Gestionale Magazzino

# Indice

1 Indice 2

2 Introduzione 4

2.1 Informazioni sul progetto 4

2.2 Abstract 4

2.3 Scopo 4

2.3.1 Scopi didattici 4

2.3.2 Scopi operativi 4

3 Analisi 5

3.1 Analisi del dominio 5

3.2 Analisi e specifica dei requisiti 5

3.3 Use case 10

3.4 Pianificazione 11

3.5 Analisi dei mezzi 12

3.5.1 Software 12

3.5.2 Hardware 12

4 Progettazione 13

4.1 Design dell’architettura del sistema 13

4.2 Design dei dati e database 14

4.2.1 Diagramma ER 14

4.2.2 Descrizione delle Tabelle 16

4.2.3 Descrizione delle Relazioni 17

4.2.4 Procedure e Trigger 17

4.3 Design delle interfacce 18

4.3.1 Design iniziale 18

4.4 Design procedurale 22

4.4.1 Noleggio articoli 22

4.4.2 Chiusura noleggio 23

4.4.3 Sistema di notifica noleggio in scadenza 24

4.4.4 Sistema di notifica noleggio scaduto 24

5 Implementazione 25

5.1 Configurazione server 25

5.1.1 Installazione MySQL 25

5.1.2 Installazione Nodejs 25

5.1.3 Installazione PM2 e script per l’auto deploy 26

5.2 Certificato TLS/SSL 27

5.2.1 Generazione certificato 27

5.2.2 Implementazione applicativo 27

5.3 Struttura applicativo 28

5.4 Connessione al database 29

5.5 Models 29

5.5.1 Mappers 30

5.5.2 Utils 31

5.6 Controllers 33

5.6.1 Login controller 33

5.6.2 Home controller 34

5.6.3 Prodotti controller 34

5.6.4 Categorie controller 37

5.6.5 Noleggi controller 39

5.6.6 Utenti controller 42

5.6.7 Archivio controller 46

5.6.8 Manuale controller 46

5.6.9 Logout controller 47

5.7 Views 47

5.7.1 Templates 47

5.7.2 Headers 48

5.7.3 Alberatura view 49

5.8 Middlewares 50

5.8.1 Log middleware 50

5.8.2 Auth middleware 51

5.8.3 Upload middleware 52

5.9 Routes 53

5.10 Sistema di notifica via email 54

5.10.1 Modulo mailer 54

5.10.2 Modulo mailer worker 55

5.11 Tests 58

6 Test 61

6.1 Protocollo di test 61

6.2 Risultati test 69

6.3 Mancanze/limitazioni conosciute 70

7 Consuntivo 71

8 Conclusioni 72

8.1 Sviluppi futuri 72

8.1.1 Integrazione di robot per il picking e la movimentazione 72

8.1.2 Sviluppo di sistemi di navigazione avanzati 72

8.1.3 Integrazione con il sistema di gestione del magazzino (WMS) 72

8.1.4 Implementazione di monitoraggio remoto e manutenzione programmata 72

8.1.5 Utilizzo di intelligenza artificiale e apprendimento automatico 72

8.1.6 Analisi delle statistiche sui prodotti 72

8.2 Considerazioni personali 73

8.2.1 Amir Kawsarani 73

8.2.2 Davide Branchi 74

8.2.3 Gioele Cappellari 75

9 Bibliografia 76

9.1 Sitografia 76

10 Glossario 77

11 Indice delle figure 78

12 Allegati 79

# Introduzione

## Informazioni sul progetto

Allievi: Gioele Cappellari, Davide Branchi, Amir Kawsarani

Docente: Michel Palucci

Scuola: CPT Trevano SAMT I3

Data inizio: 12.01.2024

Data consegna: 03.05.2024

## Abstract

*Managing a medium-scale warehouse alone, without tools that simplify the work, can become a very tedious process and allow room for error. For this project we’ve decided to create a program that allows you to manage all of this automatically and intuitively. With it a person can choose whether to manage the warehouse all alone or distribute the load among several people. Without it, coordinating multiple people and updating a list of available equipment would be nearly impossible and would take a lot of time. Instead, with the use of the Warehouse Management System, these problems are solved and the time spent making sure that all the equipment is present is pretty much non existent.*

## Scopo

### Scopi didattici

Gli obiettivi principali di questo progetto sono farci lavorare usando la metodologia Agile e prepararsi per il progetto del quarto anno.

Invece riguardo alla creazione del progetto volevamo migliorare a interagire con le banche dati e integrarci un sistema di codici QR, il tutto utilizzando NodeJs per sviluppare il nostro applicativo.

### Scopi operativi

Senza un sistema di gestione del magazzino, gestirne uno di medie/grandi dimensioni diventa molto laborioso e duraturo. Perciò lo scopo del progetto è proprio quello di riuscire semplificare la gestione dei magazzini implementando un gestionale che è in grado di:

* Aggiornare Database tramite la lettura dei codici QR.
* Visualizzare una lista di oggetti e gestirla.
* Implementare un sistema a noleggi con date di scadenza.
* Implementare un sistema a utenti con diversi permessi.

# Analisi

## Analisi del dominio

Un’azienda di cinematografia ha un magazzino che contiene tutto il materiale che viene usato per registrare. Essa per gestirlo conta a mano che il materiale sia presente. Stessa cosa per quando viene portato via per fare dei progetti.

Essendo che la quantità di materiale da gestire è elevata e c’è solamente una persona a gestire il magazzino, queste azioni richiedono troppo tempo e sono troppo soggette a errori.

Per semplificare la gestione del magazzino verranno usati dei codici QR sul materiale per poterli memorizzare e scansionare in maniera veloce.

Per la parte di presa del materiale per i progetti viene introdotto un sistema che ti permette di vedere che cos’è disponibile e che gestisce in maniera automatica la parte di “noleggio”.

Esso verrà utilizzato principalmente dal gestore del magazzino. Esso dovrà leggere il manuale di utilizzo per capire il pieno funzionamento dell’applicativo e ridurre il margine di errore.

## Analisi e specifica dei requisiti

In base alle direttive del cliente, sono state date le seguenti specifiche per il software:

|  |  |
| --- | --- |
| **Req-001** | |
| **Nome** | Presenza database |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Deve essere presente un database dove verranno memorizzati tutti i dati riguardo al noleggio del materiale e le credenziali di accesso al sito |

|  |  |
| --- | --- |
| **Req-002** | |
| **Nome** | Login |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L’utente necessita di un’interfaccia per accedere al sito.  Solo i dipendenti dell’azienda potranno fare l’accesso al sito. |
| **Sotto requisiti** | |
| **Req-001** | Si necessita la presenza di un database funzionante |

|  |  |
| --- | --- |
| **Req-003** | |
| **Nome** | Gestione articoli |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Il gestore del magazzino deve poter gestire i prodotti.  Quando viene registrato un nuovo prodotto, viene generato un codice QR |
| **Sotto requisiti** | |
| **Req-002** | L’utente deve aver fatto l’accesso al sito come “gestore magazzino” |

|  |  |
| --- | --- |
| **Req-004** | |
| **Nome** | Visualizzazione articoli |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L’utente deve poter visualizzare tutti gli articoli dell’azienda |
| **Sotto requisiti** | |
| **Req-002** | L’utente deve aver fatto l’accesso al sito |
| **Req-003** | Devono essere presenti degli articoli (già registrati) |

|  |  |
| --- | --- |
| **Req-005** | |
| **Nome** | Gestione categorie articoli |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Il gestore del magazzino deve poter gestire le categorie dei prodotti |
| **Sotto requisiti** | |
| **Req-002** | L’utente deve aver fatto l’accesso al sito come “gestore magazzino” |

|  |  |
| --- | --- |
| **Req-006** | |
| **Nome** | Visualizzazione categorie |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Il gestore del magazzino deve poter visualizzare tutte le categorie di prodotti |
| **Sotto requisiti** | |
| **Req-002** | L’utente deve aver fatto l’accesso al sito come “gestore magazzino” |
| **Req-005** | Devono essere presenti delle categorie (già registrate) |

|  |  |
| --- | --- |
| **Req-007** | |
| **Nome** | Visualizzazione informazioni singolo articolo |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L’utente deve poter visualizzare le informazioni di un determinato articolo |
| **Sotto requisiti** | |
| **Req-002** | L’utente deve aver fatto l’accesso al sito |
| **Req-003** | Devono essere presenti degli articoli |

|  |  |
| --- | --- |
| **Req-008** | |
| **Nome** | Ricerca articolo tramite QR |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L’utente deve poter scannerizzare il codice QR per visualizzare la scheda dell’articolo |
| **Sotto requisiti** | |
| **Req-002** | L’utente deve aver fatto l’accesso al sito |
| **Req-003** | Devono essere presenti degli articoli |

|  |  |
| --- | --- |
| **Req-009** | |
| **Nome** | Gestione noleggi |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L’utente deve poter gestire i noleggi degli articoli |
| **Sotto requisiti** | |
| **Req-002** | L’utente deve aver fatto l’accesso al sito |
| **Req-003** | Devono essere presenti degli articoli |

|  |  |
| --- | --- |
| **Req-010** | |
| **Nome** | Gestione utenti sito |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L’amministratore deve poter gestire gli utenti del sito, quindi deve poterli creare ed eliminare |
| **Sotto requisiti** | |
| **Req-002** | L’utente deve aver fatto l’accesso al sito come “amministratore” |

|  |  |
| --- | --- |
| **Req-011** | |
| **Nome** | Visualizzazione lista utenti |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L’amministratore deve poter visualizzare la lista di tutti gli utenti presenti nell’applicativo |
| **Sotto requisiti** | |
| **Req-002** | L’utente deve aver fatto l’accesso al sito come “amministratore” |
| **Req-013** | Devono essere presenti degli utenti |

|  |  |
| --- | --- |
| **Req-012** | |
| **Nome** | Stampa codice QR |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | Il gestore del magazzino deve poter stampare tramite etichettatrice i codici QR che fanno riferimento ad un articolo |
| **Sotto requisiti** | |
| **Req-002** | L’utente deve aver fatto l’accesso al sito come “gestore magazzino” |
| **Req-003** | Devono essere presenti degli articoli |
| **Req-006** | Visualizzazione QR dalle informazioni dell’articolo |

|  |  |
| --- | --- |
| **Req-013** | |
| **Nome** | Gestione inventario |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | Il gestore del magazzino deve poter gestire l’inventario controllando che ci sia tutto il materiale |
| **Sotto requisiti** | |
| **Req-002** | L’utente deve aver fatto l’accesso al sito come “gestore magazzino” |
| **Req-003** | Devono essere presenti degli articoli |

|  |  |
| --- | --- |
| **Req-014** | |
| **Nome** | Sistema di allerta per restituzione noleggio |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | Deve esserci un sistema di notifica tramite email che deve avvisare l’utente quando sta per scadere il noleggio.  Se il noleggio è scaduto, deve rinviare l’email all’utente e deve allertare il gestore del magazzino. |

**Spiegazione elementi tabella dei requisiti:**

**ID**: identificativo univoco del requisito

**Nome**: breve descrizione del requisito

**Priorità**: indica l’importanza di un requisito nell’insieme del progetto, definita assieme al committente. Ad esempio, poter disporre di report con colonne di colori diversi ha priorità minore rispetto al fatto di avere un database con gli elementi al suo interno. Solitamente si definiscono al massimo di 2-3 livelli di priorità.

**Versione**: indica la versione del requisito. Ogni modifica del requisito avrà una versione aggiornata.

Sulla documentazione apparirà solamente l’ultima versione, mentre le vecchie dovranno essere inserite nei diari.

