Descrizione del progetto

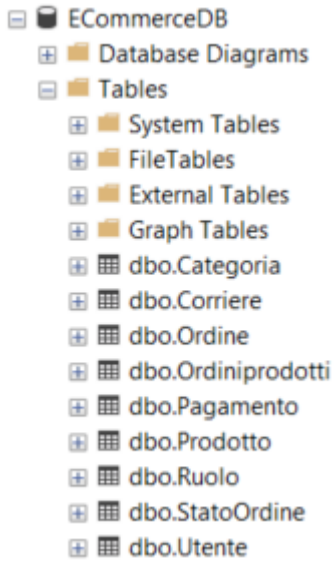
L'obiettivo di questo progetto è l'estensione dell' applicazione sviluppata per la gestione di un negozio online di prodotti tecnologici. Tale estensione prevederá di utilizzare pattern e modalità di sviluppo piú professionali che sono la base per realizzare applicazioni sempre piú orientate alle nuove esigenze di mercato.
Infatti oltre a migliorare quello che è stato fatto nel precedente progetto, verranno affrontati nuove tecnologie quali Microsoft Identity ed utilizzate risorse/funzionalitá del Cloud Azure, quali, per esempio il processo di App Registrations, l'utilizzo del key vault e dello storage account.

Di seguito le specifiche per l'estensione dell'applicazione di gestione di un negozio online di prodotti tecnologici. L'applicazione deve essere sviluppata utilizzando ASP.NET Core 6.0 Rest API,la gestione della autenticazione ed autorizzazione tramite Microsoft Identity. Sará posta ancora maggiore attenzione all'organizzazione del codice in class library, utilizzando anche le interfacce per la definizione dei vari metodi che saranno utilizzati.

A tale proposito verrá fornito uno scheletro di soluzione da cui partire.

Inoltre ci sará un'attenta gestione del logging tramite la libreria Serilog ed una class library scritta per semplificare le operazioni di logging.

Come db (Microsoft Sql server) si partirá da quello giá utilizzato ECommerceDB



Ma dovranno essere aggiunte alcune tabelle per gestire le nuove specifiche applicative che saranno descritte successivamente.

Dovrá essere usato EntityFrameworkCore con l'utilizzo di migrations e dovrá essere fatta la configurazione manuale di EF Core e di tutte le entitá (classi C# che rappresentano il modello dati). (NO AIUTI CON LO SCAFFOLDING)

Il front-end sará sviluppato in Angular (NO ASP.NET MVC).

Un paio di settimane prima di terminare questa attività il database sarà portato sul cloud Azure (servizio PASS); analogamente le applicazioni REST Api e il front-end Angular saranno pubblicate su App Service del cloud Azure.

Quanto prima si dovrà configurare Microsoft Identity e fare la registrazione dell'applicazione di back end Rest Api in App Registrations del cloud Azure.

Uno degli obiettivi di questa esercitazione è quello di leggere i dati di configurazione (ad es. la stringa di connessione al db) dal key vault; inoltre sarà implementata una funzionalità che consentirá di eseguire

l’upload in modalitá sia sincrona che asincrona di un file sullo storage account sempre del cloud Azure. Come step successivo ed opzionale ci sarà quello di effettuare un upload multiplo di file.

La nuova versione dell'applicazione deve consentire ad un amministratore di registrare i negozi dell’ azienda Contoso e per ciascun negozio associare almeno un Owner/Admin che gestirà il magazzino ed emetterá le fatture ai vari clienti dell’ECommerce.
Va implementata questa gerarchia:
Azienda Contoso
    Negozio Negozio A
        Negozio B
        ...
        Negozio N

I ruoli sono:

- Amministratore Globale
- Owner/Admin Negozio
- Utente ECommerce

Quando un utente compra un articolo andrá specificato di quale negozio è. In questo modo l’utente con il ruolo Owner/Admin del suddetto negozio potrà emettere fattura e salvarla sul db con il progressivo del negozio opportunamente valorizzato.
Le fatture avranno un progressivo relativo al negozio ed un altro progresso generale basato sulla data di emissione della fattura. Quindi solo l’utente con il ruolo Amministratore Globale potrà effettuare l’upload della fattura sullo storage account. In questa fase la fattura avrá il numero progressivo globale dell’azienda Contoso.
Riepilogando:

- l’utente con il ruolo Amministratore Globale vedrá tutti gli articoli e solo lui potrà eseguire l’upload della fattura già presente nel db sullo storage account (Cloud Azure).
- l’utente con il ruolo Owner/Admin Negozio vedrá solo gli articoli del proprio negozio ed emetterá e salverá la fattura sul db
- l’utente con il ruolo Utente E-Commerce vedrá tutti i prodotti e fornirá i dati personali di fatturazione

