# Relazione Tecnica
# NAO Challenge24

**By**

Team NaoNexus

**Per maggiori informazioni:**

Visita il nostro sito, la nostra repository
e scrivi una mail a: `socialnaonexus@gmail.com`

Scuole "Alle Stimate", Verona

**Sommario**

La Nao Challenge quest'anno si basa sul retail: la vendita diretta di beni o servizi ai consumatori finali attraverso punti vendita fisici oppure online.

Il team NaoNexus ha scelto quindi di utilizzare il NAO come assistente robotico in negozio Swarovski al fine di rendere la scelta del prodotto da acquistare più facile e veloce per tutti. Inoltre il NAO viene utilizzato per gestire le quantità dei prodotti presenti in magazzino per negozio e avvisare lo Store Manager in caso di necessità di rifornimento.

In questa relazione verranno spiegati i metodi di lavoro, le tecnologie utilizzate e i sotto team della squadra NaoNexus.

# Indice

# 1 Metodologia e tecniche utilizzate

Poiché NaoNexus è un gruppo numeroso, c'era la necessità di tenere traccia dei *tasks* e mantenere ogni componente al passo con l'ultima versione del software sviluppato. Per tale scopo, il team ha sfruttato due importanti tecnologie: *GitHub* e *Notion*.

## 1.1 GitHub

GitHub è uno strumento che ha aiutato il team a salvare e gestire il codice. È una piattaforma cloud-based di hosting di repository Git che aiuta i membri a tenere traccia dei cambiamenti fatti nel codice grazie al controllo di versione.Tutte le modifiche apportate sono tracciate e possono essere annullate nel caso portino ad errori.

Attraverso il branching e il merging, ogni membro è stato in grado di lavorare in modo sicuro sul codice, evitando di intaccare direttamente la versione principale.

## 1.2 DevOps con Notion e metodologia Agile

Per lavorare al progetto e tenere traccia del progresso, il team ha utilizzato Notion, una piattaforma collaborativa per la gestione di note. Il termine DevOps proviene dall'unione di "development" e "operations" ed è costituito dai vari metodi che hanno lo scopo di migliorare la collaborazione end-to-end. La web app Notion presenta

molti strumenti come la gestione di task, il tracciamento dell'andamento del progetto e le liste to-do, grazie ai quali ogni componente del team é al corrente degli incarichi da portare a termine ad ogni appuntamento. Questa piattaforma di collaborazione integra wiki e database, creando uno spazio di lavoro per la gestione di dati e task.

Il progetto è stato gestito con la metodologia Agile: i compiti principali sono stati divisi in sub-tasks per essere poi distribuiti tra i membri del team, portando a una maggiore efficienza nello sviluppo del software.

## 2 I due ruoli dei NAO

Il primo NAO è quello che sta tra i clienti e viene usato come un assistente al quale rivolgersi direttamente in negozio. È connesso ad un telefono tramite un server HTTP e gli utenti possono iniziare la conversazione con il robot tramite un'applicazione a cui si sono precedentemente collegati. Dopo aver analizzato il cliente tramite la tecnologia di Morphcast, il NAO gli chiede alcuni dettagli (il budget, se si tratta di un regalo e il sesso del destinatario) e propone un prodotto. Questo è scelto tramite un decision tree che utilizza tutte le informazioni in suo possesso sul destinatario del gioiello. Il dialogo con il cliente continua fino a quando egli non sceglie di aggiungere il prodotto e i suoi ipotetici abbinamenti (sempre proposti dal NAO) al carrello.

Il ruolo del secondo NAO, presente in cassa, è quello di assistente gestionale del negozio. La web app è connessa tramite un server HTTP ad un database e permette ai dipendenti di Swarovski di visualizzare le vendite di un determinato prodotto, gli utenti registrati, i carrelli attivi, i prodotti sullo scaffale e in magazzino. Il NAO ha il ruolo di avvisare lo Store Manager della necessità di ordinare i prodotti sotto una certa soglia in magazzino (valore ottenuto dalla web app), compresi quelli in vetrina, che verranno riforniti il prima possibile.

## 2.1 NAO come shop assistant

Il primo NAO, già presente in negozio, inizia il dialogo con il cliente e lo analizza: calcola il livello di rabbia, disgusto, paura, felicità, neutralità, tristezza, sorpresa e attenzione, rileva se si tratta di un uomo o di una donna e calcola un'età probabile. A questo punto viene chiesto al cliente se si tratta di un regalo per un'altra persona.

