Documentazione NebulaWatches

1 Introduzione 3

1.1 Informazioni sul progetto 3

1.2 Abstract 3

1.3 Scopo 3

2 Analisi 4

2.1 Analisi del dominio 4

2.2 Analisi e specifica dei requisiti 4

2.3 Use case 7

2.4 Pianificazione 7

2.5 Analisi dei mezzi 8

2.5.1 Software 8

2.5.2 Hardware 8

3 Progettazione 9

3.1 Design dell’architettura del sistema 9

3.2 Design dei dati e database 10

3.3 Design delle interfacce 11

3.4 Design procedurale 11

4 Implementazione 12

4.1.1 MySQL 12

4.1.2 Node 12

4.1.3 Java 12

4.1.4 Maven 12

4.1.5 Python3 13

4.1.6 Servizi 13

4.2 Frontend 14

4.3 Backend 14

4.3.1 Security 15

4.3.2 Storage 18

4.3.3 Team 20

4.3.4 Client 21

4.3.5 Watches 21

5 Test 23

5.1 Protocollo di test 23

5.2 Risultati test 23

5.3 Mancanze/limitazioni conosciute 23

6 Consuntivo 23

7 Conclusioni 23

7.1 Sviluppi futuri 24

7.2 Considerazioni personali 24

8 Glossario 24

9 Bibliografia 25

9.1 Sitografia 25

10 Allegati 25

11 Indice Delle Figure 25

# Introduzione

## Informazioni sul progetto

* Titolo Progetto: NebulaWatches
* Allievo: Alexandru Ciobanu I3BB, Gioele Chiodoni I3BB, Tom Schillerwein I3BC
* Docente responsabile: Guido Montalbetti
* Scuola: Arti e Mestieri Trevano, sezione Informatica
* Data di inizio e fine: 12.01.2024 – 03.05.2024
* Data di presentazione: 17 – 24 .05.2024

## Abstract

For this project, a web application needs to be developed to manage a watch store.

Users have several basic management tools at their disposal, such as a list of watches from which they can compose their inventory, do advanced watch searches, and view their customers and what they buy.

They also have at their disposal advanced tools such as a chatbot, based on data from out Database, and a company message centralization point to improve and optimize the management of their watch store.

In addition, this project is about learning how to manage a project in the agile mode and how to work and share work in a team.

## Scopo

Per questo progetto bisogna sviluppare un’applicazione web che permetta la gestione di un negozio di orologi.

Gli utenti hanno a disposizione diversi tool di gestione base, come una lista di orologi da cui possono comporre il loro inventario, fare ricerche avanzate sugli orologi, visualizzare i propri clienti e ciò che comprano.

Inoltre hanno a disposizione dei tool avanzati come un chatbot, basato sui dati del database, e un punto di centralizzazione dei messaggi dell’azienda per migliorare e ottimizzare la gestione del proprio negozio di orologi.

Inoltre questo progetto serve a imparare a gestire un progetto nella modalità agile e a lavorare e spartirsi il lavoro in un team.

# Analisi

## Analisi del dominio

Abbiamo scelto di sviluppare un applicativo web che permetta all’utente di migliorare e semplificare la gestione del proprio negozio di orologi. Il prodotto deve essere facile da usare e presentare un’interfaccia grafica moderna. A nostro sapere non esiste un prodotto specifico per la gestione dei negozi di orologi sul mercato.

## Analisi e specifica dei requisiti

|  |  |
| --- | --- |
| **ID: REQ-001** | |
| **Nome** | Accesso e login |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Un utente per poter utilizzare il sito deve effettuare il login. |
| **Sotto requisiti** | |
| **001** | Maschera di login |

|  |  |
| --- | --- |
| **ID: REQ-002** | |
| **Nome** | Registrazione |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Un utente senza account deve poter registrarsi tramite email oppure tramite Google |
| **Sotto requisiti** | |
| **001** | Maschera di registrazione |
| **002** | Se con email bisogna fare la verifica dell’email |

|  |  |
| --- | --- |
| **ID: REQ-003** | |
| **Nome** | Database |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Database contenente orologi, utenti, inventario dell’utente, clienti dell’utente e team dell’utente |