Chiaramente andranno aggiunte le tabelle, ad oggi mancanti, per soddisfare tali specifiche.
Aggiungere un campo Stato Fattura con i seguenti stati:

- None (l’utente che ha acquistato un prodotto NON ha inserto i suoi dati di fatturazione)
- Da emettere (l’utente che ha acquistato un prodotto ha inserto i suoi dati di fatturazione)
- Emessa (Salvata sul DB dal proprietario del negozio)
- Inviata (L’amministratore globale esegue l’upload)

Requisiti funzionali

Registrazione utente

- L'utente deve poter inserire il proprio nome, cognome, indirizzo email e password per registrarsi.
- L'indirizzo email deve essere unico per ogni utente.
- Dopo la registrazione, l'utente deve poter accedere all'applicazione.

Accesso utente

- L'utente deve poter inserire il proprio indirizzo email e la password per accedere all'applicazione.
- Deve essere presente la funzionalità di reset password tramite email contenente un link ad una pagina dove fare reset della password.

Gestione del profilo utente

- L'utente deve poter visualizzare e modificare le proprie informazioni personali, come nome, cognome e indirizzo email.
- L'utente deve poter modificare la propria password.

Funzionalità di ricerca

Devono esserci funzionalità di ricerca in base al ruolo:

- l'utente con il ruolo Amministratore Globale deve cercare prodotti per nome, negozio di appartenenza;
    - deve anche avere una maschera per ricercare se sono presenti fattura da salvare sullo storage account
- l'utente con il ruolo Owner/Admin Negozio deve cercare prodotti solo del suo negozio, per nome, tipologia
- l'utente con il ruolo Utente E-Commerce deve fare la ricerca dei prodotti su tutto il catalogo: progettare una funzionalità di ricerca base (ad es. free text) ed una funzionalità di filtri avanzati (ad es.per negozio, per range di prezzo, ecc.).

b. I risultati devono essere mostrati in una tabella ordinata di default in base ai prodotti più recenti
c. Ogni elemento della lista deve essere cliccabile per accedere alla pagina di dettaglio del prodotto.

Requisiti tecnici non funzionali

- L'applicazione deve utilizzare SQL Server come database.
- L'applicazione deve essere sviluppata con ASP.NET Core RESTAPI come application server ed Angular come front- end.
- L'applicazione dovrà gestire le eccezioni e loggare gli errori su un file di testo sul server
- L'applicazione dovrà utilizzare Microsoft Identity per la sicurezza e EntityFrameworkCore per l'accesso ai dati, con gestione di ruoli e/o claims.
- L'applicazione dovrà usare stored procedures per interagire con i dati
- L'applicazione deve essere multilivello, con una separazione chiara tra la logica di business, la persistenza dei dati e la presentazione dei dati.
- L'applicazione deve essere sicura, utilizzando le pratiche di sicurezza consigliate per l'accesso e la gestione dei dati degli utenti.
- L'applicazione dovrà usare il dependency injection pattern per l'iniezione delle dipendenze.
- L'applicazione deve essere testata adeguatamente per garantire la sua funzionalità e affidabilità.

Suggerimenti sullo sviluppo dell'applicazione:

Per lo sviluppo dell'applicazione, si consiglia di seguire le seguenti fasi:

1. Progettazione del database: definire lo schema del database e creare le tabelle necessarie per l'app
2. Sviluppo dei modelli: definire i modelli delle classi per rappresentare le entità del database (ad esempio, la classe User per rappresentare l'entità utente).
3. Sviluppo del controller Web API: creare il controller Web API per gestire le richieste e le risposte tra il front-end e il database. Il controller deve implementare i metodi per la gestione delle funzionalità richieste dall'applicazione (registrazione utente, accesso utente, gestione profilo utente, creazione/visualizzazione/modifica/eliminazione di post).
4. Sviluppo del servizio: creare un servizio per implementare la logica di business dell'applicazione e utilizzarlo all'interno del controller Web API.
5. Sviluppo del front-end: creare la parte front-end dell'applicazione utilizzando Angular.
6. Integrazione dei modelli con il front-end: integrare i modelli delle classi con il front-end per visualizzare i dati dell'applicazione.
7. Test dell'applicazione: testare l'applicazione per verificare la sua funzionalità e affidabilità.