In caso di risposta affermativa, l'umanoide richiede ulteriori informazioni che non può rilevare da solo: l'età e il sesso del destinatario. Se invece si tratta di un acquisto personale, questi passaggi vengono saltati in quanto viene utilizzata la tecnologia Morphcast.

In entrambi i casi communique, vengono richiesti il budget e il tipo di gioiello che il cliente sta cercando (categoria). Utilizzando questi dati, attraverso un decision tree, il NAO propone dei prodotti all'utente fino a quando questo non ne aggiunge uno al carrello. A quel punto, il cliente viene indirizzato alla cassa con il QR code generato sull'app per il pagamento.

Per il funzionamento del primo robot NAO, ci sono serviti:

- **Choregraphe:** Nel nostro progetto, abbiamo adottato un approccio innovativo alla programmazione del NAO, evitando l'uso del software Choregraphe, tradizionalmente utilizzato per configurare comportamenti e interazioni nei robot umanoidi.

Abbiamo scelto invece di scrivere il codice in Python, che interagisce direttamente con il NAO. Inizialmente, abbiamo estrapolato i blocchi di codice da Choregraphe, che, insieme a componenti chiave come il dialogo, il decision tree e la tecnologia di Morphcast, ha costituito la base per instaurare una comunicazione interattiva tra il robot e il cliente. Successivamente, i blocchi sono stati integrati all'interno di funzioni Python personalizzate, offrendoci la flessibilità di programmare le varie interazioni del NAO.

Abbiamo implementato una funzione per estrarre le risposte dei clienti, consentendo al robot di comprendere e reagire alle interazioni umane. Altre funzioni utilizzate sono quelle di riconoscimento e sintesi vocali, attraverso le quali il robot è in grado di convertire il testo in risposte vocali, arricchendo così la sua interazione con l'utente. Inoltre, abbiamo implementato una funzione che permette al robot di rilevare la presenza di un cliente e di seguirlo con lo sguardo. Questo non solo agevola la profilazione dell'utente, ma aggiunge un livello di interattività al NAO, creando un'esperienza più coinvolgente per il cliente finale.

L'approccio basato su codice Python ci ha consentito di personalizzare le interazioni del robot in modo dettagliato e di adattarle dinamicamente alle esigenze del progetto. La nostra metodologia mira a sfruttare

appieno le potenzialità del robot NAO, offrendo un'esperienza unica e interattiva agli utenti.

La funzione animated say permette al NAO di riprodurre vocalmente stringhe Python, mentre la funzione face tracker permette al NAO di seguire il volto dell'umano con cui sta dialogando o interagendo.

Ecco come abbiamo implementato questi blocchetti Choregraphe su Python:

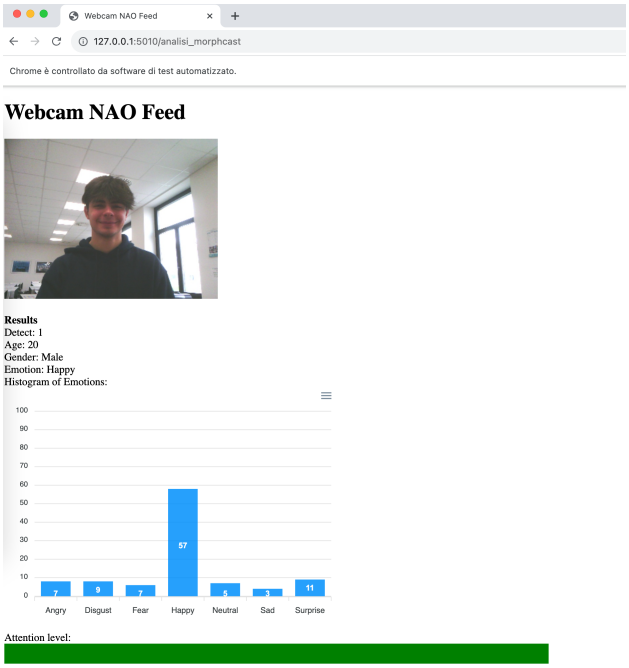
```
def nao_animatedSayText(text_to_say):
tts_proxy = ALProxy("ALAnimatedSpeech",\
nao_ip, nao_port)
animated_speech_config = {"bodyLanguageMode":\
"contextual"}
tts_proxy.say(text_to_say,\
animated_speech_config)
tts_proxy = None

def nao_face_tracker():
tracker_proxy = ALProxy("ALTracker", nao_ip,\
nao_port)
targetName = "Face"
faceWidth  = 0.1
tracker_proxy.setMode("Head")
tracker_proxy.registerTarget(targetName,\
faceWidth)
tracker_proxy.track(targetName)
```