|  |  |
| --- | --- |
| **ID: REQ-004** | |
| **Nome** | Chatbot |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Chatbot basato su dati del database che aiuta e dà consigli all’utente |

|  |  |
| --- | --- |
| **ID: REQ-005** | |
| **Nome** | Comunication center |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Centro di raggruppamento email |

|  |  |
| --- | --- |
| **ID: REQ-006** | |
| **Nome** | Orologi nel magazzino |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L’utente può selezionare gli orologi che ha nel proprio inventario/magazzino |
| **Sotto requisiti** | |
| **001** | Orologi venduti e comprati |
| **002** | Informazioni su spedizione, consegna, ecc. |

|  |  |
| --- | --- |
| **ID: REQ-007** | |
| **Nome** | Orologi |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Gli orologi sono ordinati per brand. |
| **Sotto requisiti** | |
| **001** | Ricerca e filtri di ricerca |
| **002** | Preferiti |

|  |  |
| --- | --- |
| **ID: REQ-008** | |
| **Nome** | Team |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L’utente può aggiungere e gestire i propri membri del team |
| **Sotto requisiti** | |
| **001** | Informazioni di vendita dei membri del team |

|  |  |
| --- | --- |
| **ID: REQ-009** | |
| **Nome** | Clienti |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Informazioni sui propri clienti |
| **Sotto requisiti** | |
| **001** | Orologi comprati |

|  |  |
| --- | --- |
| **ID: REQ-010** | |
| **Nome** | Admin |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Gestione CRUD degli utenti |
| **Sotto requisiti** | |
| **001** | Paginazione e ricerca dei utenti. |

## Use case

|  |
| --- |
| 1 Use Case |

## Pianificazione

Prima di stabilire una pianificazione bisogna avere almeno una vaga idea del modello di sviluppo che si intende adottare. In questa sezione bisognerà inserire il modello concettuale di sviluppo che si seguirà durante il progetto. Gli elementi di riferimento per una buona pianificazione derivano da una scomposizione top-down della problematica del progetto.

La pianificazione può essere rappresentata mediante un diagramma di Gantt:

|  |
| --- |
| 2 Diagramma di Gantt Preventivo |

## Analisi dei mezzi

Per questo progetto usiamo tre PC scolastici su cui eseguiamo delle macchine virtuali Linux Mint per lo sviluppo.

### Software

* Visual Studio Code 1.85.2, come editor di testo
* IntelliJ IDEA Community Edition 2023.3.2, come IDE
* Google Chrome 116.0, come browser per visualizzare il sito durante lo sviluppo
* Mozilla Firefox 110.0, per testare la visualizzazione del sito
* Microsoft Project, per il Gantt
* Microsoft Visio, per fare lo Use Case
* Microsoft Word e Obsidian, per redigere la documentazione, il QdC e il diario
* GitHub, come Repository online
* Postman v10.22 per testare gli endpoint dell’API
* Trello.com, per la gestione dei task negli sprint

### Hardware

Computer utilizzato come host delle macchine virtuali:

* Processore: Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz
* RAM: 32GB
* Scheda Video: NVIDIA GeForce RTX 2060
* SSD: 512GB
* 2 Display 1920x1080 60Hz

Virtual Machine su cui viene hostato il progetto:

* RAM: 4GB
* IP: 10.20.4.181

# Progettazione

## Design dell’architettura del sistema

|  |
| --- |
| 3 Architettura |

## Design dei dati e database

|  |
| --- |
| 4 Schema ER |

## Design delle interfacce

|  |
| --- |
| 5 Design della Home |

È stato realizzato solo il design della pagina Home, dato che le altre pagine saranno abbastanza simili a questa.

## Design procedurale

Cioba fa diagramma o qualcosa del genere di classi

# Implementazione

### MySQL

Per installare MySQL sul server bisogna eseguire i seguenti comandi da terminale:

sudo apt update

sudo apt install mysql-server

sudo systemctl start mysql.service

Dopo l’esecuzione di questi comandi bisognerà cambiare la password dell’utente root.

