Documentazione API

NEBULAWATCHES

TOM SCHILLERWEIN, Giole Chiodoni e Alexandru Ciobanu I3

2024

Documentazione API

### Security

#### Endpoint

Di seguito vengono descritti gli endpoint dei tre controller contenuti in security: AuthenticationController (“/auth”), UserController (“/v1/user”) e AdminController (“/v1/admin”).

***POST “/auth/register”***

***body: request : RegisterRequest***

Questo endpoint serve a registrare un nuovo utente, la richiesta contiene username, email e password. Dopo l’inserimento nel database del nuovo utente, li viene mandata una email con un codice di verifica. Ecco il metodo sendEmail che permette di mandare un email contenente il codice di verifica all’utente tramite mailjet:

***POST “/auth/authenticate”***

***body: request : RegisterRequest***

Questo endpoint serve a autenticare un utente, la richiesta contiene email e password. Risponde con un JWT se va a buon fine. L’endpoint richiama il metodo authenticate in AuthenticaionService e controlla per primo se l’email è valida e che l’account non sia disabilitato. Dopodiché, usando l’AuthenticationManager di Java Spring viene controllato se l’email e password inserita sono corrette.

Infine, genera un token JWT per l'utente autenticato utilizzando jwtService e lo ritorna.

***GET “/auth/isTokenValid”***

***parameter: jwt : String***

Serve a verificare se il JWT che viene passato è valido. Risponde con ok (codice 200) se la richiesta è valida, altrimenti con un bad request (400). Se risponde con ok, nella response inserisce lo username dell’utente a cui appartiene il JWT.

***POST “/auth/verify”***

***parameter: code : int***

***parameter: email : String***

Serve a verificare l’email di un utente. Il controller richiama verifyEmail situato in AuthenticationService, dove viene controllato il token e se valido ritorna un JWT e imposta l’account come verificato.

***GET “/auth/isVerified/{email : String }”***

Serve a verificare se l’email, e perciò l’utente, è verificato. Ritorna true se è verificato e false se non lo è.

***GET “/auth/exists/{email : String}”***

Serve a verificare se l’email è già stata usata. Ritorna true se è usata e false se non lo è.

***GET “/auth/sendAgain”***

***parameter: email : String***

Serve a mandare nuovamente l’email di verifica all’utente.

***GET “/v1/user/getName/{userEmail : String }”***

Ritorna lo username di un utente in base all’email passata.

***GET “/v1/admin/getUsers”***

***header: headers : HttpHeaders***

Ritorna la lista di utenti del sito. Per tutti gli endpoint in AdminController ho controllato se l’account che gli esegue è di tipo Admin con questo codice:

***GET “/v1/admin/getUser/{id : int}”***

***header: headers : HttpHeaders***

Ritorna l’utente con id specificato, se la richiesta viene fatta da un account Admin.

***GET “/v1/admin/getRole/{userEmail : String}”***

***header: headers : HttpHeaders***

Ritorna il ruolo (Admin o User) di un utente, se la richiesta viene fatta da un account Admin.

***POST “/v1/admin/deleteUser”***

***header: headers : HttpHeaders***

***body: request : AdminRequest***

Archivia un utente e perciò lo disattiva, se l’account è Admin.

***POST “/v1/admin/saveUser”***

***header: headers : HttpHeaders***

***body: request : AdminRequest***

Salva un nuovo utente con le informazioni presenti in AdminRequest, se la richiesta viene fatta da un account Admin.

***POST “/v1/admin/updateUser”***

***header: headers : HttpHeaders***

***body: request : AdminRequest***

Modifica un utente con le informazioni nuove passate in AdminRequest, se la richiesta viene fatta da un account Admin.

***POST “/v1/admin/searchUser”***

***header: headers : HttpHeaders***

***body: searchRequest : SearchRequest***

Ritorna una lista di utenti che rispettano il contenuto della stringa di ricerca contenuta in SearchRequest. Questo funziona solo se l’account che fa la richiesta è di tipo Admin.

