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 9

3.3 Design delle interfacce 9

3.4 Design procedurale 9

4 Implementazione 10

5 Test 10

5.1 Protocollo di test 10

5.2 Risultati test 11

5.3 Mancanze/limitazioni conosciute 11

6 Consuntivo 11

7 Conclusioni 11

7.1 Sviluppi futuri 11

7.2 Considerazioni personali 11

8 Glossario 11

9 Bibliografia 12

9.1 Bibliografia per articoli di riviste: 12

9.2 Bibliografia per libri 12

9.3 Sitografia 12

10 Allegati 12

# 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

Descrive:

* Ciobanu Fa schema

## Design dei dati e database

|  |
| --- |
| 3 Schema ER |

## Design delle interfacce

|  |
| --- |
| 4 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

## Installazione server

## Frontend

Ciao

## Backend

Spiegazione struttura, pacchetti, models, service, controller, ecc.

### 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.

#### 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.

#### Controller

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

***POST “/auth/register”***

***body: request : RegisterRequest***

Serve a registrare un nuovo utente, la richiesta contiene username, email e password.

***POST “/auth/authenticate”***

***body: request : RegisterRequest***

Serve a autenticare un utente, la richiesta contine email e password. Risponde con un JWT se va a buon fine.

***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

A

### Chatbot

Login

Nel nostro applicativo il login si può fare con due modalità. La prima tipologia di login è basata su JWT (JSON web token), la seconda è un login con Google.

Login JWT

Per registrare un nuovo utente, l’applicazione necessita di un email, username e password. Se la richiesta è valida, l’applicativo ritorna un token che serve a dire al server che questo utente è autenticato.

Per autenticare un utente già esistente, l’applicazione necessita di email e password dell’utente. Anche in questo caso, se la richiesta è valida, ritorna un token che segnala al server che questo utente è autenticato.

L’utente invia il JWT ricevuto in ogni richiesta successiva e l’applicazione verifica il JWT per autenticare l’utente.

Se il token sta per scadere, l’applicazione può decidere di rinnovarlo e inviare un nuovo JWT all’utente.

Login Google

Quando un utente vuole accedere, l’applicazione lo reindirizza alla pagina di login di Google. L’utente inserisce poi le proprie credenziali di Google e viene autenticato se questo è andato a buon fine.

# 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. |
|  | . |

# 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).

**Esempio:**

* http://standards.ieee.org/guides/style/section7.html, *IEEE Standards Style Manual*, 07-06-2008.

# Allegati

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

# Indice Delle Figure

[1 Use Case 7](#_Toc165023081)

[2 Diagramma di Gantt Preventivo 8](#_Toc165023082)

[3 Schema ER 9](#_Toc165023083)

[4 Design della Home 10](#_Toc165023084)