### Node

Si è deciso di installare Node.js con NVM anziché utilizzare APT, poiché quest'ultimo non installa necessariamente l'ultima versione. Inoltre, considerando l'ampio utilizzo di librerie, sarebbe facile cambiare la versione di Node.js nel caso in cui certe librerie non fossero compatibili con la versione attuale.

Per l’installazione bisogna eseguire i seguenti comandi da terminale:

curl -o- <https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh>

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh | bash

source ~/.bashrc

Dopodichè si avrà la possibilità di scegliere la versione di Node con i seguenti comandi da terminale:

nvm list-remote

nvm install vXX.XX.X

nvm use vXX.XX.X.

### Java

Per installare Java 21 si deve eseguire il seguente comando da terminale:

sudo apt install openjdk-21-jdk

### Maven

Per installare Maven basta eseguire il seguente comando da terminale:

sudo apt install mvn

Dopo l’installazione è necessario impostare il proxy, altrimenti non sarà possibile scaricare le librerie usate.

Per impostare il proxy bisogna modificare il file nella home dell’utente, al seguente percorso “.m2/settings.xml”:

<settings>

  <proxies>

    <proxy>

      <id>proxy</id>

      <active>true</active>

      <protocol>http</protocol>

      <host>10.20.5.51</host>;

      <port>8888</port>

      <username></username>

      <password></password>

      <nonProxyHosts>localhost|127.0.0.1</nonProxyHosts>;

    </proxy>

  </proxies>

</settings>

### Python3

Per installare Python3, che di solito è già installato, bisogna eseguire il seguente comando da terminale:

sudo apt install python3

Per installare Pip3, che serve a installare le librerie di Python, bisogna eseguire il seguente comando da terminale:

apt install python3-pip

### Servizi

Per creare i servizi bisogna creare prima i file con estensione service.

nebulaWatchesAPI.service:

[Unit]

Description=nebulaWatchesAPI service

After=mysqld.service

StartLimitIntervalSec=0

[Service]

Type=simple

Restart=always

RestartSec=1

User=administrator

ExecStart=java -jar /home/administrator/NebulaWatches/5\_Applicativo/nebulaWatchesAPI/target/nebulaWatchesAPI-0.0.1-SNAPSHOT.jar

[Install]

WantedBy=multi-user.target

nebulaWatchesFrontend.service:

[Unit]

Description=nebulaWatchesFrontend service

After=nebulaWatchesFrontend.service

StartLimitIntervalSec=0

[Service]

Type=simple

Restart=always

RestartSec=1

User=administrator

WorkingDirectory=/home/administrator/NebulaWatches/5\_Applicativo/client

ExecStart=/usr/bin/npm run serve

[Install]

WantedBy=multi-user.target

nebulaWatchesChatbot.service:

[Unit]

Description=nebulaWatchesChatbot service

After=mysqld.service

StartLimitIntervalSec=0

[Service]

Type=simple

Restart=always

RestartSec=1

User=administrator

ExecStart=/usr/bin/python3 /home/administrator/NebulaWatches/5\_Applicativo/Chatbot/server.py

[Install]

WantedBy=multi-user.target

Dopo aver creato questi file è necesario spostarli in “/etc/systemd/system”, per poi abilitarli e farli partire con i seguenti comandi da terminale:

sudo systemctl enable nebulaWatchesAPI

sudo systemctl enable nebulaWatchesFrontend

sudo systemctl enable nebulaWatchesChatbot

sudo systemctl start nebulaWatchesAPI

sudo systemctl start nebulaWatchesFrontend

sudo systemctl start nebulaWatchesChatbot

## Frontend

Il frontend è sviluppato con Vue JS usando la sintassi di script setup e composition api. Per la parte grafica abbiamo usato Tailwind in combinazione con Shadcn Vue. Abbiamo strutturato le view utilizzando componenti creati da noi e usando quelli di Shadcn Vue.

|  |
| --- |
| 6 Struttura Frontend |

### Struttura View

Una view è strutturata

## Backend

Il backend è principalmente basato su Java Spring Boot. Tuttavia, abbiamo sviluppato il chatbot in Python per ragioni di comodità e perché le librerie utilizzate non erano completamente compatibili con Java.

|  |
| --- |
| 7 Struttura file progetto |

I file sono organizzati in diversi pacchetti, seguendo le best practice di Java Spring. Ogni pacchetto rappresenta una funzionalità specifica del sistema, come ad esempio 'watches' o 'security'.

