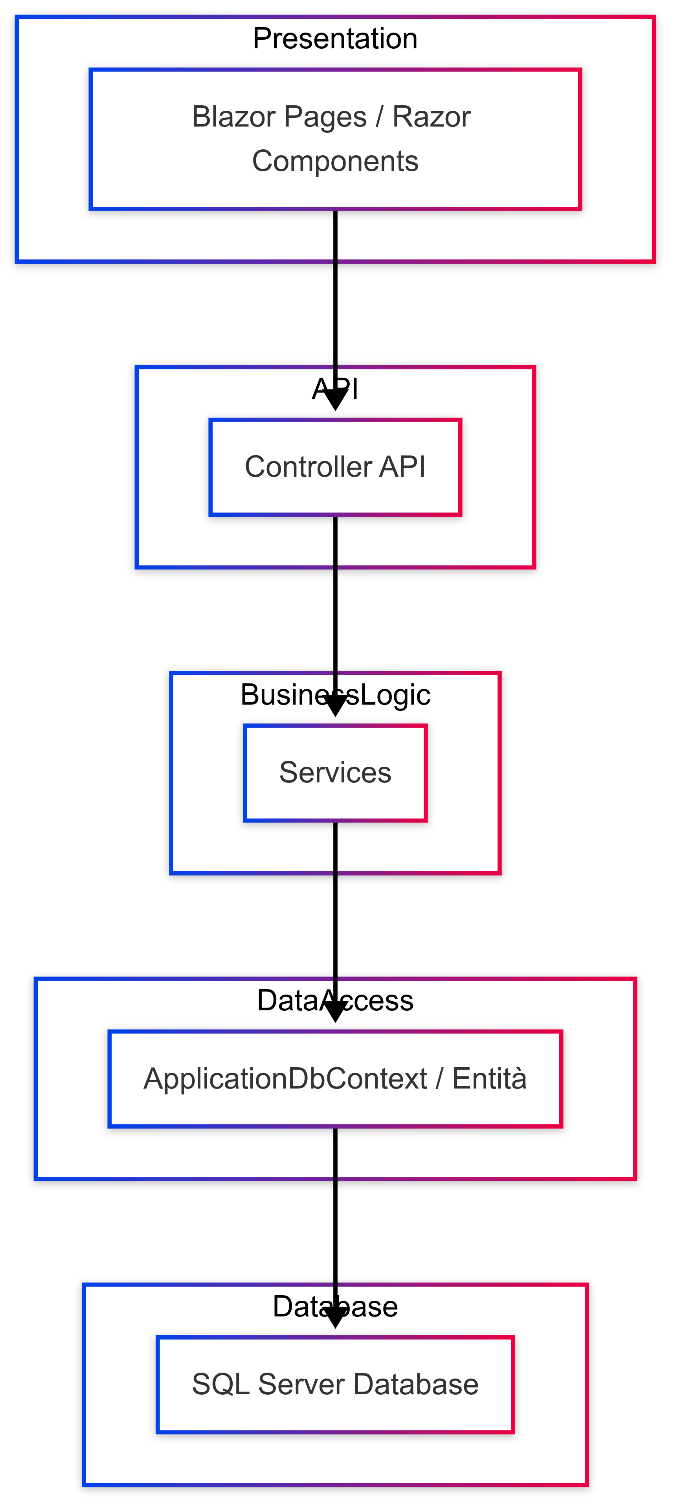
L’applicazione, al momento pensata per una singola officina, potrebbe in futuro essere adattata per:

-Gestire più sedi con utenti, veicoli e personale divisi per sede.

-Configurazioni indipendenti per ogni officina, mantenendo lo stesso codice base ma dati separati.

