Progetto di Applicazioni Web e Mobile Unicam.

Di Leonardo Serena, Ventura Sara.

Cos'è Giftify?

Giftify è una Single Page Application progettata per gestire in maniera semplice lo scambio di regali anonimi tra i membri di una community. Il concetto alla base dell'applicazione è quello di renderne più fluida l'organizzazione, integrando un sistema di sorteggio, la creazione di liste dei desideri, un sistema di autenticazione sicuro e un'interfaccia utente moderna. Questa piattaforma si adatta a diversi contesti sociali, come gruppi di amici, colleghi di lavoro o famiglie, ed è pensata per offrire un'esperienza intuitiva e divertente.

La scelta di sviluppare una Single Page Application rispetto ad una Multiple Page Application è data da una migliore prestazione, in quanto i dati scambiati sono minori rispetto ad una MPA e da un'interattività migliore. Grazie ai framework esistenti, inoltre, permettono l'implementazione di componenti modulari, riducendo così i tempi di sviluppo.

Funzionalità Principali

Gli utenti hanno la possibilità di creare comunità virtuali personalizzate o di unirsi a quelle esistenti mediante l'utilizzo di un codice univoco. All'interno di ogni comunità, i membri possono redigere una lista dei regali che desiderano ricevere (wishlist) e consultare quelle degli altri partecipanti.

Le comunità sono gestite da un amministratore, ovvero il creatore della comunità, il quale dispone di strumenti avanzati per monitorare e controllare le attività del gruppo. L'amministratore può, infatti, decidere di rimuovere membri dalla comunità o chiudere il gruppo una volta raggiunto il numero di partecipanti desiderato. Alla chiusura di quest'ultimo, il sistema procederà automaticamente con un sorteggio per assegnare a ciascun membro un altro partecipante a cui fare il regalo.

Ogni comunità ha una scadenza prestabilita che coincide con la data di scambio dei regali.

Sistema di Autenticazione

Il sistema di autenticazione implementato si basa su **Spring Security** e utilizza **JWT (JSON Web Tokens)** per garantire un accesso sicuro e protetto all'applicazione. Quando un utente si registra tramite l'endpoint dedicato, i suoi dati vengono salvati nel database, con la password criptata tramite **BCrypt**. Viene quindi generato un token JWT che contiene le informazioni necessarie per identificare l'utente e le sue autorizzazioni (token di accesso), e un token di refresh, che consente di rinnovare il token di accesso prima che scada.

Ogni richiesta successiva dell'utente include il token nell'intestazione **Authorization**. Un filtro personalizzato, denominato **AuthFilter**, intercetta la richiesta, valida il token utilizzando il servizio **JwtService** e autentica l'utente, registrandolo nel contesto di sicurezza di **Spring**.

Questo meccanismo protegge le risorse sensibili, consentendo l'accesso solo agli utenti autenticati.

Il processo di **logout** avviene tramite l'endpoint /auth/logout, dove il token in uso viene revocato e registrato in un repository dedicato, chiamato **RevokedTokenRepository**. La configurazione della sicurezza, definita nella classe **SecurityConfig**, stabilisce che gli endpoint di registrazione e login siano pubblici, mentre tutte le altre richieste richiedano l'autenticazione. Le politiche **CORS** consentono l'accesso da fonti specifiche, mentre la **Content Security Policy** protegge l'applicazione da attacchi come il **Cross-Site Scripting (XSS)**. La gestione del logout è affidata alla classe **LogoutHandlerImpl**, che aggiorna lo stato del token nel database.

Struttura dell'Applicazione

La SPA (Single Page Application) è stata sviluppata seguendo il pattern **MVC (Model-View-Controller)**.

Backend

Il backend è costituito dalle **entità**, che rappresentano i dati e la logica di business dell'applicazione. Le entità sono contrassegnate con l'annotazione @Entity per mappare i dati a tabelle del database. In **Giftify**, le entità corrispondono ai singoli elementi all'interno della SPA. In particolare, tra le principali entità troviamo:

- **Community**: Rappresenta una community, con attributi come nome, descrizione, amministratore, elenco di utenti, budget, data di scadenza per lo scambio dei regali, codice di accesso univoco, set di **GiftAssignment**, e altri.
- **Wish**: Rappresenta un desiderio dell'utente all'interno della sua wishlist.
- **WishList**: Rappresenta la lista dei desideri associata all'utente.
- **Account**: Rappresenta un account utente.
- **Token**: Rappresenta un token JWT e il suo stato.
- **RevokedToken**: Rappresenta i token scaduti.
- **GiftAssignment**: Rappresenta l'assegnazione di un regalo all'interno di una community.

Ogni **repository** è associato a una singola entità. I **servizi** contengono la logica di business e utilizzano i repository per accedere ai dati nel database. Questi servizi vengono iniettati direttamente nei controller. I principali servizi implementati sono:

- **CommunityService**: Gestisce la creazione di nuove community, l'adesione a una community esistente, la gestione delle wishlist degli utenti, l'estrazione di un nome all'interno della community e, se l'utente è amministratore, consente di chiudere o eliminare una community e rimuovere partecipanti.
- **Wish/WishlistService**: Consente di creare, modificare e rimuovere desideri, aggiornando automaticamente la wishlist associata.