**Note**: eventuali osservazioni importanti o riferimenti ad altri requisiti.

**Sotto requisiti**: elementi che compongono il requisito.

## Use case

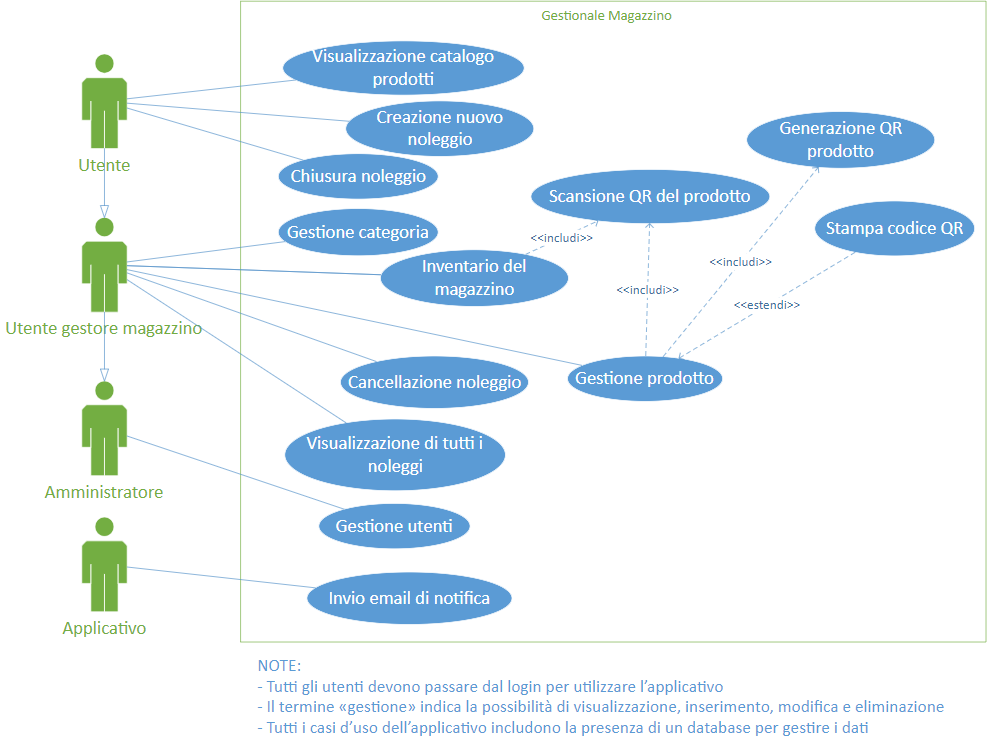


Figura 1 Use Case

Nell’applicativo l’attore principale sarà l’utente, che si specializza in utente gestore del magazzino e utente amministratore. Inoltre, sarà presente un altro attore che è l’applicativo stesso, e che avrà il compito di gestire il sistema di invio delle email di notifica per i noleggi in scadenza e scaduti.

Tutti gli utenti per poter utilizzare l’applicativo dovranno effettuare un login, i casi d’uso che rappresentano la differenza fra l’utente normale e il gestore sono la possibilità di gestire le categorie, effettuare l’inventario del magazzino, gestire i prodotti, visualizzare tutti i noleggi ed effettuare la chiusura forzata di un noleggio specifico.

L’utente amministratore invece avrà come possibilità in più la funzionalità di gestione degli utenti.

## Pianificazione

Per questo progetto abbiamo scelto di utilizzare una metodologia Agile, più precisamente abbiamo utilizzato il framework Scrumban.

Qui di seguito è presente il diagramma di Gantt con le attività pianificate in linea di massima e le relative tempistiche stimate:

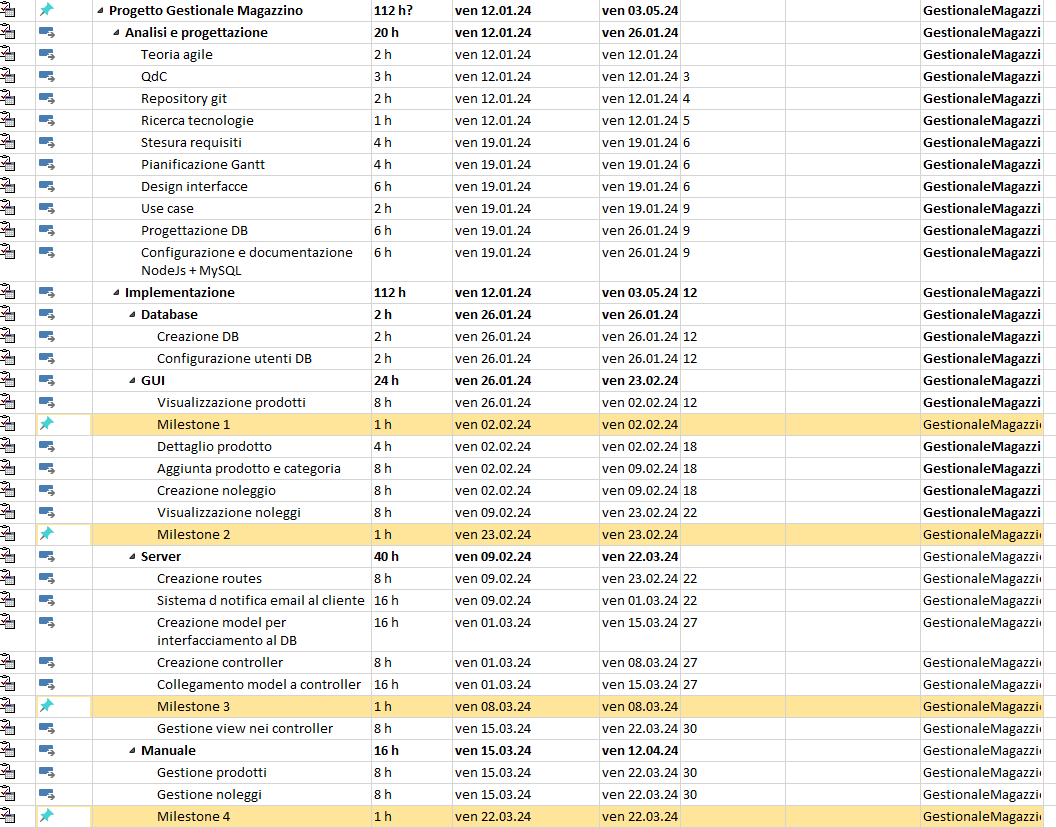


Figura 2 Diagramma di Gantt

Naturalmente la fase che in questo progetto richiederà più tempo sarà quella dell’implementazione. Abbiamo deciso di suddividere questa fase in quattro principali sottofasi: inizialmente abbiamo pianificato una fase di implementazione del database, poi abbiamo tutta la fase di creazione delle GUI, poi sarà presente l’implementazione del backend e infine la scrittura dei manuali per facilitare l’utilizzo da parte dell’utente finale.

## Analisi dei mezzi

### Software

* **MySQL 8.0.35**
* **Visual Studio Code 1.86.0**
* **NodeJS 20.11.0**
* **Npm 10.2.4**
* **Mozilla Firefox 103.0.1**
* **VirtualBox 7.0**: per hostare la macchina virtuale Ubuntu di sviluppo

I moduli di NodeJS che servono per il funzionamento dell’applicativo sono presenti in allegato nel file “moduli applicativo.docx”.

### Hardware

L’hardware utilizzato per lo sviluppo del software:

* **Una Webcam**: questa viene utilizzata per sviluppare e testare il funzionamento della lettura dei codici QR.
* **Un tablet**: esso serve per testare il responsive dell’applicativo e per simulare un possibile scenario dove il “gestore del magazzino” utilizza l’applicativo.
* **Un PC scolastico**: esso serve per sviluppare e testare l’applicativo. Il PC ha le seguenti specifiche:
  + Intel Core i7 7700
  + 16 GB di RAM
  + SSD 512 GB
  + Windows 10
* **I telefoni**: essi servono per testare il responsive dell’applicativo e per simulare un possibile scenario dove un “utente normale” utilizza l’applicativo e fa un noleggio.
* **Stampante**: quest’ultima serve per stampare i codici QR dei prodotti. Simula l’etichettatrice che dovrebbe stampare le etichette.

L’hardware il quale dovrà ospitare l’applicativo WEB:

* **Un server scolastico**: all’interno del server è hostata la macchina virtuale che utilizziamo per hostare l’applicativo e il database. La macchina virtuale ha le seguenti specifiche:
  + 2 GB di RAM
  + Ubuntu Server

# Progettazione

## Design dell’architettura del sistema

La struttura architetturale del nostro applicativo è molto semplice, infatti, essendo un'applicazione aziendale è destinata all'utilizzo esclusivo all'interno della rete dell’azienda. Per questo motivo non avremo bisogno di nessun hardware particolare.

L’infrastruttura è costituita da un singolo server, il quale si occupa di ospitare il database per la memorizzazione dei dati e l’applicativo web stesso.

Gli utenti dell’azienda potranno accedere all’applicativo utilizzando i PC aziendali, quindi attraverso la rete cablata interna, oppure tramite i dispositivi mobili o pc collegati alla rete wireless. Questi tipi di accesso sono stati progettati per facilitare la gestione dei noleggi per gli amministratori e i gestori; mentre per quanto riguarda gli altri utenti dell’azienda risulterà più comodo effettuare dei noleggi dato che possono utilizzare direttamente il telefono.

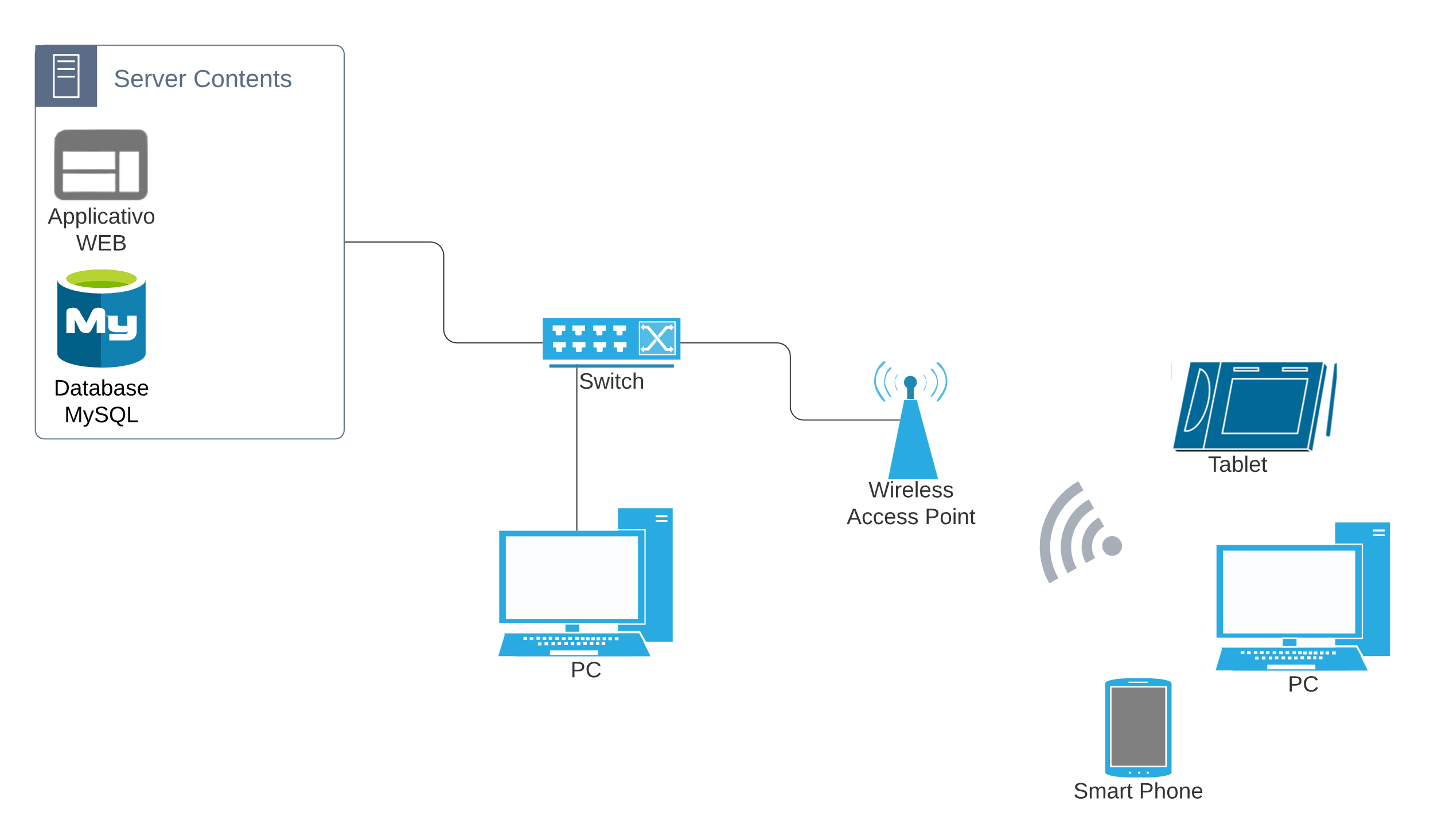


Figura 3 Architettura di sistema

## Design dei dati e database

### Diagramma ER

Prima versione:

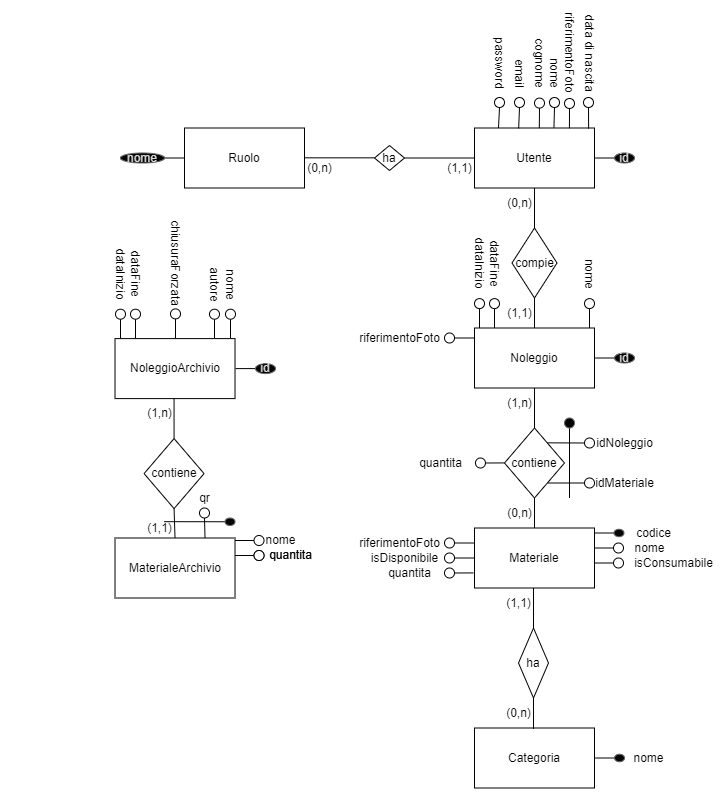


Figura 4 Prima versione diagramma ER

Dopo aver fatto la prima versione ci siamo accorti che non era necessario tenere la relazione tra “NoleggioArchivio” e “MaterialeArchivio” in base alle esigenze, perciò è stato modificato il diagramma ER e il database.

Seconda versione:

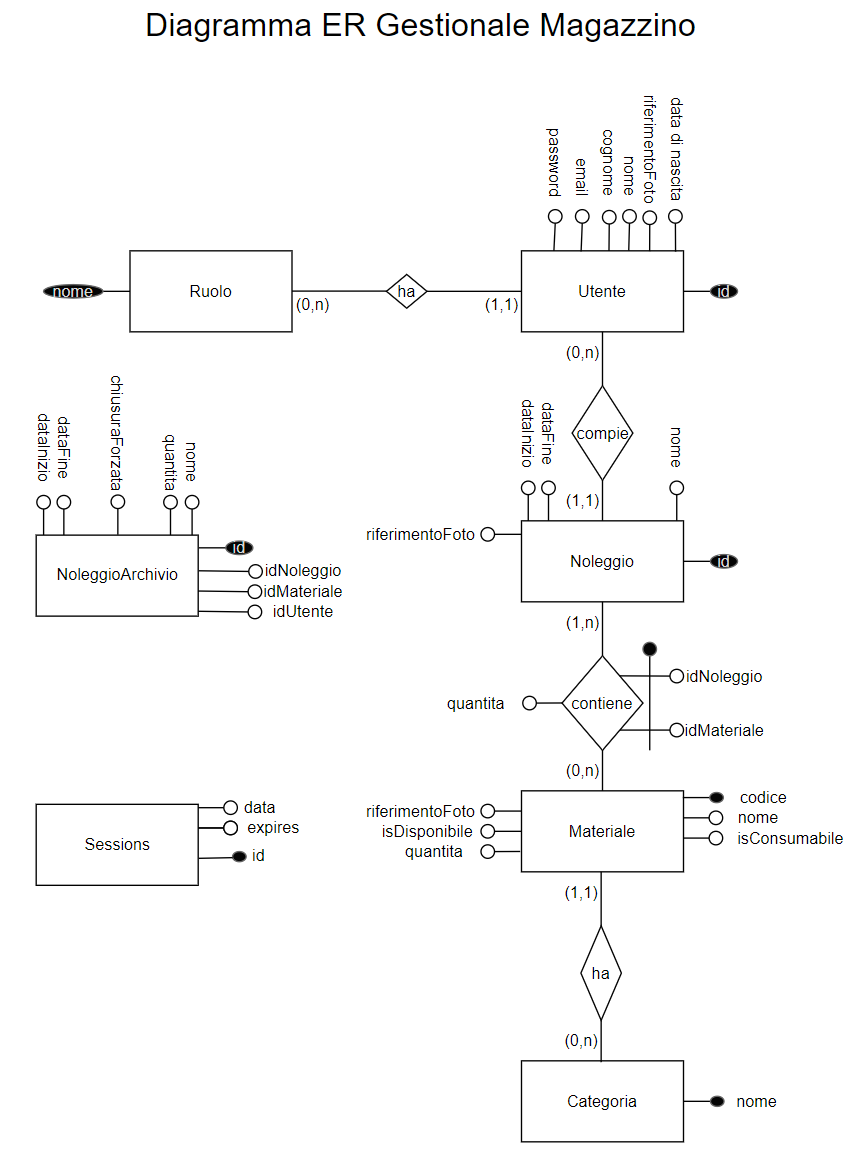


Figura 5 Seconda versione diagramma ER

### Descrizione delle Tabelle

**Ruolo**

* Contiene i ruoli degli utenti del sistema, come amministratori, gestori e utenti normali.
* Attributi: nome (chiave primaria).

**Sessions**

* Memorizza le sessioni degli utenti per il controllo dell'autenticazione.
* Attributi: id (chiave primaria), expires, data.

**Categoria**

* Rappresenta le categorie di materiali presenti nel magazzino.
* Attributi: nome (chiave primaria).

**Utente**

* Contiene informazioni sugli utenti del sistema.
* Attributi: id (chiave primaria), nome, cognome, riferimentoFoto, dataNascita, email, password, ruolo (chiave esterna verso la tabella Ruolo).

**Noleggio**

* Rappresenta i noleggi effettuati dagli utenti.
* Attributi: id (chiave primaria), nome, riferimentoFoto, dataInizio, dataFine, idUtente (chiave esterna verso la tabella Utente), chiusuraForzata.

**Materiale**

* Contiene informazioni sui materiali presenti nel magazzino.
* Attributi: codice (chiave primaria), nome, riferimentoFoto, quantita, isConsumabile, isDisponibile, categoria (chiave esterna verso la tabella Categoria).

**MaterialeNoleggio**

* Associa i materiali ai noleggi, indicando la quantità noleggiata.
* Attributi: idNoleggio (chiave esterna verso la tabella Noleggio), idMateriale (chiave esterna verso la tabella Materiale), quantita.

**Archivio**

* Conserva lo storico dei noleggi conclusi.
* Attributi: id (chiave primaria), nome, idNoleggio (valori presi dalla tabella ponte tra Noleggio e Materiale), idMateriale (valori presi dalla tabella ponte tra Noleggio e Materiale), idUtente, dataInizio, dataFine, quantita, chiusuraForzata.
* Chiusura forzata tiene conto se il noleggio è stato chiuso in maniera forzata (ha avuto delle perdite).
* Viene aggiornato usando una procedura perciò non è una vera e propria relazione.

### Descrizione delle Relazioni

La tabella:

* **Utente** ha una relazione con la tabella Ruolo attraverso l'attributo ruolo, che indica il ruolo dell'utente nel sistema.
* **Noleggio** è collegata alla tabella Utente tramite l'attributo idUtente, identificando l'utente che ha effettuato il noleggio.
* **Materiale** è collegata alla tabella Categoria tramite l'attributo categoria, specificando a quale categoria appartiene il materiale.
* **MaterialeNoleggio** collega i noleggi ai materiali noleggiati attraverso gli attributi idNoleggio e idMateriale, con una relazione molti a molti che indica che uno stesso materiale può essere noleggiato in più noleggi e un noleggio può contenere più materiali.
* **Archivio** registra i dettagli dei noleggi conclusi e archiviati, collegandoli tramite gli attributi idNoleggio e idMateriale alle rispettive tabelle di origine.

### Procedure e Trigger

**archivio\_update\_noleggio:**

Questo trigger viene attivato prima della cancellazione di una riga dalla tabella noleggio.

Prima della cancellazione, il trigger esegue la procedura descritta sopra per ciascuna riga nella tabella materialeNoleggio associata al noleggio che sta per essere eliminato.

L'uso di questo trigger assicura che, ogni volta che un noleggio viene eliminato, i dettagli dei materiali noleggiati vengano archiviati nella tabella archivio prima della cancellazione effettiva del noleggio stesso.

Ciò garantisce che lo storico dei noleggi e dei materiali noleggiati sia conservato anche dopo che i noleggi vengono cancellati dal sistema.

## Design delle interfacce

### Design iniziale

Per iniziare, abbiamo deciso di progettare le interfacce da mobile perché sono più semplici da fare ed è molto più intuitivo passare da un’interfaccia mobile a desktop. Poi rispetta di più le nostre esigenze di sviluppare un’applicazione comoda a livello di utente finale che utilizzerà principalmente il telefono.

#### Inventario

Queste è il design che era stato pattuito all’inizio. Una volta scelta la sezione inventario si vedrà una tabella con tutti i prodotti presenti nel magazzino e le rispettive informazioni. Se si clicca il pulsante di aggiunta di un nuovo prodotto, si deve aprire la GUI dove si inseriscono i dati del nuovo prodotto.

Mentre se si clicca sul bottone per visualizzare il dettaglio del prodotto, deve aprire una nuova interfaccia.

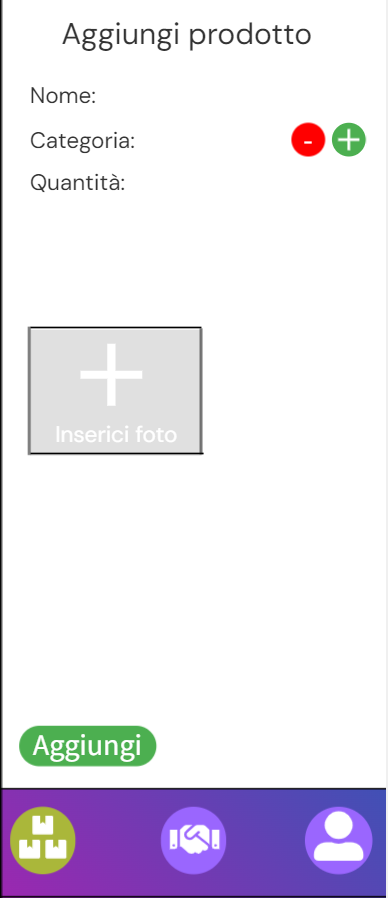
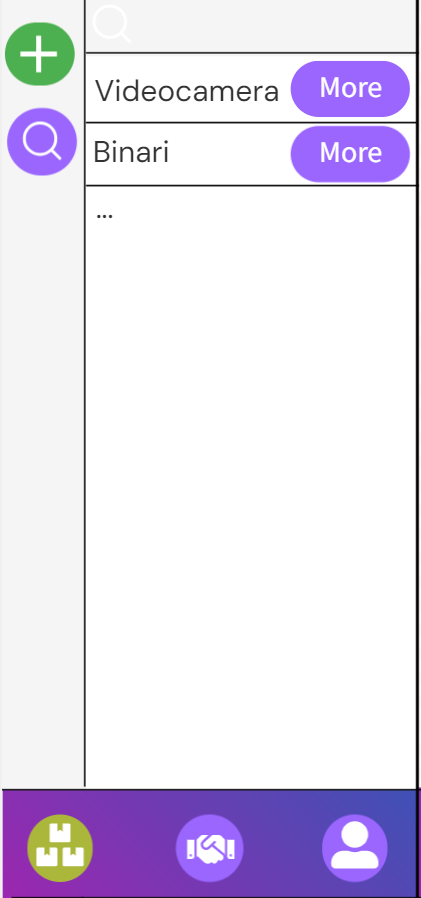


Figura 6 Design interfacce prodotti

#### Noleggi

Il design delle interfacce dei noleggi è simile a quello dell’inventario.

