Descrizione del progetto Team 1

L'obiettivo di questo progetto è lo sviluppo di un'applicazione multilivello per realizzare un Blog, usando .NET Core e ASP.NET Core Web API come application server, SQL Server come database e MVC come front-end (utilizzando, se possibile, bootstrap come libreria per lo stile).

L'applicazione deve consentire agli utenti di registrarsi, accedere e gestire le loro informazioni personali. Inoltre, l'applicazione deve consentire agli utenti registrati di creare, visualizzare, modificare ed eliminare i propri post, oltre che aggiungere commenti sia ai propri post che ai post degli altri

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 fere 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.

Gestione dei post

- L'entità Post dovrà contenere almeno un titolo ed un summary.
- Dovrà essere possibile assegnare una o più categorie al post: le categorie definiscono il tipo di post.
- Per ogni post gli utenti potranno sottomettere vari commenti
- L'utente deve poter creare un nuovo post inserendo il titolo, le categorie e il contenuto del post.
- L'utente deve poter visualizzare tutti i post creati da lui stesso.
- L'utente deve poter modificare dettagli di un post già creato.
- L'utente deve poter eliminare un post già creato.
- La lista dei post del blog deve essere visibile a tutti gli utenti, anche non registrati, ma per creare nuovi post o sottomettere commenti sarà necessario registrarsi nel Blog come detto sopra.
- Tutte le liste dovranno essere provviste di paginazione e della possibilità di ordinamento in base alle varie colonne della lista.

Accesso amministratore

- L'applicazione dovrà prevedere una utenza amministrativa del blog che avrà la possibilità di vedere e modificare (cancellare, modificare, nascondere senza cancellare) post e commenti sottomessi dagli utenti del blog.
- Se il commento di un utente viene cancellato/nascosto, deve partire via email un email informativa all'utente relativo.

Funzionalità di ricerca

a. Il blog deve contenere una funzionalità di ricerca che permette di cercare Post in base al testo ed in base alle categorie con cui è stato caratterizzato.

b. I risultati devono essere mostrati in una tabella ordinata di default in base ai post più recenti
c. Ogni elemento della lista deve essere cliccabile per accedere alla pagina di dettagli del post, con la possibilità di aggiungere commenti o, se si è il proprietario del post, di modificarlo.

Requisiti tecnici non funzionali

- L'applicazione deve utilizzare SQL Server come database.
- L'applicazione deve essere sviluppata con ASP.NET Core come application server e MVC come front-end.
- L'applicazione dovrà gestire le eccezioni e loggare gli errori su un file di testo sul server
- L'applicazione dovrà utilizzare IdentityCore 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.

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 il framework MVC di ASP.NET Core.
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.

Suggerimenti per lo sviluppo

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 per creare l'applicazione.
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 JavaScript e il framework di validazione fornito da ASP.NET Core. La validazione lato client deve essere utilizzata per migliorare l'esperienza utente e per prevenire la maggior parte degli errori di input degli utenti.
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 non inserisca più di 140 caratteri in un post.
9. Verifica delle dipendenze: verificare che i dati inseriti dagli utenti non abbiano dipendenze non valide o non esistenti, ad esempio, verificare che l'utente non possa aggiungere un commento a un post che non esiste.
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.

Descrizione del progetto Team 2

Di seguito le specifiche per un'applicazione di gestione di un negozio online di prodotti tecnologici. L'applicazione deve essere sviluppata utilizzando ASP.NET Core, SQL Server come database e tecnologie front-end come Angular oppure ASP.NET MVC.

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 fere 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.

Requisiti funzionali

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, 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. 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.

Requisiti tecnici non funzionali

- L'applicazione deve utilizzare SQL Server come database.
- L'applicazione deve essere sviluppata con ASP.NET Core Web API come application server.
- L'applicazione dovrà utilizzare IdentityCore 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 dovrà gestire le eccezioni e loggare gli errori su un file di testo sul server
- 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.

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 MVC (Model-View-Controller) o 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 o MVC.
   - 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.
- Se si utilizza Angular, utilizzare le API del back-end create nella fase 4 per recuperare e visualizzare i dati nel front-end.

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