***GET “/v1/admin/isEmailUsed/{userEmail : String}”***

Verifica se l’email è gia usata, in questo caso ritorna true, altrimenti ritorna false.

### Storage

Il package storage contiene tutto quello che ha a che fare con gli orologi preferiti di un utente e i gli orologi che possiede o ha venduto.

#### Endpoint

Di seguito vengono descritti gli endpoint dei due controller contenuti in storage: StorageController (“/v1/storage”) e FavouriteController (“/v1/favourite”).

***GET “/v1/storage/{userEmail : String}”***

Questo endpoint ritorna una ResponseEntity contenente una lista di Orologi che l’utente ha nello storage.

***POST “/v1/storage/addWatchToStorage”***

***body: request : StorageRequest***

Questo endpoint aggiunge un nuovo orologio allo storage di un utente. Ritorna una ResponseEntity che esprime se l’operazione di aggiunta allo storage è andata a buon fine. Tutte le informazioni richieste per aggiungere un nuovo orologio allo storage sono contenute in StorageRequest.

***POST “/v1/storage/removeStorageWatch”***

***body: request : StorageRequest***

Questo endpoint rimuove un orologio dallo storage di un utente. Ritorna una ResponseEntity che esprime se l’operazione di rimozione dallo storage è andata a buon fine.

***POST “/v1/storage/editStorageWatch”***

***body: request : StorageRequest***

Questo endpoint modifica un orologio contenuto nello storage di un utente. Esso viene utilizzato quando un utente vende un orologio e bisogna modificare stato, data di vendita, prezzo di vendita e quantità di un orologio nello storage.

***POST “/v1/storage/saveCustom”***

***parameters: reference : String***

***parameters: name : String***

***parameters: description : String***

***parameters: retailPrice : float***

***parameters: email : String***

***parameters: file : MultipartFile***

Questo endpoint serve a aggiungere un orologio custom da parte dell’utente. Esso è strutturato in modo diverso dagli altri endpoint di inserimento (tramite parametri e non tramite body), a causa del MultipartFile che viene passato dall’utente come immagine dell’orologio. Dopo che i dati sono stati controllati nel service vengono inseriti nel repository.

***GET “/v1/storage/custom/{userEmail : String}”***

Questo endpoint ritorna una ResponseEntity contenente una lista di Orologi custom che l’utente ha inserito manualmente.

***GET “/v1/storage/custom/{reference : String}/image”***

Questo endpoint ritorna una ResponseEntity contenente i byte dell’immagine di un orologio custom.

***POST “/v1/storage/removeCustom”***

***body: request : CustomWatchRequest***

Questo endpoint elimina un orologio custom e ritorna stato ok (200) se è andato a buon fine.

***GET “/v1/storage/getCustomInfo/{watchReference : String}”***

Questo endpoint ritorna un Orologio custom in base alla reference, se esiste.

***GET “/v1/storage/getStorageByTeam/{teamId : Long}”***

Questo endpoint ritorna una lista di orologi nello storage che hanno nel campo teamId, l’id passato come parametro. Questo serve a prendere tutti gli orologi venduti da un membro del team.

***GET “/v1/storage/getWatchSoldByClient}”***

***parameters: email : String***

***parameters: clientId : Long***

Questo endpoint ritorna una lista di orologi nello storage che hanno nel campo clientId, l’id passato come parametro. Questo serve a prendere tutti gli orologi venduti a un cliente.

***GET “/v1/storage/getWatchSoldByClientNum}”***

***parameters: email : String***

***parameters: clientId : Long***

Questo endpoint ritorna il numero di orologi comprati da un cliente.

***GET “/v1/storage/getWatchSoldByClientTotalExpenses}”***

***parameters: email : String***

***parameters: clientId : Long***

Questo endpoint ritorna quanto un singolo cliente ha speso comprando orologi.