Quando si entra nella sezione dei noleggi, si ripresenterà la tabella simile a quella dell’inventario, solo che questa volta si vedranno tutti i noleggi.

Mentre se si clicca il bottone per vedere i dettagli dei noleggi si aprirà un'altra interfaccia che mostra tutti i dettagli del noleggio compresi i prodotti coinvolti in esso.

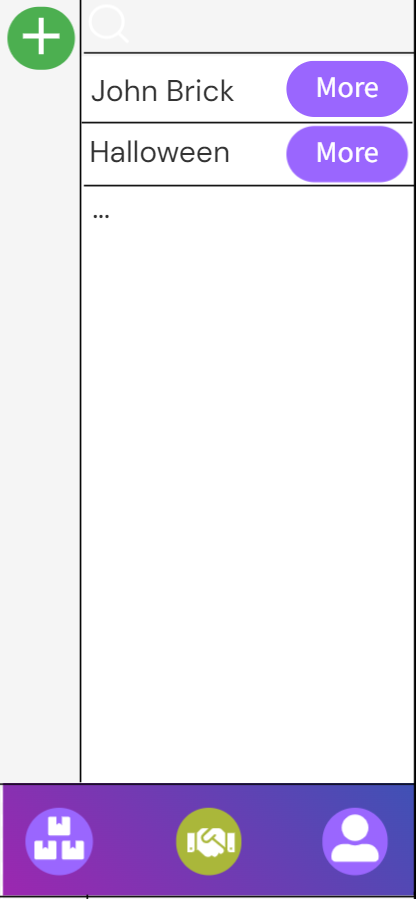


Figura 7 Desgin interfacce visualizzazione noleggi

Le altre due funzionalità sono quelle di “Aggiunta noleggio” e “Ritorna noleggio”.

L’interfaccia di aggiunta noleggio è abbastanza simile a quella dell’aggiunta di un prodotto, infatti ha i campi per inserire i dati necessari al noleggio e in più ha la funzionalità della scannerizzazione dei prodotti che si vogliono noleggiare.

L’interfaccia del ritorno di un noleggio invece è abbastanza semplice perché ha solo la funzionalità di scannerizzare gli oggetti del noleggio.

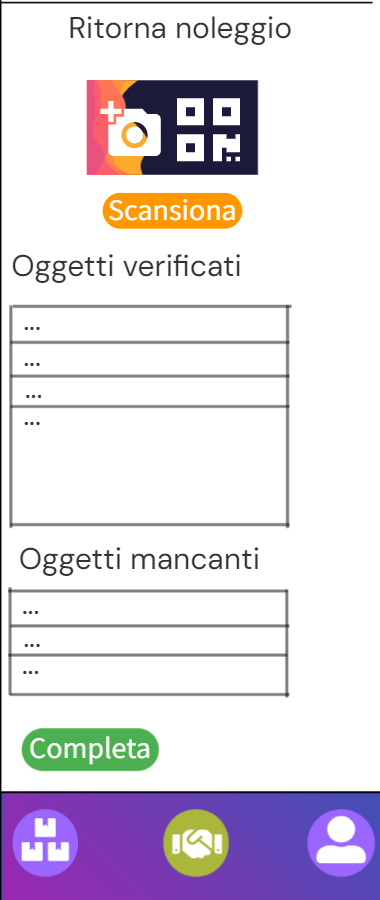


Figura 8 Design interfacce aggiunta e ritorno noleggio

#### Utente

Per il design delle interfacce dell’utente sono state riciclate quelle dell’inventario.

Infatti, come si può vedere dalle immagini seguenti, si ripresenta la medesima tabella presente anche in inventario.

Quando si clicca il bottone per vedere i dettagli si aprirà un’interfaccia che mostrerà i dettagli dell’utente con la lista dei noleggi aperti.

Mentre se si clicca il pulsante per aggiungere un utente si aprirà una nuova interfaccia per l’aggiunta dell’utente.

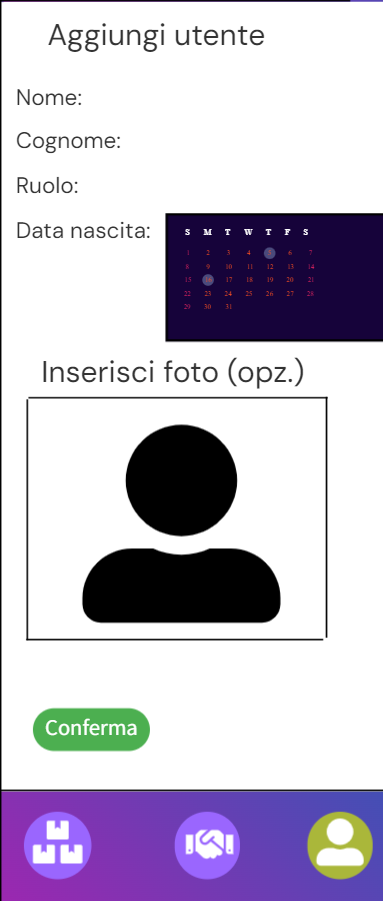
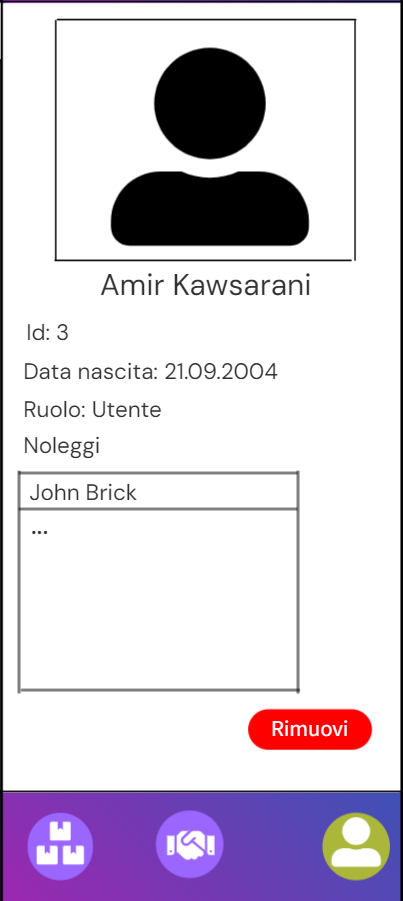
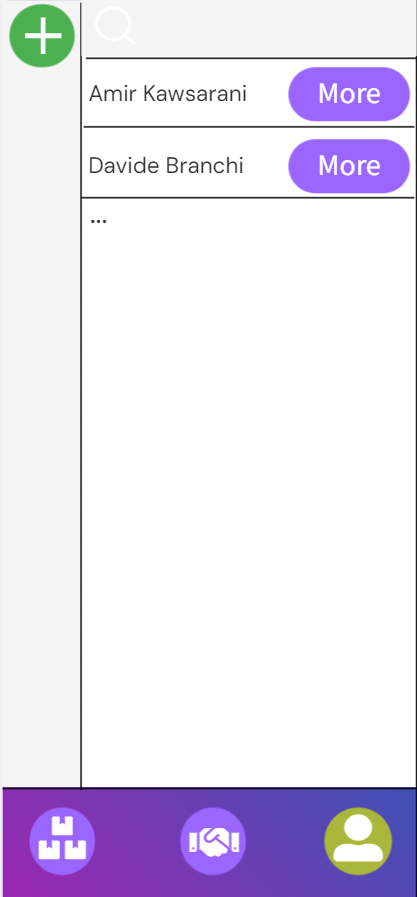


Figura 9 Design interfacce gestione utente

## Design procedurale

### Noleggio articoli

Il diagramma di flusso mostra il processo per noleggiare gli articoli. Per effettuare un noleggio, l'utente deve semplicemente scansionare tutti i codici QR presenti sugli articoli che desidera noleggiare. Nel frattempo, l'applicativo web si occuperà automaticamente del resto. Ogni volta che viene effettuata una scansione, l'applicativo aggiungerà l'oggetto al nuovo noleggio. Solo quando l'utente avrà terminato di scansionare tutti i codici QR, l'applicativo salverà nel database il nuovo noleggio.

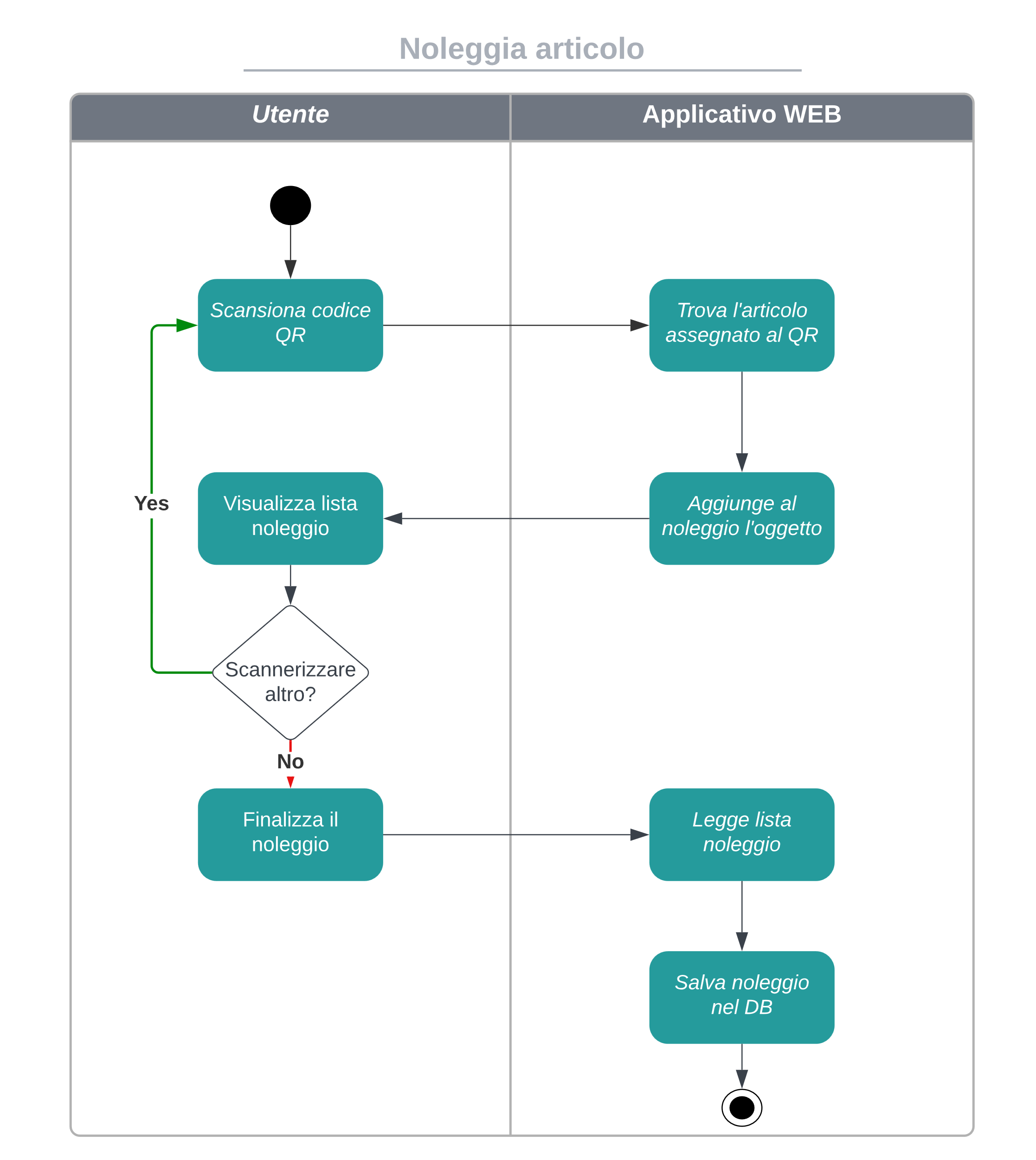


Figura 10 Diagramma di flusso noleggio articoli

### Chiusura noleggio

Il diagramma di flusso mostra il processo per chiudere un noleggio. Quando un utente deve restituire del materiale, deve prima selezionare il noleggio precedentemente creato, quindi cliccare sul pulsante "Ritorno materiale". Successivamente, deve scansionare tutti i codici QR presenti sul materiale preso dal magazzino. Dopo aver completato la scansione dei codici, deve cliccare sul pulsante "Chiusura noleggio". A questo punto, viene effettuato un controllo per verificare se tutto il materiale precedentemente preso è stato restituito. Se il materiale è al completo, il noleggio viene chiuso. Se manca del materiale, si presume che qualcosa sia stato perso. L’applicativo, quindi, verifica se l'utente che sta chiudendo il noleggio è un "gestore magazzino". In tal caso, il gestore può chiudere il noleggio tramite una chiusura forzata. Mentre se l'utente è un utente normale, viene richiesto di scansionare i codici QR non ancora scansionati. Nel caso l'utente desideri comunque chiudere il noleggio, deve contattare un gestore.