All'interno di ciascun pacchetto, troviamo ulteriori suddivisioni:

* **config**: contiene i file di configurazione per l'applicazione Spring.
* **controller**: ospita i controller Spring responsabili della gestione delle richieste HTTP.
* **dto**: qui sono collocati i Data Transfer Object (DTO), utilizzati per semplificare il trasferimento dei dati tra il client e il server.
* **exception**: raccoglie le eccezioni personalizzate, come ad esempio 'WatchNotFoundException', utili per gestire situazioni anomale durante l'esecuzione dell'applicazione.
* **model**: contiene i modelli dati, rappresentati tramite classi Java, utilizzati dal sistema per rappresentare le entità.
* **repository**: include le repository Spring Data, che offrono un'interfaccia per l'interazione con il database. All'interno di queste repository, vengono definite le query personalizzate per l'accesso ai dati, ad esempio quelle per la ricerca degli orologi.
* **service**: contiene i servizi Spring, che implementano la logica di business dell'applicazione. Spesso, i servizi fanno uso delle repository per accedere ai dati e sono utilizzati dai controller per soddisfare le richieste degli utenti.

### Security

Il login dell’applicativo funziona con JWT (JSON Web Token), il quale viene generato al login di un utente e scade dopo 24 ore. Questo migliora la sicurezza e inoltre questo metodo offre una scalabilità molto ampia dato che facilita l’autenticazione tra più server. Inoltre in questo package si trovano tutti i metodi collegati all’admin.

#### Config

Nella cartella config si trovano tutte le classi e metodi che servono a configurare in modo corretto il login con JWT e a definire a quali pagine un utente può accedere.

Nella classe SecurityConfig viene impostato una whitelist degli URL che sono accessibili senza login, perciò tutte le pagine che hanno a che fare con l’authenticazione, nel nostro caso “/auth”.

Nella classe JwtAuthenticationFilter viene filtrato il token, e tramite metodi presenti nei services, verifica se il token fornito è valido.

Nella Classe ApplicationConfig sono definiti vari metodi che servono a facilitare l’uso dei metodi di autenticazione nelle altre classi.

#### 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:

public void sendEmail(String email, int code) throws MailjetException {  
 MailjetClient client;  
 MailjetRequest request;  
 MailjetResponse response;  
 client = new MailjetClient("", "");  
 request = new MailjetRequest(Emailv31.resource)  
 .property(Emailv31.MESSAGES, new JSONArray()  
 .put(new JSONObject()  
 .put(Emailv31.Message.FROM, new JSONObject()  
 .put("Email", "nebulawatchesproject@gmail.com")  
 .put("Name", "nebulawatchesproject@gmail.com"))  
 .put(Emailv31.Message.TO, new JSONArray()  
 .put(new JSONObject()  
 .put("Email", email)  
 .put("Name", email)))  
 .put(Emailv31.Message.SUBJECT, "LogIn in NebulaWatches")  
 .put(Emailv31.Message.HTMLPART,  
 "<title>Benvenuto su NebulaWatches</title"  
 )));  
 response = client.post(request);  
 System.out.println(response.getStatus());  
 System.out.println(response.getData());  
}

***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.

public AuthenticationResponse authenticate(AuthenticationRequest request) {  
 if (InputUtils.isEmailValid(request.getEmail()) && !repository.isArchived(request.getEmail()).get()) {  
 authenticationManager.authenticate(  
 new UsernamePasswordAuthenticationToken(  
 InputUtils.testInput(request.getEmail()),  
 InputUtils.testInput(request.getPassword())  
 )  
 );  
 var user = repository.findByEmail(InputUtils.testInput(request.getEmail())).orElseThrow();  
 var jwtToken = jwtService.generateToken(user);  
 return AuthenticationResponse.builder()  
 .token(jwtToken)  
 .build();  
 }  
 return null;  
}

***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:

String token = headers.getFirst("Authorization");

if(token != null && adminService.isAdminByToken(token)){  
 return adminService.getAllUsers();  
}else{  
 return null;  
}