***GET “/v1/storage/getWatchesOwnedByClientMonth}”***

***parameters: id : Long***

***parameters: month : int***

Questo endpoint ritorna quanti orologi sono stati venduti nell’ultimo mese ad un cliente.

***GET “/v1/favourite/{userEmail : String}”***

Questo endpoint ritorna una ResponseEntity contenente una lista di Orologi che l’utente ha marcato come preferiti.

***POST “/v1/storage/addFavourite”***

***body: request : FavouriteRequest***

Questo endpoint aggiunge un nuovo orologio ai preferiti di un utente. Ritorna una ResponseEntity che esprime se l’operazione di aggiunta allo storage è andata a buon fine. Tutte le informazioni richieste per aggiungere un nuovo orologio ai preferiti sono contenute in FavouriteRequest.

***POST “/v1/storage/removeFavouriteWatch”***

***body: request : FavouriteRequest***

Questo endpoint rimuove un orologio dai preferiti di un utente. Ritorna una ResponseEntity che esprime se l’operazione di rimozione dai preferiti è andata a buon fine.

***GET “/v1/favourite/checkFavourite/{reference : String}/{userEmail : String}”***

Questo endpoint controlla se un certo orologio è già nei preferiti di un utente, ritornando un boolean come risposta.

### Team

Il package team contiene tutto quello che ha a che fare con i membri del team, che vendono gli orologi del nostro negozio.

#### Endpoint

Di seguito vengono descritti gli endpoint del controller contenuto in team: TeamController (“/v1/team”).

***GET “/v1/team/getTeam/{email : String}”***

Questo endpoint ritorna una lista di membri del team che sono del utente la cui email è stata passata come parametro.

***GET “/v1/team/getTeamMember/{id : Long}”***

Questo endpoint ritorna un membro del team in base all’id.

***POST “/v1/storage/add”***

***body: request : TeamRequest***

Questo endpoint aggiunge un nuovo membro al team di un utente. Ritorna una ResponseEntity che esprime se l’operazione di aggiunta allo storage è andata a buon fine. Tutte le informazioni richieste per aggiungere un nuovo orologio ai preferiti sono contenute in TeamRequest.

***POST “/v1/storage/delete/{id : int}”***

Questo endpoint imposta il membro del team come disabilitato. Questo è utile per non far perdere la coerenza dei dati nello storage. Ritorna una ResponseEntity che esprime se l’operazione di rimozione del membro del team è andata a buon fine.

***POST “/v1/storage/updateTeam”***

***body: request : TeamRequest***

Questo endpoint modifica il membro del team in base alle informazioni contenute in TeamRequest. Ritorna una ResponseEntity che esprime se l’operazione di update è andata a buon fine.

***GET “/v1/team/countSold/{id : int}”***

Questo endpoint ritorna il numero di orologi venduti dal membro del team.

***GET “/v1/team/countSoldMonth/{id : int}/{month : int}”***

Questo endpoint ritorna il numero di orologi venduti nell’ultimo mese.

### Client

Il package client contiene tutto quello che ha a che fare con i membri del team, che vendono gli orologi del nostro negozio.

#### Endpoint

Di seguito vengono descritti gli endpoint del controller contenuto in client: ClientController (“/v1/clients”).

***GET “/v1/clients/all/{email : String}”***

Ritorna una lista di clienti associati all’utente di cui si passa l’email.

***GET “/v1/clients/{id : Long }”***

Ritorna un Cliente che ha l’id come quello passato come parametro.

***POST “/v1/clients/add”***

***body: request : ClientRequest***

Questo endpoint aggiunge un nuovo cliente all’utente. Ritorna una ResponseEntity che esprime se l’operazione di aggiunta allo storage è andata a buon fine. Tutte le informazioni richieste per aggiungere un nuovo orologio ai preferiti sono contenute in ClientRequest.

***DELETE “/v1/clients/delete/{id : Long}”***

Questo endpoint imposta il cliente come disabilitato. Questo è utile per non far perdere la coerenza dei dati nello storage. Ritorna una ResponseEntity che esprime se l’operazione di rimozione del membro del team è andata a buon fine.