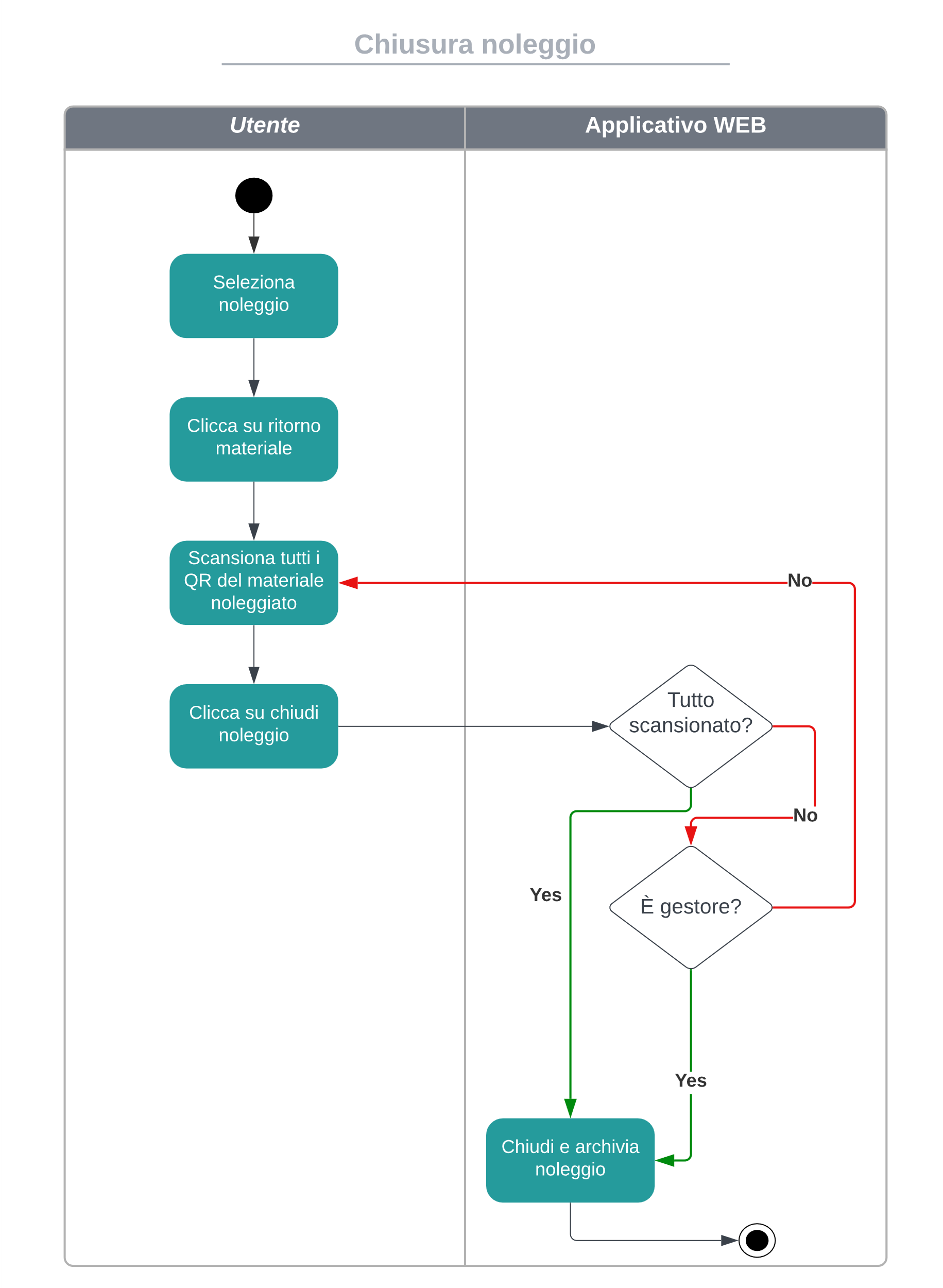


Figura 11 Diagramma di flusso chiusura noleggio

### Sistema di notifica noleggio in scadenza

Il diagramma di flusso mostra il processo del sistema di notifica per i noleggi che sono prossimi alla scadenza.

L’applicativo WEB verifica i noleggi che sono aperti e che sono a 24 ore dalla scadenza.

A questi utenti viene mandata un’email di notifica che li avverte della scadenza del loro noleggio.

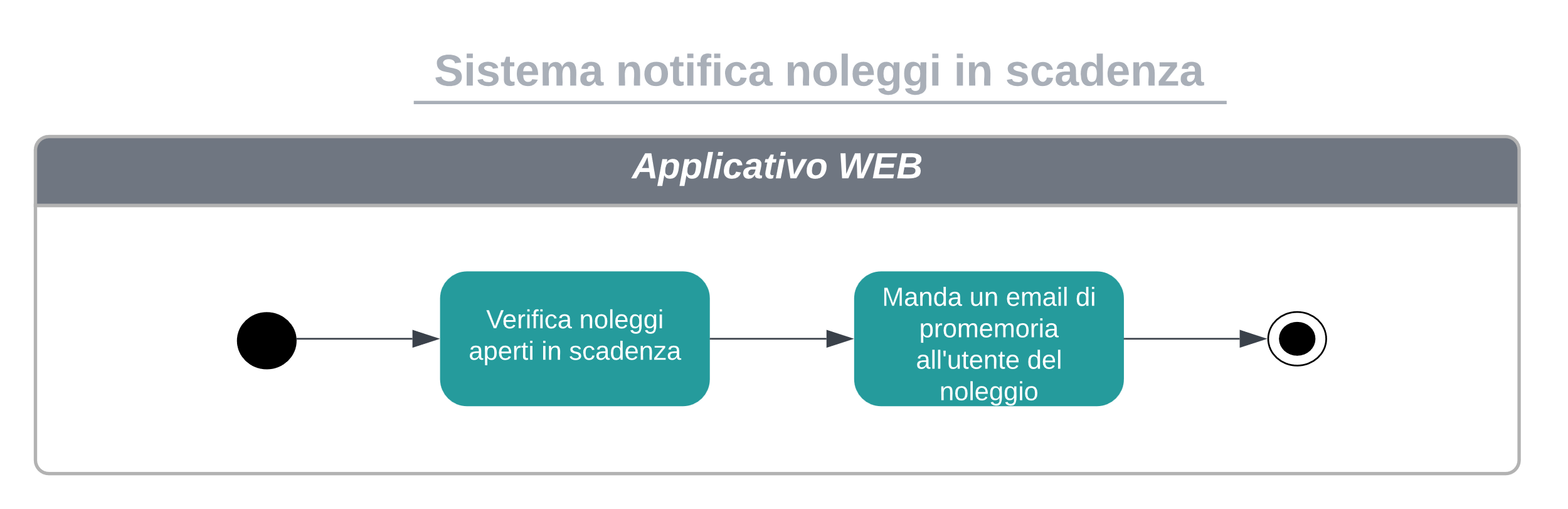


Figura 12 Diagramma di flusso sistema di notifica noleggio in scadenza

### Sistema di notifica noleggio scaduto

Il diagramma di flusso mostra il processo del sistema di notifica per i noleggi che sono scaduti.

L’applicativo WEB verifica i noleggi che sono aperti e che sono scaduti. A questi utenti viene inviata un’email che li avverte che il loro noleggio è scaduto e che devono riconsegnare il materiale. In più verrà mandata anche un email di allerta al gestore del magazzino.

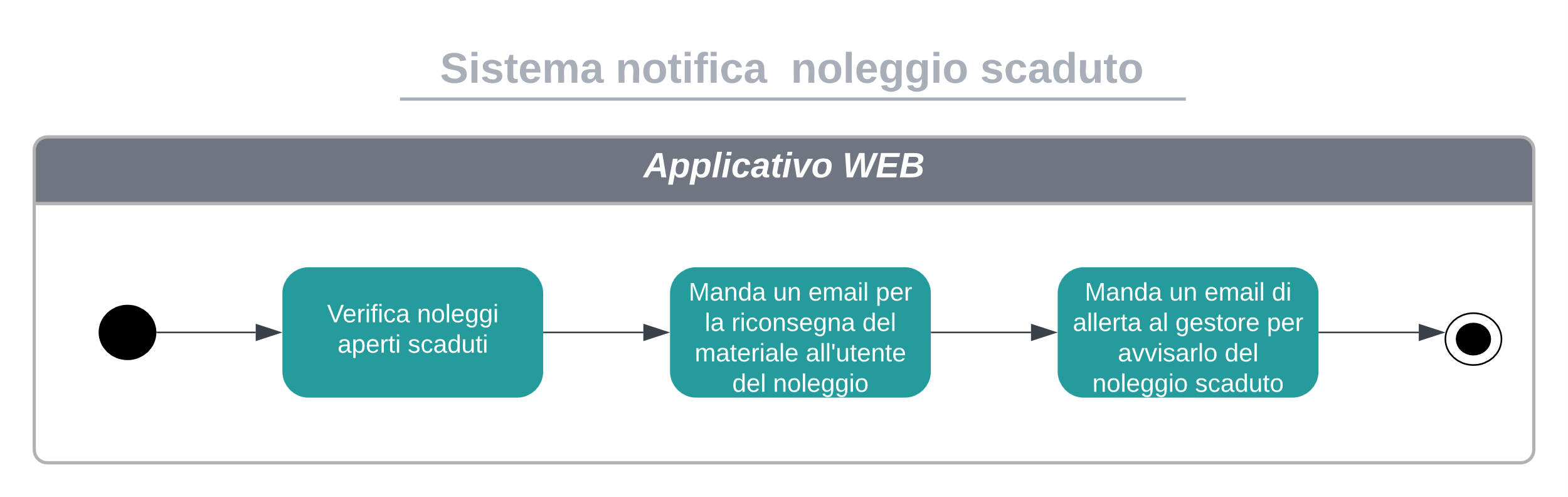


Figura 13 Diagramma di flusso sistema di notifica noleggio scaduto

# Implementazione

## Configurazione server

### Installazione MySQL

Sul server abbiamo deciso di installare il database MySQL per gestire tutti i dati dell’applicativo.

La versione installata è la 8.0.35.

I comandi eseguiti per l’installazione sono stati i seguenti:

Figura 14 Comandi installazione MySQL

sudo apt update

sudo apt install mysql-server

sudo mysql

ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql\_native\_password BY 'tmp\_pass';

exit

sudo mysql\_secure\_installation

n

y

y

y

y

### Installazione Nodejs

Per il funzionamento del nostro applicativo WEB è stato utilizzato Nodejs.

La versione installata è la LTS corrente, ovvero la 20.11.0.