- **HTTP Server:** Il server è una parte del codice molto complessa scritta in Python in cui tutti i dati del progetto sono gestiti. È connesso al database per estrarre e aggiungere dati, permette il funzionamento della web app e del NAO e l'analisi delle immagini passate dal robot per l'estrazione delle emozioni.
- **NAO:** NAO è la parte centrale del progetto. È connesso al server HTTP e analizza tramite la sua fotocamera l'utente, estraendo e salvando le emozioni. Interagisce con il cliente consigliando l'oggetto più adatto da scegliere. Inoltre, avvisa la Store Manager quando i prodotti sullo scaffale o in magazzino stanno per terminare, specificando il modello del gioiello.
- **API Morphcast:** Morphcast ci ha permesso di utilizzare le sue API di intelligenza artificiale. Per maggiori informazioni visita il loro sito:

  MorphCast®

  Qui viene riportata la pagina web della web app che permette di visualizzare graficamente il lavoro della tecnologia di Morphcast:




- **Decision Tree:** Il decision tree è un modello di apprendimento automatico che organizza decisioni logiche in una struttura ad albero per la classificazione o la regressione. Ogni nodo rappresenta una scelta basata su un attributo, con rami che indicano le possibili conseguenze di quella decisione. Il nostro decision tree prende in considerazione tre variabili principali che vengono date in input dall'utente:
  - Gender
  - Age
  - Budget

  In base a queste variabili, il codice restituisce una

lista di gioielli che possono interessare all'utente. Vediamo più a fondo il nostro codice:

- Con la prima condizione si controlla il genere della persona a cui è destinato il gioiello e in base alla risposta dell'utente si procede in un determinato ramo dell'albero decisionale (male/female).
- La richiesta dell'età del futuro possessore del gioiello è necessaria per poter consigliare un prodotto che possa essere gradito maggiormente.
- Quando siamo entrati all'interno dell'if e abbiamo fornito un budget, il programma attraverso una successione di condizioni controlla il prezzo di ogni prodotto verificando se rispetta le necessità dell'utente.

Se il decision tree trova uno o più prodotti che sono conformi a genere, età, categoria di gioiello e budget, restituisce in output una lista con i consigliati.

Una parte del codice del decision tree, scritto su Python:

```
.....................................................
if gender == 'male':
    if age > 60:
        return []
    elif age < 20:
```

```
            if category == 'bracelet':
                if budget >= 155:
                    for item in product_info:
                        if item['gender'] == 'M'\
                        and item['age'] < 20 and \
                        item['category'] == 'bracelet'\
                        and item['prezzo'] >= 155:
                       gioielli_consigliati.append\
                       (item['id'])
                    return gioielli_consigliati
                else:
                    return []
            elif category == 'necklace':
                if budget < 175:
                    return []
                else:
                    for item in product_info:
                        if item['gender'] == 'M' and \
                        item['age'] < 20 and \
                        item['category'] == 'necklace'\
                        and item['prezzo'] >= 175:
                        gioielli_consigliati.append\
                        (item['id'])
                    sreturn gioielli_consigliati
            else:
                return []
.....................................................
```

- **Database:** Per gestire le informazioni relative ai prodotti, ai clienti e agli ordini, abbiamo creato un database utilizzando PostgreSQL tramite l'interfaccia grafica di pgAdmin.

### Modello ER (Entità-Relazione)

Prima di procedere con l'implementazione del database, abbiamo realizzato una fase preliminare di progettazione del modello ER per identificare le entità e le relazioni chiave all'interno del sistema.

Tale modello aiuta infatti a visualizzare la struttura del database e le relazioni tra le entità coinvolte.