***PUT “/v1/clients/update/{id : Long}”***

***body: newClient : Client***

Questo endpoint modifica il client alle informazioni contenute in Client, se il client non esiste ne crea uno nuovo. Ritorna una ResponseEntity che esprime se l’operazione di update è andata a buon fine.

### Watches

Il package “Watches” contiene tutto ciò che riguarda gli orologi.

#### Endpoint

Di seguito vengono descritti tutti gli endpoint del package.

***GET “/v1/watches/ {watchReference : String}”***

Ritorna la classe “Optional<WatchDTO>”, la classe “Optional” serve a ritornare null se non viene trovato un orologio con un certo “watchReference”, la classe “WatchDTO” serve a mandare le informazioni dell’orologio, non si manda la classe “Watch” direttamente perché altrimenti le relazioni molti a molti non verrebbero mappate, come ad esempio i materiali usati dall’orologio o i prezzi.

***GET “/v1/watches/ {watchReference : String}/image”***

Ritorna l’immagine dell’orologio con “watchReference” i bytes dell’immagine in Base64.

***GET “/v1/watches/filter”***

***RequestParams: query: String (not required), page: int, sortBy: String (default “name”),***

***pageSize: int (default 20), brands: List<Brand> (not required), families: List<Families>,***

***watchShapes: List<WatchShape> (not required), watchIndexes: List<WatchIndex>***

***(not required), dialColors: List<DialColor> (not required), dialFinishes: List<DialFinish>***

***(not required), materials: List<Materials> (not required), minDiameter: float (default 0),***

***maxDiameter: float (default 1000), minWaterResistance: float (default 0),***

***maxWaterResistance: float (default 30000), minPrice: float (not required),***

***maxPrice: float (not required)***

Ritorna tutti gli orologi che corrispondono ai filtri selezionati, la query è ottimizzata e ritorna soltanto “reference”, “name”, “isLimitedTo” per essere il più veloce possibile e anche perché nella view della ricerca, vengono usati soltanto questi valori.

***GET “/v1/brands/”***

***RequestParams: page: int, sortBy: String***

Ritorna i “Brand” per pagina.

***GET “/v1/brands/{brandName :String}/families”***

***RequestParams: page: int, sortBy: String***

Ritorna i “Family” per pagina del “Brand”.

***GET “/v1/brands/{brandName : brandName}/families/all”***

***RequestParams: page: int, sortBy: String***

Ritorna i “Family” del “Brand”, usati nella view della ricerca, per i filtri.

***GET “/v1/brands/{brandName : String}/rndimage”***

Ritorna una immagine random di un orologio del ”Brand”.

***GET “/v1/brands/all”***

Ritorna tutti i “Brand”, usati nella view della ricerca, per i filtri.

***GET “/v1/families/”***

Ritorna tutti i “Family”, usati nella view della ricerca, per i filtri.

***GET “/v1/families/{familyId : int}/rndimage”***

Ritorna una immagine random di un orologio della “Family”.

***GET “/v1/families/{familyId : int}/watches”***

***RequestParams: page: int, sortBy: String***

Ritorna tutti i “WatchDTO” per “Family”.

***GET “/v1/dial\_colors/”***

Ritorna tutti i “DialColor”, usati nella view della ricerca, per i filtri.

***GET “/v1/dial\_finishes/”***

Ritorna tutti i “DialFinish”, usati nella view della ricerca, per i filtri.

***GET “/v1/hands/”***

Ritorna tutti i “Hands”, usati nella view della ricerca, per i filtri.

***GET “/v1/materials/”***

Ritorna tutti i “Material”, usati nella view della ricerca, per i filtri.

***GET “/v1/ watch\_indexes/”***

Ritorna tutti i “WatchIndexes”, usati nella view della ricerca, per i filtri.

***GET “/v1/ watch\_shapes/”***

Ritorna tutti i “WatchShape”, usati nella view della ricerca, per i filtri.