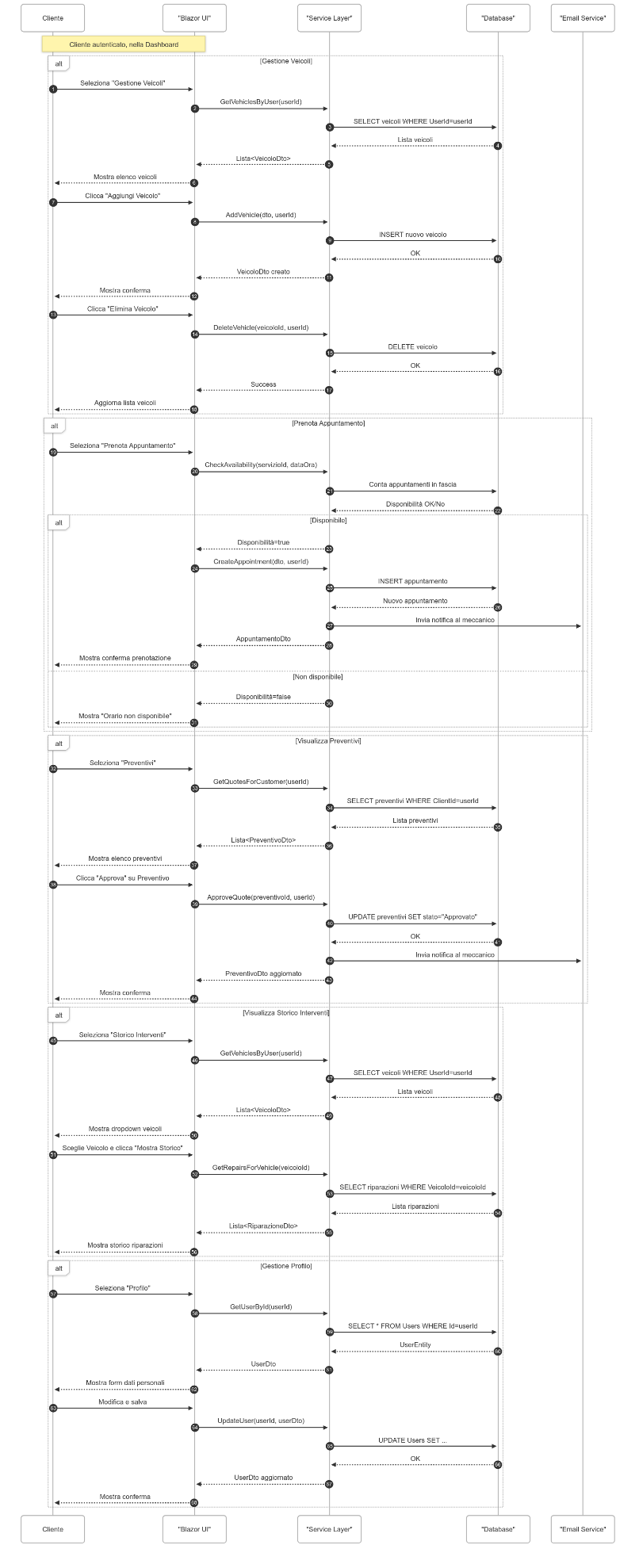
**7-Diagrammi di flusso e architettura**