Le principali entità identificate nel modello ER includono:

- Cliente: rappresenta i clienti del negozio, con attributi come ID, username, nome, cognome, password
- Carrello: rappresenta le informazioni riguardanti i carrelli dei clienti, con attributi come ID e ID del cliente
- Oggetto: rappresenta i vari articoli di gioielleria offerti dal negozio, con attributi come ID, categoria, descrizione, prezzo, foto ecc.
- Carrello Oggetto: rappresenta gli oggetti presenti nel carrello, con attributi come ID, ID del carrello e ID dell'oggetto

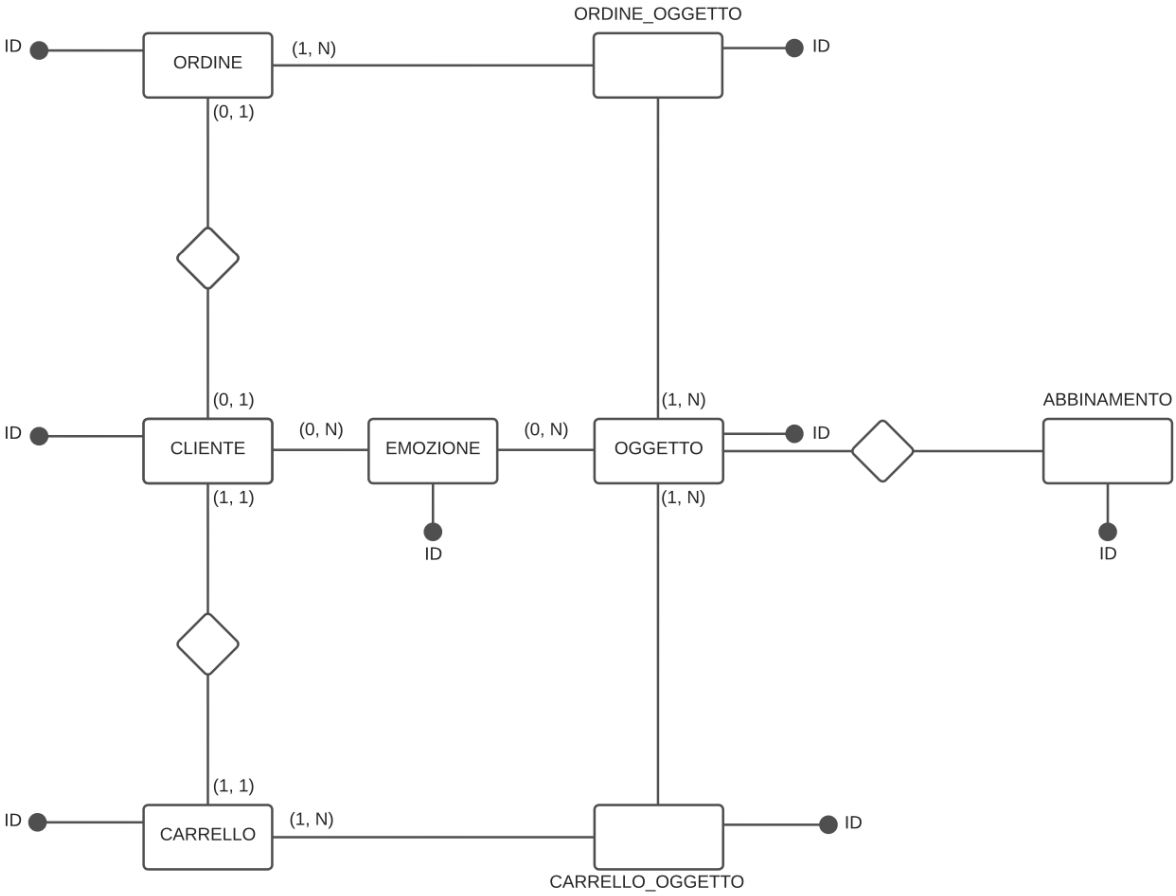
- Ordine: rappresenta gli ordini effettuati dai clienti, con attributi come ID, ID del cliente, data dell'acquisto, modalità di pagamento ecc.
- Ordine Oggetto: rappresenta i dettagli specifici di ciascun ordine,
  includendo: ID dell'ordine, ID dell'articolo, quantità, prezzo unitario, ecc.
- Emozione: rappresenta il profilo del cliente e le sue emozioni, con attributi come ID, età, sesso e indice di gradimento
- Abbinamento: rappresenta i possibili abbinamenti per ogni prodotto, con attributi come ID del primo oggetto, ID del secondo ecc.