- **JWTService**: Gestisce la generazione, validazione e gestione dei token JWT.
- **AccountService**: Si occupa delle operazioni sugli account utente, come la creazione e il recupero di utenti.
- **AuthenticationService**: Gestisce la registrazione, l'autenticazione degli utenti e la gestione dei token JWT.

I **controller** gestiscono le richieste HTTP e restituiscono le risposte al client. Nel caso specifico, l'applicazione include i controller **@RestController**, che espongono le API REST. I principali controller implementati sono: **AccountController**, **CommunityController**, **WishController**, e **AuthenticationController**.

Tecnologie Utilizzate

Per lo sviluppo del backend della SPA, è stato scelto il framework **SpringBoot**, che gestisce la configurazione e l'automazione del ciclo di vita dei componenti tramite annotazioni come @Service, @Component e @Autowired. SpringBoot si è rivelato scalabile, versatile e facilmente integrabile con altre tecnologie, supportando in modo ottimale le API REST.

Per la gestione dell'autenticazione e dell'autorizzazione degli utenti, è stato utilizzato **Spring Security**, che permette di implementare il controllo degli accessi sia a livello di API che di risorse. La gestione delle sessioni utente è affidata ai **JWT (JSON Web Tokens)**, che consentono di autenticare gli utenti senza la necessità di mantenere uno stato di sessione sul server.

Per il popolamento e la gestione dei dati, è stato scelto **H2 Database**, un database leggero e veloce, utile durante la fase di sviluppo. Questo database è creato in memoria e distrutto al termine dell'applicazione. La persistenza dei dati è gestita tramite **JPA (Java Persistence API)**, che consente di interagire con il database utilizzando entità e repository.

SpringBoot, combinato con **Lombok**, riduce la ripetitività del codice generando automaticamente metodi comuni come getter, setter, costruttori, e altri tramite annotazioni come @Getter, @Setter, @AllArgsConstructor, e @NoArgsConstructor. Inoltre, le annotazioni @Scheduled vengono utilizzate per eseguire operazioni periodiche, come il controllo della scadenza delle community o la rimozione di community obsolete. Questi task pianificati sono gestiti dal framework di **Spring**.

Frontend

Nel contesto del frontend, le entità vengono implementate in modo simile a quanto avviene nel backend. I controller, invece, sono realizzati tramite le componenti. In **Giftify**, le componenti principali includono quella per il login e la registrazione dell'utente, nonché la **HomeComponent**, che rappresenta la schermata principale dell'applicazione. Quest'ultima contiene diverse card, tra cui una per visualizzare le community di cui l'utente è membro, una per consentire l'accesso a una community già esistente e una per creare una nuova

community. Se l'utente clicca sul nome di una community nella prima card, verrà indirizzato alla schermata specifica di quella community.

La **CommunityComponent** rappresenta invece una community qualsiasi e include diverse card: una con le informazioni relative alla community, una che visualizza la wishlist dell'utente, una che mostra il nome estratto e una che permette di visualizzare i partecipanti alla community. L'utente ha la possibilità di aggiungere o rimuovere desideri dalla propria wishlist e, cliccando sul nome di un partecipante, può visualizzare la wishlist associata a quest'ultimo. Se l'utente è amministratore della community, oltre a poter chiudere ed eliminare la community, può rimuovere un partecipante dal gruppo.

Nel frontend, sono stati implementati anche i servizi per facilitare la comunicazione con il backend. Queste classi sono state create per gestire le componenti della home, della community, i desideri e l'autenticazione.

Inoltre, per il corretto funzionamento della Single Page Application (SPA), sono state implementate due classi fondamentali:

- **AuthGuard**: Questo servizio protegge le rotte dell'applicazione, consentendo l'accesso solo se l'utente è autenticato. L'interfaccia Angular **CanActivate** e il **AuthService** permettono di verificare l'autenticazione dell'utente e, in caso positivo, consentirgli di proseguire nell'applicazione, mentre in caso contrario, ne impediscono l'accesso.
- **TokenInterceptor**: Si tratta di un intercettore HTTP che aggiunge automaticamente il token di autenticazione a tutte le richieste HTTP, gestendo anche la scadenza dei token.

Tecnologie Utilizzate

Per lo sviluppo del frontend, è stato scelto il framework **Angular**, che grazie alla sua architettura basata su componenti, facilita una gestione modulare e scalabile dell'applicazione. Angular supporta anche il **two-way data binding**, che consente una sincronizzazione automatica tra il modello e la vista, aggiornando l'interfaccia ogni volta che i dati sottostanti vengono modificati. Inoltre, per la gestione del routing, Angular offre un router integrato che permette di definire e gestire i percorsi di navigazione e gli stati dell'applicazione in modo semplice ed efficace.