Best practices

1. Prima di iniziare lo sviluppo dell'applicazione, leggere attentamente le specifiche del progetto e creare un piano di lavoro.
2. Utilizzare un ambiente di sviluppo integrato (IDE) come Visual Studio 2022 per creare l'applicazione. E visual Code per lo sviluppo dell'applicazione client in Angular (NodeJS).
3. Seguire le best practice per la scrittura del codice, ad esempio utilizzare il principio SOLID, creare test automatizzati e utilizzare un sistema di controllo versione come Git.
4. Assicurarsi di implementare tutte le funzionalità richieste e di testare l'applicazione per verificare la sua funzionalità e affidabilità.
5. Documentare il codice in modo chiaro e preciso.
6. Al termine dello sviluppo, consegnare il codice sorgente dell'applicazione e una breve relazione che descriva le scelte implementative e le difficoltà riscontrate durante lo sviluppo.

Validazione

1. Validazione lato client: implementare la validazione dei dati lato client utilizzando reactive form utilizzo e validazione delle form.
    a. https://angular.io/guide/reactive-forms
    b. https://www.tektutorialshub.com/angular/angular-reactive-forms/
2. Usare Microsoft identity platform for authentication using auth code flow
    a. https://learn.microsoft.com/en-us/azure/active-directory/develop/tutorial-v2-angular-auth-code
2. Validazione lato server: implementare la validazione dei dati lato server utilizzando le annotazioni di validazione di ASP.NET Core. La validazione lato server deve essere utilizzata per garantire che i dati siano conformi alle regole di business dell'applicazione e per proteggere l'integrità del database.
3. Verifica dell'unicità dei dati: verificare che i dati inseriti dagli utenti siano unici, ad esempio, verificare che l'indirizzo email di un utente non sia già presente nel database.
4. Validazione dei dati di input: verificare che i dati inseriti dagli utenti siano validi e conformi alle regole di business dell'applicazione. Ad esempio, verificare che la password dell'utente abbia almeno otto caratteri e che contenga almeno un carattere maiuscolo, un carattere minuscolo, un numero e un simbolo.
5. Gestione degli errori: gestire gli errori di validazione in modo chiaro e comprensibile per gli utenti, ad esempio, mostrare messaggi di errore in modo esplicito e fornire suggerimenti per correggere gli errori.
6. Protezione contro gli attacchi di injection: proteggere l'applicazione contro gli attacchi di injection, ad esempio, implementare la validazione dei dati di input e utilizzare i parametri dei comandi SQL per proteggere il database dagli attacchi di injection SQL.
7. Verifica delle autorizzazioni: verificare che l'utente abbia le autorizzazioni necessarie per accedere alle funzionalità dell'applicazione. Ad esempio, verificare che l'utente sia autenticato per accedere alle funzionalità riservate.
8. Verifica dei limiti: verificare che i dati inseriti dagli utenti rispettino i limiti imposti dalle regole di business dell'applicazione, ad esempio, verificare che l'utente possa fare upload solo di docx e pdf.Limitare la size del file da uploadare a 30 MB
9. Verifica delle dipendenze: verificare che i dati inseriti dagli utenti non abbiano dipendenze non valide o non esistenti.
10. Gestione delle eccezioni: gestire le eccezioni in modo corretto e mostrare messaggi di errore comprensibili agli utenti. Le eccezioni devono essere gestite in modo da evitare la visualizzazione di informazioni sensibili sul sistema.