I seguenti comandi servono per installare curl, Nodejs e NPM:

Figura 15 Comandi installazione Nodejs

sudo apt install curl

curl -sL [https://deb.nodesource.com/setup\_20.x -o nodesource\_setup.sh](https://deb.nodesource.com/setup_20.x%20-o%20nodesource_setup.sh)

sudo bash ./nodesource\_setup.sh

sudo apt-get install nodejs -y

### Installazione PM2 e script per l’auto deploy

Per gestire il processo dell’applicazione Nodejs abbiamo installato PM2 che è un process manager per gestire gli applicativi.

In questo modo ogni volta che il server subirà ad esempio un riavvio, l’applicativo tornerà online automaticamente.

Ecco i comandi utilizzati per l’installazione e la configurazione:

sudo npm install pm2@latest -g

sudo adduser app-user

sudo chown -R app-user:app-user /opt

su app-user

pm2 startup systemd

sudo env PATH=$PATH:/usr/bin /usr/lib/node\_modules/pm2/bin/pm2 startup systemd -u app-user --hp /home/app-user

sudo shutdown -r 0

sudo service pm2-app-user start

Figura 16 Comandi installazione PM2

Nella cartella /opt/scripts è presente uno script auto\_deploy.py che controlla se su GitHub ci sono state modifiche alla repository, successivamente esegue un pull delle modifiche e riavvia il processo dell’app.

Figura 17 Script auto\_deploy.py

#!/usr/bin/python3

import os

import subprocess

os.chdir("/opt/GestionaleMagazzino")

git\_process = subprocess.Popen(["git", "fetch", "--dry-run"], stdout=subprocess.PIPE, stderr=subprocess.PIPE)

git\_fetch\_result = git\_process.communicate()

if git\_fetch\_result != (b'', b''):

    subprocess.check\_output("git pull", shell=True)

    subprocess.check\_output("pm2 restart app", shell=True)

Infine per fare in modo che lo script venga eseguito continuamente, è stato creato un CronJob che esegue lo script ogni minuto.

Figura 18 CronJob auto\_deploy.py

\* \* \* \* \* python3 /opt/scripts/auto\_deploy.py

## Certificato TLS/SSL

### Generazione certificato

Per poter usufruire della fotocamera da Chrome era necessario avere HTTPS sulla pagina. Per questo abbiamo scelto di usare OpenSSL e generarci un self-signed certificate.

Ogni azione è stata fatta nella cartella certs.

Per installare OpenSSL, creare una chiave privata RSA (2048 bit) e generare un certificato CSR, bisogna utilizzare i seguenti comandi:

sudo apt update

sudo apt install openssl

openssl genrsa -out key.pem 2048

openssl req -new -key key.pem -out request.csr

openssl x509 -req -days 365 -in request.csr -signkey key.pem -out cert.pem

Figura 19 installazione OpenSSL, generazione chiave privata e certificato SSL

### Implementazione applicativo

Infine per far sì che il nostro applciativo NodeJS usufruisse del certificato abbiamo dovuto inserire la seguente parte nell’app.js:

Figura 20 implementazione certificato applicativo

//HTTPS

const options = {

key: fs.readFileSync('certs/key.pem'),

cert: fs.readFileSync('certs/cert.pem')

};

const server = https.createServer(options, app);

## Struttura applicativo

Per questo progetto abbiamo deciso di utilizzare il pattern MVC, la struttura di cartelle è la seguente:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Figura 21 Struttura di cartelle dell'applicativo

* **database:** Questa cartella contiene i moduli necessari ad effettuare la connessione al database mysql
* **certs:** In questa cartella è presente il certificato ssl self-signed e la relativa chiave privata
* **manuali:** In questa cartella sono contenuti i manuali utente che mostrerà l’applicativo
* **mail:** In questa cartella sono presenti i moduli che si occupano di gestire il sistema di notifica via email agli utenti e ai gestori per i noleggi in scadenza e scaduti
* **middlewares:** Questa cartella contiene tutti i middleware, ovvero le funzioni da cui passa la richiesta prima di raggiungere il controller, un esempio è il middleware per verificare che l’utente sia autenticato
* **public:** In questa cartella sono contenute tutte le immagini, script JS, file CSS e librerie lato client, che devono essere accessibili dal web
* **tests:** In questa cartella si trovano tutti gli unit test che vanno a testare il buon funzionamento dei models
* **models:** Questa cartella contiene tutti i moduli che si occupano di andare a comunicare con il database e ritornare i dati ai controller
* **routes:** In questa cartella sono presenti i moduli che gestiscono tutte le routes dell’applicativo, integrando i middlewares e i controller
* **controllers:** In questa cartella sono presenti tutti i controller, ovvero le funzioni che vanno a fare da tramite fra i models e le views
* **views:** Questa cartella contiene tutte le pagine dell’applicativo scritte in EJS
* **node\_modules:** Questa cartella viene utilizzata per salvare tutti i moduli scaricati tramite NPM da cui l’applicativo dipende

## Connessione al database

Per effettuare la connessione al database mysql abbiamo utilizzato la libreria *mysql2*, scaricato tramite NPM. Nel modulo che abbiamo creato si va a creare un pool di connessioni al database e viene utilizzata la funzione *promise()* per fare in modo che si possa andare a chiamare il database in maniera asincrona.

const db = mysql.createPool({

host: process.env.DB\_HOST,

port: process.env.DB\_PORT,

user: process.env.DB\_USER,

password: process.env.DB\_PASS,

database: process.env.DB\_SCHEMA

}).promise();

Figura 22 Connessione al database

## Models

Per le entità più importanti presenti nel database (materiale, utente, noleggio, categoria, archivio), abbiamo creato delle classi molto semplici che contengono tutti gli attributi necessari.

class Noleggio{

id;

nome;

riferimentoFoto;

dataInizio;

dataFine;

idUtente;

chiusuraForzata;

constructor(id, nome, riferimentoFoto, dataInizio, dataFine, idUtente, chiusuraForzata){

this.id = id;

this.nome = nome;

this.riferimentoFoto = riferimentoFoto;

this.dataInizio = dataInizio;

this.dataFine = dataFine;

this.idUtente = idUtente;

this.chiusuraForzata = chiusuraForzata;

}

}

Figura 23 Classe Noleggio (model)

Tutti gli altri models sono simili all’esempio soprastante della classe “Noleggio”, in ogni model sono presenti gli attributi che rispecchiano le colonne presenti nelle tabelle del database.

### Mappers

Per andare a prendere i dati dal database sono stati creati dei moduli “mapper” per ognuna delle entità più importanti citate in precedenza. Questi moduli contengono tutte le funzioni più utilizzate per prendere e gestire i dati del database. Le operazioni che eseguono queste funzioni vanno dalle classiche operazioni CRUD fino a funzioni specifiche per una funzionalità dell’applicativo.

Qui di seguito sono presenti le funzioni per prendere tutti i noleggi e tutti i materiali di un noleggio.

async function getAll(){

const [result] = await db.query("SELECT \* FROM noleggio");

let noleggi = [];

for(let item of result){

noleggi.push(new Noleggio(item.id, item.nome, item.riferimentoFoto, item.dataInizio, item.dataFine, item.idUtente, item.chiusuraForzata));

}

return noleggi;

}

Figura 24 funzione getAll() - NoleggioMapper

async function getMaterialeOfNoleggio(idNoleggio){

const materialeMapper = require("./materialeMapper");

const [result] = await db.query("SELECT \* FROM materialeNoleggio WHERE idNoleggio=?", [idNoleggio]);

//Array bidimensionale con i materiali e le loro relative quantità

//es: [[materiale, quantita], [materiale, quantita] ecc...]

let materiali = [];

for(let item of result){

let m = await materialeMapper.getByCodice(item.idMateriale);

materiali.push([m, item.quantita]);

}

return materiali;

}

Figura 25 funzione getMaterialeOfNoleggio() - NoleggioMapper

### Utils

Per andare a gestire le operazioni che non riguardano il database e che sono di vitale importanza per la validazione e manipolazione dei dati, sono stati creati dei model di supporto specifici.

#### DatastoreManager

Questo model si occupa di andare a gestire il datastore; quindi, si occupa di gestire i file che devono essere salvati sul server.

Il model ha le funzioni per controllare se un file è presente, per creare il percorso completo dove salvare il file e per eliminare un file dal datastore.

function createFullDatastorePath(path){

    return "public" + path;

}

Figura 26 funzione deleteDatastoreElement() - datasotreManager (model util)

async function deleteDatastoreElement(path){

    const fs = require("fs").promises;

    try{

        path = createFullDatastorePath(path);

        if(datastoreElementExists(path) && !path.includes("default")){

            await fs.unlink(path);

            return true;

        }else{

            return false

        }

    }catch(err){

        return false;

    }

}

Figura 27 funzione datastoreElementExists() - datastoreManager (model util)

function datastoreElementExists(path){

    const fs = require("fs");

    return fs.existsSync(path);

}

Figura 28 funzione createFulDatastorePath() - datastoreManager (model util)

#### QRGenerator

Per andare a generare il codice QR del prodotto è stato creato questo model molto basico, difatti contiene solo la funzione per generare il codice come stringa in base 64.

async function toBase64String(value){

    let qrData = await QRCode.toDataURL(value);

    return qrData;

}

Figura 29 funzione toBase64String() - QRGenerator (model util)

#### Sanitizer

Per pulire, sanificare e validare i dati in maniera più semplice e diretta si è creato il modulo sanitizer. Questo modulo contiene le funzioni per andare a pulire l’input che arriva direttamente dall’utente; il tutto per evitare problemi di sicurezza come inserimento di tag html malevoli e inserimento di caratteri malevoli e fastidiosi come ‘/’ e ‘\’.

Di seguito sono riportare le funzioni per sanificare l’input e per validare un email.

Figura 30 funzione sanitizeInput() - sanitizer (model util)

function sanitizeInput(value){

    if(typeof value === "string"){

        value = value.trim();

        value = value.replace(/[\/\\]/g, "");

        value = sanitizeHtml(value , {allowedTags: [], allowedAttributes: {}});

    }

    return value;

}

Figura 31 funzione validateEmail() - sanitizer (model util)

function validateEmail(email){

    return validator.isEmail(email);

}

## Controllers

Anche i controllers sono presenti per le entità più importanti, quest’ultimi fanno da tramite fra i models (che hanno il compito di andare a prendere i dati), e le views da mostrare all’utente. Inoltre, nei controller vengono sanificati tutti i dati e vengono anche fatti tutti i controlli per verificare che i dati inseriti dall’utente siano validi.

### Login controller

Il controller di login, come dice il nome stesso, si occupa di gestire le operazioni di autenticazione degli utenti.

Contiene solo due metodi: uno per fare il render della view e uno per effettuare l’accesso.

#### login()

La funzione per effettuare il login riceve i dati dal “body” della richiesta POST, poi fa la sanificazione e controlla che le credenziali mandate dall’utente siano corrette, in tal caso salva l’utente nella sessione e lo rindirizza alla home del sito, altrimenti mostra un errore.

async function login(req, res){

let {email, password} = req.body;

email = sanitizer.sanitizeInput(email);

password = sanitizer.sanitizeInput(password);

if (!sanitizer.validateEmail(email)){

let message = "Inserire un formato di email valido";

return res.status(401).render("login/index.ejs", { displayError: true, message: message });

}

// controlli utente

const user = await userMapper.getByEmail(email);

if (!user){

return res.status(401).render("login/index.ejs", { displayError: true, message: "Inserire delle credenziali valide" });

}

// controlli password

const passwordEqual = await bcrypt.compare(password, user.password);

if (!passwordEqual){

return res.status(401).render("login/index.ejs", { displayError: true, message: "Inserire delle credenziali valide" });

}

req.session.user = user;

req.session.save(function() {

return res.status(200).redirect("/home");

});

}

Figura 32 funzione login() - loginController

### Home controller

Una volta eseguito l’accesso, tutti gli utenti vengono rindirizzati alla home del sito; questa home è gestita dal medesimo controller, ovvero ‘homeController’.

Quest’ ultimo è uno dei più semplici, infatti ha una sola funzione che serve a fare il render della dashboard.

### Prodotti controller

Il controller dei prodotti è uno dei più importanti dell’intero applicativo perché si occupa di tutte le funzionalità relative alla visualizzazione, creazione, modifica ed eliminazione dei prodotti.