**5.1 Diagramma dei livelli**

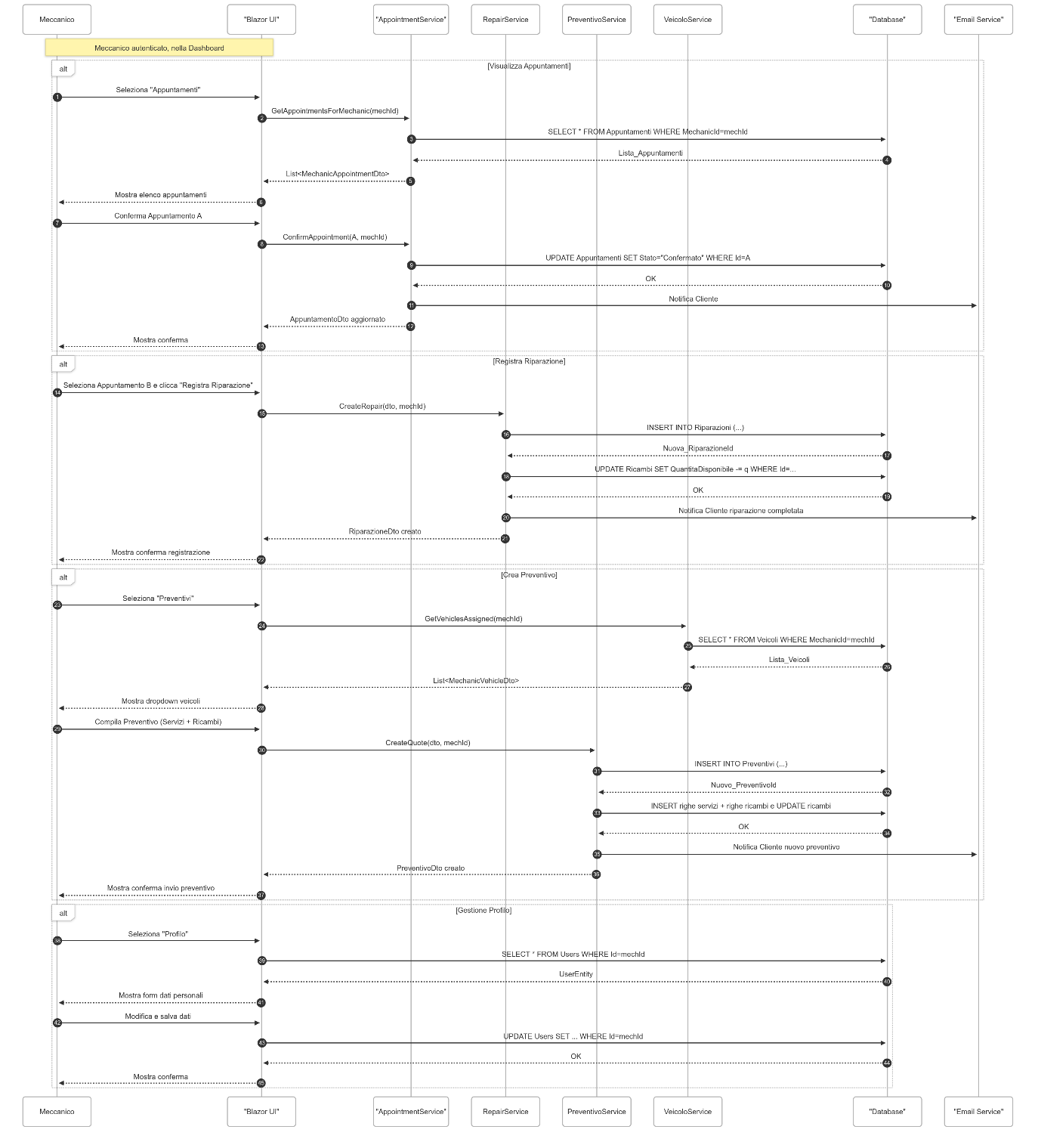
****

**5.2 Diagrammi di flusso per ruolo**

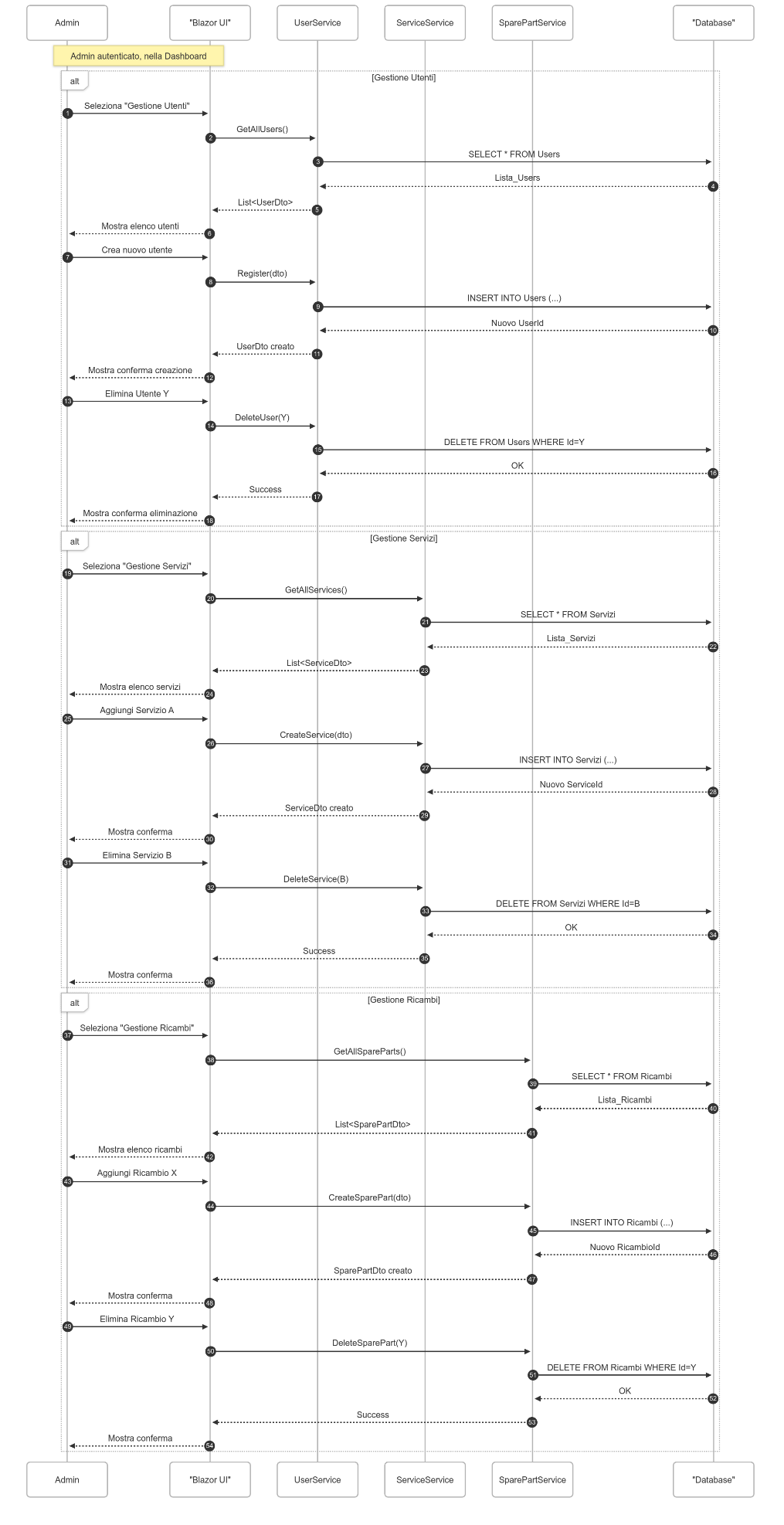
**5.2.1 Cliente**

****

**5.2.2 Meccanico**

****

**5.2.3 Admin**

****

**5.2.4 Login e autenticazione**