Requisiti funzionali E-Commerce

1. Catalogo dei prodotti: l'applicazione deve consentire agli utenti di visualizzare il catalogo dei prodotti disponibili nel negozio online di abbigliamento.
2. I prodotti sono caratterizzati da attributi quali nome, categoria, prezzo unitario, quantità disponibile in magazzino.
3. Deve essere possibile cercare i prodotti per nome, categoria, range di prezzo.
4. L'utente deve poter visualizzare la cronologia degli ordini effettuati, con la possibilità di cancellare ordini non ancora spediti.
5. Ogni ordine può contenere uno o più prodotti, ciascuno con una quantità rappresentata da un numero intero (che determinerà il costo in base al costo unitario del prodotto presente nella tabella Prodotto). L'ordine deve contenere attributi quali il cliente, l'indirizzo di spedizione, la data dell'ordine, il metodo di spedizione (selezionare il tipo di corriere da una lista) eventuali commenti da parte del cliente, i dati di fatturazione solo se si vuole la fattura, lo stato dell'ordine (ad es. sottomesso, spedito, consegnato, ecc.) più eventuali altri metadati necessari all'applicazione.
6. Carrello della spesa: l'applicazione deve consentire agli utenti di aggiungere prodotti al proprio carrello della spesa, visualizzare il contenuto del carrello e rimuovere i prodotti dal carrello.
7. Checkout: l'applicazione deve consentire agli utenti di effettuare il checkout e di inserire le informazioni necessarie per la spedizione e la fatturazione degli ordini. Inoltre, deve essere possibile scegliere la modalità di pagamento preferita.
8. All'atto dell'acquisto dovrà partire una mail di notifica sia all'utente (con i dettagli dell'ordine) sia agli amministratori del sito.
9. All'atto dell'acquisto la disponibilità del prodotto dovrà essere decurtata in base alla quantità acquistata.
10. Gestione degli ordini: l'applicazione deve consentire al personale del negozio (tramite un'opportuna utenza di tipo amministrativo) di visualizzare gli ordini effettuati dagli utenti, modificare lo stato degli ordini (ad esempio da sottomesso a spedito a consegnato) e fornire informazioni di tracciamento della spedizione.
11. Quando un ordine cambia stato da parte degli amministratori, un'email di notifica deve partire al cliente.
12. Per ogni ordine che è stato pagato, l'amministratore deve essere in grado di generare una fattura che sarà memorizzata in una tabella che avrà almeno le seguenti colonne:
    a. Chiave primaria
    b. ID ordine
    c. Numero fattura (int incrementale per ogni anno solare)
    d. Stato fattura
    e. Data invio
    f. Note
    g. Data inserimento
    h. Partita IVA
    i. Codice fiscale
    j. Documento (varbinary(max)): conterrà il PDF della fattura.
    k. Data documento
    l. ID Utente creatore
    m. Indirizzo
    n. codice postale
    o. Città
    p. Paese

12. Gestione del magazzino: l'applicazione deve consentire al personale del negozio di gestire il magazzino e di aggiungere/modificare/rimuovere prodotti dal catalogo, modificare prezzi, disponibilità, ecc.
13. Analisi delle vendite: l'applicazione deve consentire al personale del negozio di visualizzare le statistiche sulle vendite, ad esempio, il numero di ordini effettuati, il fatturato, il numero di prodotti venduti per categoria o per marca. Tali statistiche saranno visibili da pagine web appropriate.

Fasi di sviluppo

1. Progettazione del database:
   - Identificare le entità principali dell'applicazione, ad esempio, prodotti, ordini, righe ordine, categorie prodotti (ogni prodotto dovrà avere una categoria prodotto associata), tabelle utenti e ruoli, ecc.
   - Progettare la struttura delle tabelle per ogni entità e definire le relazioni tra le tabelle.
2. Implementazione del database:
   - Creare le tabelle del database utilizzando un tool di gestione del database.
   - Definire le relazioni tra le tabelle utilizzando chiavi esterne.
   - Progettare le stored procedures per manipolare e leggere i dati gestiti dall'applicazione.
   - Inserire i dati di esempio nelle tabelle per verificare che il database funzioni correttamente.
3. Progettazione del back-end:
   - Progettare l'architettura del back-end, ad esempio, il modello API (Application Programming Interface).
   - Definire le API che verranno utilizzate dal front-end per interagire con il back-end (nel caso si volessero usare API REST).
4. Implementazione del back-end:
   - Creare i vari progetti per il back-end utilizzando il framework selezionato (ad esempio class libraries per l'accesso ai dati o ASP.NET Core Web applications).
   - Implementare le API utilizzando il linguaggio di programmazione del framework.
   - Utilizzare il database creato nella fase 2 per salvare e recuperare i dati.
5. Progettazione del front-end:
   - Selezionare il framework appropriato per il front-end, ad esempio Angular.
   - Progettare l'interfaccia utente utilizzando HTML, CSS e JavaScript.
   - Definire le funzionalità che l'utente dovrà essere in grado di eseguire, ad esempio, navigare tra le pagine, visualizzare i prodotti, effettuare un ordine, ecc.
6. Implementazione del front-end:

- Creare un progetto vuoto per il front-end utilizzando il framework selezionato.
- Implementare l'interfaccia utente utilizzando il linguaggio di programmazione del framework.
- Per la client app sviluppata in Angular, utilizzare le API del back-end create nella fase 4 per recuperare e visualizzare i dati nel front-end.
- Implementare l'upload https://www.bezkoder.com/angular-15-multiple-file-upload/
- Usare reactive form utilizzo e validazione delle form
- Implementare auth guard
- utilizzo ed integrazione librerie e componenti terze parti (Ex. ngPrime, material, ngBootstrap, fontAwesome).

7. Test e debugging:
   - Testare l'applicazione per verificare che funzioni correttamente.
   - Risolvere eventuali errori o bug rilevati durante i test.