### Schema Logico Relazionale

Sulla base del modello ER progettato, abbiamo creato uno schema logico e relazionale per implementare il database utilizzando le tabelle in PostgreSQL.

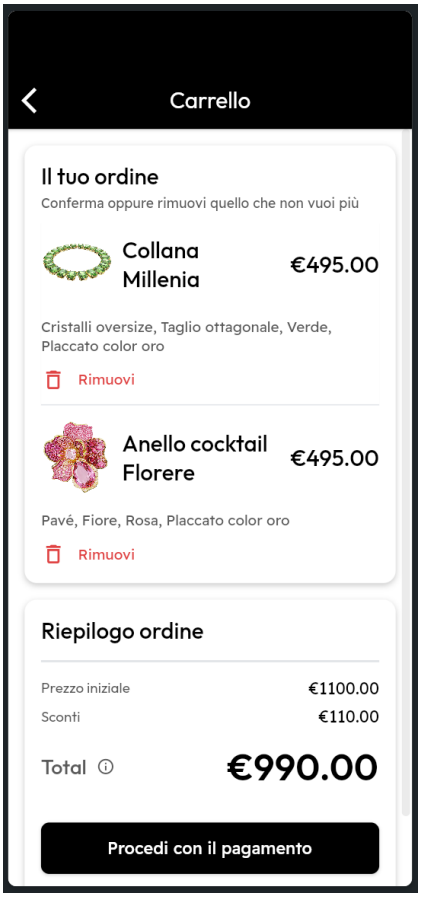
Le relazioni tra le tabelle sono state implementate utilizzando chiavi primarie e chiavi esterne per garantire l'integrità referenziale e l'associazione corretta dei dati.

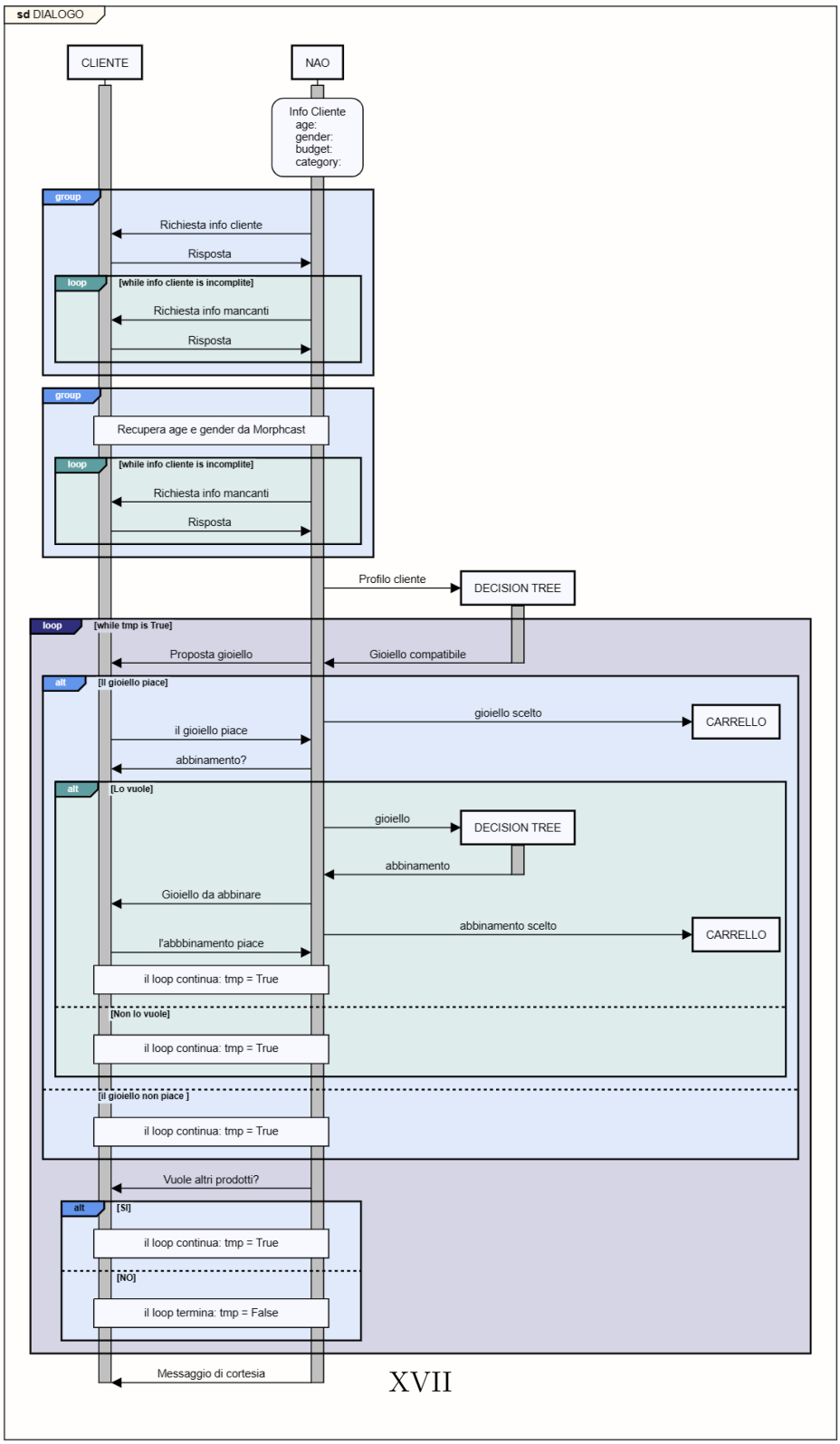
Per assicurarci che il database funzionasse come previsto, durante la sua creazione abbiamo popolato le tabelle con dati di esempio e testato le query.