***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.

# Test

## Protocollo di test

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-001 | **Nome:** | Endpoint registrazione utente |
| **Descrizione:** | Passando username, email, password, viene creato l’utente e viene generato un token di autenticazione. | | |
| **Prerequisiti:** | Installato Postman, bisogna avere il server MySQL, avere il server Spring acceso. | | |
| **Procedura:** | 1. Accendere Postman 2. Creare collezione “NebulaWatches” 3. Creare richiesta POST e mettere come URL “localhost:64321/api/v1/auth/register” poi andare nella tab “Body/raw”: { “username” : “gino”, “email”: “gino@gmail.com”, “password”: “1234” } 4. Schiacciare Send | | |
| **Risultati attesi:** | Nella response un token di autorizazzione | | |

## Risultati test

## Mancanze/limitazioni conosciute

Centro e-mail, non abbiamo implementato questa feature perché ci è finito il tempo e sarebbe stato una parte complicata e non molto veloce da fare.

Login con Google, abbiamo speso molto tempo cercando di realizzare questo, ma non siamo riusciti a far funzionare il resto del sito con il login di Google. Tuttavia, siamo riusciti a farlo funzionare in un progetto a parte.

# Consuntivo

Consuntivo del tempo di lavoro effettivo e considerazioni riguardo le differenze rispetto alla pianificazione (cap. 1.7) (ad esempio Gantt consuntivo).

# Conclusioni

Questo progetto è abbastanza unico a suo modo dato che non ci sono noti altri siti web per la gestione del proprio negozio di orologi. Questo sito è molto specifico, infatti il suo target è un mercato di nicchia. Comunque con una interfaccia grafica avvincente e moderna pensiamo che questo progetto possa essere utile.

## Sviluppi futuri

Implementazione del centro di e-mail, implementazione del login con Google, implementare che un membro del team aggiunto nella sezione “Team” può accedere e inserire vendite all’interno dell’applicativo.

## Considerazioni personali

Siamo generalmente soddisfatti de nostro progetto, anche se non completamente finito, ha molte feature utili per una gestione di un negozio.

Abbiamo imparato a lavorare in un gruppo e con una modalità agile su un progetto di un semestre. Inoltre, abbiamo imparato molto sullo sviluppo web, specialmente usando due linguaggi che a scuola non abbiamo trattato. Anche lo sviluppo di un chatbot ci ha permesso di imparare molti concetti e skill utili per il futuro.

# Glossario

|  |  |
| --- | --- |
| **Termine** | **Descrizione** |
| JWT | **JSON Web Token**: standard web per lo scambio di dati definito dalla RFC 7519 proposto nel 2015. |
| MailJet | **Mailjet** è una piattaforma di email ceh permette di mandare email tramite Rest API. |
| Rest API | **Representational State Transfer Application Programming Interface:** fornisce un'interfaccia basata su HTTP per consentire ai client di accedere e manipolare le risorse del server in modo efficiente e scalabile |
| Tailwind CSS | **Tailwind CSS** è un framework CSS open source usato per applicare velocemnte uno stile alle pagine HTML. |
| Shadcn Vue | **Shadcn Vue:** libreria di componenti grafici che utilizza vue e Tailwind per creare componenti grafici accattivanti. |

# Bibliografia

## Sitografia

1. URL del sito (se troppo lungo solo dominio, evt completo nel diario),
2. Eventuale titolo della pagina (in italico),
3. Data di consultazione (GG-MM-AAAA).

* <https://spring.io/>, varie pagine
* <https://chat.openai.com/>, varie chat

# Allegati

* Diari di lavoro
* Applicativo web
* Quaderno dei Compiti
* Abstract
* Gantt (preventivo e consuntivo)
* Use Case

# Indice Delle Figure

[1 Use Case 7](#_Toc165031449)

[2 Diagramma di Gantt Preventivo 8](#_Toc165031450)

[3 Architettura 9](#_Toc165031451)

[4 Schema ER 10](#_Toc165031452)

[5 Design della Home 11](#_Toc165031453)

[6 Struttura file progetto 14](#_Toc165031454)