#### showAll()

La funzione si occupa di andare a prendere tutti i materiali tramite il modulo “materialeMapper” e poi renderizza la view per visualizzare tutti i prodotti.

#### loadViewAddProduct()

Quando viene chiamata questa funzione viene renderizzata una view per aggiungere un nuovo prodotto. Però prima di caricare la view, va a prendere tutte le categorie tramite il modulo “categorieMapper” che vengono poi passate alla view per essere renderizzate.

#### loadViewEditProduct()

La funzione per caricare la view di modifica del prodotto è più complessa delle 2 precedenti, questo perché oltre che a caricare le categorie, deve caricare anche il prodotto che deve essere modificato e per questo deve eseguire un controllo se il prodotto che si vuole modificare esiste veramente. Nel caso non esistesse carica una view di errore.

Figura 33 funzione loadViewEditProduct() - prodottiController

async function loadViewEditProduct(req, res) {

    const categorie = await categoriaMapper.getAll();

    const prodotto = await materialeMapper.getByCodice(sanitizer.sanitizeInput(req.params['codice']));

    if (prodotto === null){

        return res.status(404).render("\_templates/error.ejs", { error: { status: 404 } });

    }

    return res.status(200).render("prodotto/modifica.ejs", { session: req.session, categorie: categorie, prodotto: prodotto });

}

#### showProductDetails()

La funzione per visualizzare i dettagli di un prodotto è abbastanza complessa.

Sanifica il codice del prodotto da visualizzare ricevuto come parametro. Poi grazie al codice fa il controllo se esiste effettivamente un prodotto associato a quel codice, in caso contrario carica un errore. Dopodiché viene effettuato un controllo se nell’URL è passato come query il parametro ‘json’, quest’ultimo serve per mandare i dati del prodotto allo script lato client per creare un nuovo noleggio. Nel caso non fosse stato passato il parametro ‘json’, si prendono tutti i dati necessari tramite i mappers, ovvero: i noleggi nel quale è coinvolto quel prodotto, il QR per la stampa e la prossima disponibilità.

Infine, si carica la view passandogli tutti i dati recuperati in precedenza.

Figura 34 funzione showProductDetails() - prodottiController

const codice = sanitizer.sanitizeInput(req.params['codice']);

const product = await materialeMapper.getByCodice(codice);

if (product === null){

    return res.status(404).render("\_templates/error.ejs", { error: { status: 404 }});

}

if(req.query.json){

    return res.status(200).json(product);

}else{

    const noleggiId = await materialeMapper.getNoleggiIdByMaterialeCodice(codice);

    const noleggi = await noleggioMapper.getNoleggiByNoleggiId(noleggiId);

    const noleggiJson = [];

    for (let noleggio of noleggi){

        noleggiJson.push({

            data: noleggio,

            quantitaMateriale: await materialeMapper.getQuantitaMaterialeNoleggio(codice, noleggio.id)

        });

    }

    let qr = await QRGenerator.toBase64String(codice);

    const jsonData = {

        product: product,

        noleggi: noleggiJson,

        prossimaDisponibilita: product.isDisponibile ? "adesso" : materialeMapper.getDataDisponibilitaByNoleggi(noleggi),

        qrCode: qr,

        session: req.session

    }

    return res.status(200).render("prodotto/dettagli.ejs", jsonData);

}

#### addProduct()

La funzione per aggiungere un nuovo prodotto è una classica funzione di inserimento dati.

Riceve i dati dal “body” della richiesta POST, poi esegue la sanificazione, in seguito si controlla se l’utente gestore ha messo la spunta sul prodotto consumabile e se l’utente ha caricato la foto del prodotto, se non fosse così si imposta direttamente quella di default.

const nome = sanitizer.sanitizeInputTruncate(req.body.nome);

const quantita = toInt(sanitizer.sanitizeInput(req.body.quantita));

const categoria = sanitizer.sanitizeInputTruncate(req.body.categoria);

// controllo se caricato foto, altrimenti c'è quella di default

let foto = '/datastore/default.jpg';

if(req.body.fileUploadTry){

    if(!req.file){

        data.message = "Formato immagine non valido!";

        return res.status(400).render("prodotto/aggiunta.ejs", data);

    }else{

            foto = req.file.path.replace("public", "");

    }

}

let isConsumabile = false;

if (req.body.isConsumabile !== 'undefined'

&& req.body.isConsumabile === 'true'){

    isConsumabile = true;

}

Figura 35 funzione addProduct() sanificazione dati - prodottiController

Poi fa giusto due controlli: il primo è per la lunghezza del nome (non deve essere lungo più di 64 caratteri), mentre il secondo controlla che non esista già un prodotto con lo stesso nome e la stessa categoria. Quest’ultimo controllo è stato fatto in quanto si è deciso con il cliente che un prodotto è ritenuto uguale ad un altro solo se ha lo stesso nome e la stessa categoria.

if (nome.length > 64){

    data.message = "Il nome del prodotto è troppo lungo. Massimo 64 caratteri!";

    return res.status(400).render("prodotto/aggiunta.ejs", data);

}

const materiale = await materialeMapper.getMaterialeByNomeAndCategoria(nome, categoria);

if (materiale !== null){

    data.message = "Il prodotto esiste già";

    return res.status(400).render("prodotto/aggiunta.ejs", data);

}

Figura 36 funzione addProduct() controlli - prodottiController

Infine, viene fatta l’aggiunta del prodotto nel database tramite il mapper e si rindirizza l’utente nella pagina dei prodotti dove si farà visualizzare una conferma dell’aggiunta.

#### deleteProduct()

La funzione per l’eliminazione del prodotto riceve il codice del prodotto dal “body” della richiesta POST, lo sanifica e lo utilizza per andare ad eliminare il prodotto tramite il mapper.

Nel caso ci fosse stato un problema nel server durante l’eliminazione mostra una view di errore, altrimenti rindirizza l’utente alla pagina dei prodotti e mostra una conferma della eliminazione.

async function deleteProduct(req, res){

    const codice = sanitizer.sanitizeInput(req.body.codice);

    const isEliminato = await materialeMapper.deleteMateriale(codice);

    if (!isEliminato) {

        return res.status(500).render("\_templates/error.ejs", { error: { status: 500 } });

    }

    req.session.save(function(){

        req.session.displaySuccessMsg = "Prodotto eliminato con successo!";

        return res.status(200).redirect("/prodotti");

    });

}

Figura 37 funzione deleteProduct() - prodottiController

### Categorie controller

In questo controller sono presenti le funzioni per gestire la visualizzazione, l’aggiunta e l’eliminazione delle categorie.

Per far ciò sono presenti 4 funzioni:

#### showAll()

Questa funzione si occupa di prendere tutte le categorie presenti nel DB tramite il modulo “categorieMapper” e di mostrarle poi quando renderizza la view.

#### loadViewAddCategoria()

La funzione si occupa di fare il render della view di aggiunta della categoria. A differenza delle altre: essa non va a prendere i dati dal database.

#### addCategoria()

Questa funzione si occupa di eseguire l’aggiunta di una categoria al database.

Riceve il nome della categoria tramite il “body” della richiesta, gli fa la sanificazione e poi inizia ad eseguire i 2 controlli: il primo è quello della lunghezza del nome e poi gli fa il controllo che la categoria non sia già presente, quest’ultimo controllo non è “*case-sensitive*”. Nel caso la categoria esistesse, mostra un errore, altrimenti la aggiunge e poi rindirizza l’utente alla pagina per visualizzare le categorie.

async function addCategoria(req, res){

const nome = sanitizer.sanitizeInput(req.body.nome);

// controllo lunghezza nome

if (nome.length > 64){

const data = {

session: req.session,

displayError: true,

message: "Il nome della categoria è troppo lungo. Massimo 64 caratteri!"

}

return res.status(400).render("categoria/aggiunta.ejs", data);

}

// controllo che la categoria non esista

const categoria = await categoriaMapper.getByNome(nome);

if (categoria !== null){

const data = {

session: req.session,

displayError: true,

message: "La categoria esiste già!"

}

return res.status(400).render("categoria/aggiunta.ejs", data);

}

await categoriaMapper.insertCategoria(nome);

req.session.save(function(){

req.session.displaySuccessMsg = "Categoria aggiunta con successo!";

return res.status(200).redirect("/categorie");

});

}

Figura 38 funzione addCategoria() - categorieController

#### deleteCategoria()

Questa funzione serve per andare ad eliminare una categoria dal database.

Riceve il nome della categoria tramite il “body” della richiesta, gli fa la sanificazione e poi tramite il modulo “categorieMapper” la elimina.

L’unico controllo che esegue è se il server è riuscito ad eliminare una categoria, nel caso non fosse riuscito mostra una pagina di errore.

async function deleteCategoria(req, res){

const nome = sanitizer.sanitizeInput(req.body.nome);

// controllo se eliminata la categoria

const isEliminata = await categoriaMapper.deleteCategoria(nome);

if (!isEliminata) {

return res.status(500).render("\_templates/error.ejs", {error: {status: 500}});

}

req.session.save(function(){

req.session.displaySuccessMsg = "Categoria eliminata con successo!";

return res.status(200).redirect("/categorie");

});

}

Figura 39 funzione deleteCategoria() - categorieController

### Noleggi controller

In questo controller sono presenti le funzioni per gestire la visualizzazione, l’aggiunta e l’eliminazione dei noleggi.

Sono presenti 4 funzioni che si occupano unicamente di renderizzare le GUI scritte in EJS e trasmetterle all’utente.

#### showAll()

Questa funzione si occupa di renderizzare la GUI con tutti i noleggi aperti presenti nel database. Per prendere i noleggi utilizza il modulo “noleggioMapper”.

#### showAddNew()

Questa funzione si occupa di renderizzare la GUI per creare un nuovo noleggio.

#### addNew()

Questa funzione ha il compito di ricevere i dati del nuovo noleggio tramite metodo POST, in seguito si occupa di validare tutti i dati e di chiamare il noleggioMapper per la memorizzazione nel database  
  
I prodotti da inserire nel noleggio arrivano al controller sottoforma di stringa con all’interno un oggetto JSON da parsare. In seguito viene controllato che tutto il materiale sia presente nel database, e se è così viene creato il noleggio.  
Qui sotto è presente la parte di codice che si occupa di quanto appena descritto.

Figura 40 Controllo prodotti creazione nuovo noleggio

let prodottiNoleggio = sanitizer.sanitizeInput(req.body.prodottiNoleggio);

if(!prodottiNoleggio){

return res.status(400).render("noleggio/aggiunta.ejs", {session: req.session, displayError: true, message: "Errore inserimento noleggio: prodotti non trovati!"});

}

try{

prodottiNoleggio = JSON.parse(prodottiNoleggio);

}catch{

return res.status(400).render("noleggio/aggiunta.ejs", {session: req.session, displayError: true, message: "Errore inserimento noleggio: prodotti non trovati!"});

}

let prodottiNoleggioDb = [];

for(let arr of prodottiNoleggio){

let codice = sanitizer.sanitizeInput(arr[0]);

let qta = sanitizer.sanitizeInput(arr[2]);

let prodotto = await materialeMapper.getByCodice(codice);

if(!prodotto){

return res.status(400).render("noleggio/aggiunta.ejs", {session: req.session, displayError: true, message: "Errore inserimento noleggio: prodotti non trovati!"});

}

prodottiNoleggioDb.push([prodotto, qta]);

}

let insertedId = await noleggioMapper.insertNoleggio(nome, riferimentoFoto, dataInizio, dataFine, req.session.user.id, 0, prodottiNoleggioDb);

#### showNoleggioDetails()

Questa funzione si occupa di renderizzare la GUI con i dettagli di un noleggio. Per visualizzare i dettagli va anche a chiamare la funzione *getMaterialeOfNoleggio()* del noleggioMapper per poter visualizzare tutti i prodotti che fanno parte di quel noleggio.

Inoltre se l’utente non è un gestore o amministratore va a controllare che il noleggio sia suo per poter visualizzarne i dettagli.