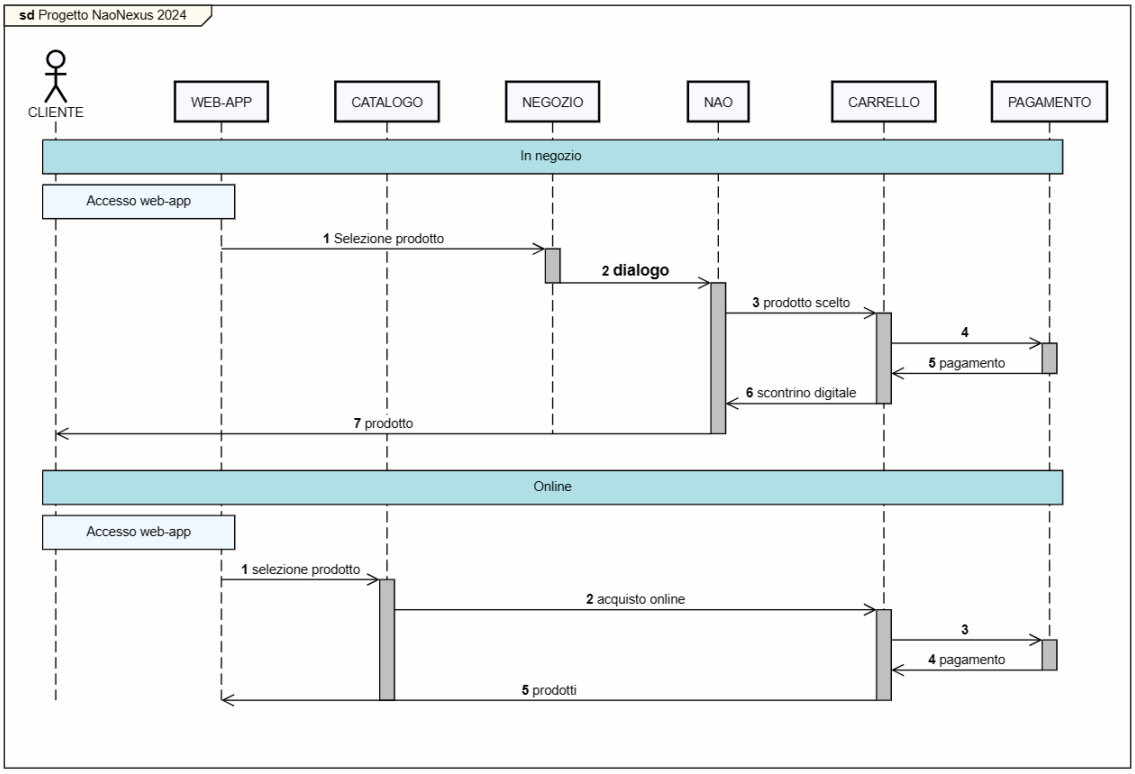

- **App:** L'applicazione è lo strumento con il quale l'utente si interfaccia con one retail. Attraverso essa , ha accesso al catalogo online, tranquillamente consultabile da casa, e all'esperienza interattiva in negozio. È stata sviluppata in flutter, un framework che permette di compilare binaries sia per android che per iOS, tagliando i tempi di sviluppo e aumentando la compatibilità. L'app comunica direttamente con il web server attraverso API Rest, scambiandosi

dati in formato JSON. In questo modo, può ricavare dati come il catalogo del negozio o il carrello associato all'utente loggato, oppure inviare richieste come l'aggiunta e la rimozione di un prodotto al carrello o addirittura comandi al NAO.




sd DIALOGO
CLIENTE
NAO
Info Cliente
age:
gender:
budget:
category:
group
Richiesta info cliente
Risposta
loop
[while info cliente is incomplite]
Richiesta info mancanti
Risposta
group
Recupera age e gender da Morphcast
loop
[while info cliente is incomplite]
Richiesta info mancanti
Risposta
Profilo cliente
DECISION TREE
loop
[while tmp is True]
Proposta gioiello
Gioiello compatibile
alt
[Il gioiello piace]
gioiello scelto
CARRELLO
il gioiello piace
abbinamento?
alt
[Lo vuole]
gioiello
DECISION TREE
abbinamento
Gioiello da abbinare
abbinamento scelto
CARRELLO
l'abbbinamento piace
il loop continua: tmp = True
[Non lo vuole]
il loop continua: tmp = True
[il gioiello non piace ]
il loop continua: tmp = True
Vuole altri prodotti?
alt
[SI]
il loop continua: tmp = True
[NO]
il loop termina: tmp = False
Messaggio di cortesia

Descrizione: Il progetto NaoNexus 2024 orchestra un'esperienza di shopping senza soluzione di continuità, unendo interazioni fisiche e online. I clienti, rappresentati da CLIENTE, possono utilizzare un'applicazione web (WEB-APP) per accedere a un catalogo (CATALOGO) e fare selezioni di prodotti. Nel negozio fisico, NEGOZIO intraprende un dialogo con il robot NAO, che aggiunge i prodotti scelti al carrello della spesa (CARRELLO). Il pagamento viene elaborato tramite PAGAMENTO e viene generato uno scontrino digitale. Allo stesso modo, nello scenario online, i clienti accedono a WEB-APP, interagiscono con il CATALOGO e selezionano articoli aggiunti al CARRELLO online. Il processo di pagamento è gestito da PAGAMENTO e i clienti ricevono prodotti acquistati e informazioni online. Questo approccio integrato migliora il percorso di acquisto per NaoNexus nel 2024.




Il Diagramma Dialogo illustra un dialogo di acquisto online tra il cliente (CLIENTE) e NAO. Vengono considerati due scenari di acquisto: l'acquisto come regalo e l'acquisto personale. Utilizzando un albero decisionale, l'assistente suggerisce gioielli compatibili in base al profilo del cliente. Il dialogo coinvolge cicli per raccogliere informazioni complete e fare selezioni di prodotti. L'interazione continua finché il cliente non completa l'acquisto o decide di non aggiungere altri articoli. Successivamente, viene inviato un messaggio di cortesia e sia il cliente che l'assistente vengono disattivati, concludendo il dialogo.

## 2.2 NAO come assistente alla gestione

Il ruolo del secondo NAO consiste nel gestire la quantità di gioielli sullo scaffale e nel magazzino, avvisando quando è necessario lo Store Manager affinché possa rifornire i gioielli mancanti. Sulla web app si possono inoltre vedere le vendite dei vari prodotti singolarmente, gli account attivi e i carrelli attivi.

Per il funzionamento del secondo robot NAO, ci sono serviti:

1. **NAO:** L'umanoide NAO è fondamentale per rendere più umana e veloce la gestione del negozio. Grazie ad esso non c'è più bisogno di controllare quanti oggetti sono presenti in magazzino o sullo scaffale in quanto basterà premere un pulsante sulla web app.

   Il NAO, una volta premuto il pulsante "Report" sulla web app, avviserà immediatamente lo Store Manager di riempire nuovamente lo scaffale o fare un ordine per il magazzino nel caso in cui i prodotti siano sotto una certa soglia n.

   Ciò è fondamentale per rendere più veloce la gestione delle quantità di gioielli, facendo così risparmiare tempo allo Store Manager.

2. **HTTP Server:** È il punto centrale in cui tutti i dati del progetto sono gestiti, ed è una parte del

codice molto complessa scritta in Python. Permette il funzionamento della webapp, è connesso al database per estrarre e aggiungere dati, permette al NAO di funzionare e consente l'analisi delle immagini passate dal NAO per l'estrazione delle emozioni.

3. **Web app:** Una parte importante del server è la webapp che viene servita da esso: le pagine HTML sono popolate utilizzando Jinja e sono inviate usando Flask. Questa libreria è responsabile del funzionamento del server Rest che distribuisce i dati anche in formato JSON. In questa webapp si possono visualizzare gli utenti e i carrelli attivi, permettendo anche la modifica dei dati; inoltre sotto forma di grafico si possono vedere le vendite dei vari prodotti. La web app inoltre permette di inserire i nuovi utenti dell'applicazione, che possono quindi essere registrati solo in negozio.
   Qui sotto viene riportata una delle pagine della web app, permette di controllare la quantità di prodotti presenti in negozio:

SWAROVSKI

Gioielli sullo scaffale:

40

Report

| ID | Titolo | Categoria | Qtà Scaffale | Foto | |
|---|---|---|---|---|---|
| 1 | angelic bracelet | bracelet | 1 |  | |
| 2 | angelic necklace | necklace | 1 |  | |
| 3 | constella cocktail ring | ring | 0 |  | |

Questa pagina della web app permette allo Store Manager di visualizzare un tabella che mostra le quantità dei vari gioielli in negozio.Il tasto "REPORT" permette di richiedere al robot NAO quali prodotti hanno bisogno di essere riforniti sullo scaffale/vetrina. L'umanoide avviserà subito lo Store Manager di quali prodotti c'è bisogno dal magazzino.

# 3 Componenti del team

Il team si è diviso in due gruppi, il team social e team di coding.

## 3.1 TEAM CODING

Il team coding ha lavorato intensamente per sviluppare il codice necessario per il funzionamento dei progetti legati al robot NAO. I membri di questo team hanno rivestito vari ruoli:

- Edoardo Polfranceschi - Team Leader: Ha coordinato le attività del team e si è occupato della creazione del server e della webapp.
- Aurora Savoia: Ha contribuito alla creazione dell'applicazione con il catalogo dei prodotti, le pagine prodotto e lo scanner dei QR code.
- Shenal Fernando: Ha sviluppato il controllo del NAO tramite Python e non più Choregraphe.
- Antonio Galati: Si è cimentato nello sviluppo dell'applicazione.
- Giacomo Santi: Ha creato il decision tree per la scelta automatica del prodotto.
- Chiara De Marchi: Ha creato il database in linguaggio SQL.

## 3.2 TEAM SOCIAL

Il team social si è occupato della promozione del progetto attraverso i canali social e la produzione di contenuti multimediali. Le principali attività svolte da questo team includono:

- Alberto Rubini - Team Leader: Ha coordinato le attività del team social e si è occupato della creazione dei contenuti.
- Davide Masini: Ha contribuito alla creazione del sito web 2024 del team NaoNexus.
- Arianna Antonelli: Ha aiutato alla creazione del sito e dei contenuti digitali.
- Laura Mascalzoni: Si è occupata dei social del team e della stesura di testi per il progetto.

# 4 Conclusioni e ringraziamenti

La Nao Challenge è stata un'esperienza intensa e formativa per tutto il team. Siamo riusciti a sviluppare due progetti innovativi che sfruttano le capacità del robot NAO in ambiti diversi, dimostrando le nostre competenze tecniche e organizzative.

Ringraziamo calorosamente il team di Swarovski Italia per averci supportato dandoci la possibilità di collaborare con loro, e in ugual modo ringraziamo Morphcast per averci permesso di utilizzare la loro tecnologia per umanizzare ancora di più l'interazione NAO-cliente.

Noi tutti membri del team ringraziamo particolarmente il professor Giovanni Bellorio (Il Supremo) per averci dato la possibilità di partecipare a questo progetto coinvolgente. E inoltre lo ringraziamo per averci seguito nella realizzazione del progetto istruendoci al mondo del lavoro di gruppo.