Figura 41 Controllo dei permessi per visualizzare i dettagli di un noleggio

if(req.session.user.ruolo == "utente" && req.session.user.id != noleggio.idUtente){

return res.status(403).render("\_templates/error.ejs", { error: { status: 403 } });

}

#### showChiusura()

Questa funzione ha il compito di renderizzare la GUI per la chiusura di un noleggio. Come per la funzione precedente anche qui c’è lo stesso controllo dei permessi, infatti un utente che non è gestore o amministratore può andare a chiudere unicamente i noleggi di cui è lui l’autore.

#### closeNoleggio()

Questa funzione ha il compito di ricevere i dati della chiusura del noleggio tramite metodo POST, successivamente li valida e procede tramite il noleggioMapper a salvare la chiusura del noleggio nel database.  
  
Come per la funzione di creazione di un nuovo noleggio anche qua i dati dei prodotti che fanno parte del noleggio arrivano sottoforma di stringa con all’interno un oggetto JSON, che viene parsato e validato nello stesso modo.

Per controllare se la chiusura del noleggio è forzata oppure no, viene controllato l’end-point a cui arriva la richiesta: se è */chiudi* la chiusura non è forzata, se invece è */chiudi-force* la chiusura del noleggio verrà impostata come forzata.

Figura 42 Controllo per la chiusura forzata di un noleggio

//Controlla se la chiusura è forzata

let force = req.originalUrl.includes("chiudi-force");

let noleggioClose = await noleggioMapper.closeNoleggio(codice, force, prodottiNoleggioDb);

### Utenti controller

In questo controller sono presenti le funzioni per gestire la visualizzazione, l’aggiunta, l’eliminazione e la modifica degli utenti.

#### showAll()

Questa funzione si occupa di fare il render della pagina per visualizzare tutti gli utenti.

Per andare a prendere questi utenti dal database utilizza il modulo “utentiMapper”.

#### loadViewAddUtente()

Questa funzione serve a fare il render della pagina per visualizzare l’aggiunta degli utenti.

Alla view vengono passate i tipi di utente in maniera statica, ovvero un array che contiene i valori.

#### loadViewEditUtente()

Questa funzione serve a fare il render della view per la modifica di un utente.

Oltre ad andare a prendere le categorie dell’utente, bisogna andare a recuperare anche l’utente da modificare tramite il modulo “utentiMapper”.

L’unico controllo che viene effettuato è se l’utente esiste, altrimenti viene mostrata una pagina di errore.

Figura 43 funzione loadViewEditUtente() - utentiController

async function loadViewEditUtente(req, res){

const ruoli = ['utente', 'gestore', 'amministratore'];

const utente = await userMapper.getById(sanitizer.sanitizeInput(req.params.id));

if (utente === null){

return res.status(404).render("\_templates/error.ejs", { error: { status: 404 } });

}

return res.status(200).render("utente/modificaAmministratore.ejs", { session: req.session, ruoli: ruoli, user: utente });

}

#### loadViewEditProfilo()

La funzione per caricare la view di modifica del profilo è semplicissima, infatti si renderizza la view passandogli come utente quello della sessione.

Non vengono passate le categorie perché un utente non deve essere in grado di modificarsi la categoria.

#### showUserDetail()

La funzione carica la view dove mostra tutti i dettagli di un utente con i propri noleggi aperti.

Riceve l’id dell’utente come parametro dell’URL, poi lo sanifica e va a prendere l’utente tramite il modulo “utentiMapper”. Successivamente controlla che l’utente esista, se non fosse così caricherebbe una view di errore.

Figura 44 funzione showUserDetail() - utentiController

async function showUserDetail(req, res){

const userId = sanitizer.sanitizeInput(req.params.userId);

const user = await userMapper.getById(userId);

// se l'utente non esiste, carico la pagina di errore

if (user === null){

return res.status(404).render("\_templates/error.ejs", { error: { status: 404 } });

}

const data = {

session: req.session,

user: user,

noleggi: await noleggioMapper.getNoleggiOfUtente(user.id)

}

return res.status(200).render("utente/dettagli.ejs", data);

}

#### addNew()

Questa funzione che si occupa della creazione degli utenti è una delle più complicate del controller e dell’intero applicativo.

Riceve tutti i dati da memorizzare nel database dal “body” della richiesta, poi li sanifica tutti.

Dopo aver sanificato tutti i dati inizia a fare diversi controlli su tutti i campi; come primo controllo verifica se l’email passata è valida, successivamente controlla che quell’email non sia associata ad un altro account. Questo è stato fatto in quanto un email deve essere univoca e quindi non possono esistere più utenti con la stessa email. Dopo aver controllato l’email, si passa controllare il nome e in questo caso si controlla la lunghezza (max. 64 caratteri), dopo si passa al cognome e si controlla anche qui la lunghezza (anch’esso max. 64 caratteri), dopodiché si controlla la password che deve corrispondere a quella ripetuta e se anche in questo caso i controlli passano, allora si fa l’hash della password tramite il modulo “bcrypt”.

Questo hash è stato generato utilizzando un tipo di algoritmo che si chiama “*blowfish*” impostando i “*saltRounds*” a 10 (come sarebbe di default).

Infine dopo aver cifrato la password e aver sanificato e verificato i dati dell’utente, si utilizza il modulo “userMapper” per inserirlo nel database.

Nel caso anche solo un controllo fallisce, ricarica la view di aggiunta con un errore.

Di seguito sono mostrati i vari controlli che esegue prima della creazione di un utente:

// controllo se email valida

if (!sanitizer.validateEmail(email)){

data.message = "Inserire un indirizzo email valido";

return res.status(400).render("utente/aggiunta.ejs", data);

}

const user = await userMapper.getByEmail(email);

if (user !== null){

data.message = `Esiste già un utente con email: ${email}`;

return res.status(400).render("utente/aggiunta.ejs", data);

}

// controllo lunghezza nome

if (nome.length > 64){

data.message = "Il nome dell' utente è troppo lungo. Massimo 64 caratteri!";

return res.status(400).render("utente/aggiunta.ejs", data);

}

// controllo lunghezza cognome

if (cognome.length > 64){

data.message = "Il cognome dell' utente è troppo lungo. Massimo 64 caratteri!";

return res.status(400).render("utente/aggiunta.ejs", data);

}

// controlli sulle password

if (password !== passwordRipetuta){

data.message = "Le passowrd non coincidono!";

return res.status(400).render("utente/aggiunta.ejs", data);

}

Figura 45 controlli vari funzione addNew() - utentiController

#### editUtente()

Questa funzione serve per andare a modificare un utente.

Anche questa riceve i dati dal “body” della richiesta, li sanifica e poi esegue esattamente gli stessi controlli come per l’aggiunta utente, solo che in questa c’è un controllo in più, ovvero controlla se la password è stata cambiata. Questo è dovuto per fare in modo che non bisogna ricifrare la password ogni volta che viene chiamato questo metodo e non c’è il rischio di fare un hash dell’hash della password.

Di seguito viene mostrato il nuovo controllo:

Figura 46 controllo funzione editUtente() - utentiController

let passwordHashata = password;

if (password != user.password){

passwordHashata = await bcrypt.hash(password, 10);

}

#### editProfilo()

La funzione si occupa di andare a modificare il profilo di un utente.

Come quella della modifica di un utente, anche questa riceve i dati dal “body” della richiesta, poi li sanifica ed esegue esattamente gli stessi controller. Le uniche funzionalità che ci sono in più: quella della modifica dell’immagine profilo e il fatto che si è impossibilitati di modificarsi il ruolo.

Figura 47 controllo immagine funzione editProfilo() - utentiController

// controllo se utente ha caricato una foto, altrimenti c'è quella che aveva prima

let foto = req.session.user.riferimentoFoto;

if(req.body.fileUploadTry){

if(!req.file){

data.message = "Formato immagine non valido!";

return res.status(400).render("utente/modifica.ejs", data);

}else{

foto = req.file.path.replace("public", "");

await datastoreManager.deleteDatastoreElement(req.session.user.riferimentoFoto); //Elimina immagine vecchia da datastore

}

}

await userMapper.updateUser(id, nome, cognome, foto, nascita, email, passwordHashata, ruolo);

req.session.user = await userMapper.getById(id); // aggiorno utente nella sessione

#### deleteUser()

La funzione si occupa dell’eliminazione di un utente.

Riceve dal “body” della richiesta l’id dell’utente, lo sanifica e poi va a recuperare l’utente tramite il modulo “utentiMapper”. Dopo aver recuperato l’utente va a recuperare tutti i suoi noleggi tramite il modulo “noleggiMapper”, questo viene fatto perché è stato deciso che se un utente ha dei noleggi aperti: non può essere eliminato.

Nel caso avesse dei noleggi aperti, mostrerebbe un errore e si verrebbe rindirizzati alla pagina degli utenti con mostrato un errore, altrimenti andrebbe ad eliminare l’utente e anche la sua immagine di profilo per poi essere rindirizzati alla pagina degli utenti con un messaggio di conferma di eliminazione.

Per vedere il codice di implementazione dei controlli consultare la prossima pagina.

Figura 48 controlli funzione deleteUser() - utentiController

if(noleggi.length != 0){

req.session.save(function(){

req.session.displayErrorMsg = "Impossibile eliminare questo utente, è necessario chiudere prima tutti i noleggi di cui è autore!";

return res.status(400).redirect("/utenti");

});

return;

}

const isEliminato = await userMapper.deleteUser(userId);

if (!isEliminato){

return res.status(500).render("\_templates/error.ejs", {error: { status:500 } });

}

//Eliminazione dell'immagine del profilo dal datastore

await datastoreManager.deleteDatastoreElement(user.riferimentoFoto);

### Archivio controller

In questo controller sono presenti le funzioni per gestire le view della pagina dell’archivio.

#### showAll()

Questa funzione si occupa di renderizzare la GUI contenente tutti i noleggi di tutti gli utenti che sono stati archiviati (chiusi).

#### showDettagli()

Questa funzione si occupa di renderizzare la GUI dei dettagli di un noleggio archiviato.

### Manuale controller

Questo controller si occupa di gestire la funzionalità del manuale all’interno dell’applicativo.

È molto semplice, infatti contiene due funzioni che servono per mostrare la view dei manuali e per mandare i manuali.

#### loadViewManuale()

La funzione fa il render della view del manuale.

#### showManualeUtente()

La funzione serve per mostrare il manuale utente.

Riceve come parametro il nome del manuale pdf da mostrare, poi legge il contenuto del file e lo manda come risposta. Settando il “Content-Type” come “application/pdf”, il browser sa che gli arriverà un pdf e caricherà il file della risposta come tale.

Figura 49 funzione showManualeUtente() - manualeController

function showManualeUtente(nomeFile){

return (req, res) => {

const data = fs.readFileSync("./manuali/" + nomeFile);

res.contentType("application/pdf");

res.status(200).send(data);

}

}

### Logout controller

Il controller di logout, come dice il nome stesso, si occupa di gestire l’operazione di logout.

È molto semplice, infatti contiene solo un metodo che serve a fare il logout dall’applicativo.

#### logout()

La funzione è molto semplice, infatti, per eseguire il logout va a semplicemente richiamare il metodo “*destroy*” della sessione.

Richiamando questo metodo la sessione verrà distrutta e si verrà rindirizzati alla pagina di login.

## Views

Le views vengono renderizzate lato server grazie ad EJS, alle views vengono passate tramite i controllers le variabili contenenti gli oggetti da mostrare. In ogni view inoltre viene passata anche la variabile di sessione (*req.body.session*) in questo modo si può accedere dalla view ad esempio alle informazioni dell’utente e permette di utilizzare la sidebar in maniera automatica.

### Templates

Sono stati creati dei template di view che vengono chiamati in altri. Questi template sono stati creati per la navbar, la dashboard e per gli errori che servono per evitare pagine con accesso vietato (403), errore del server (500) oppure per pagina non trovata (404).

#### Navbar

Per semplificare la navigazione abbiamo deciso di mettere ogni funzionalità dell’applicativo sulla sidebar. Essa usa un font importato per creare le icone, è responsive ed ha delle animazioni che gli permettono di uscire ed entrare se premuta.

Abbiamo anche implementato un sistema che sfrutta EJS per capire dove ci si trova; infatti, colora di bianco in automatico la funzionalità selezionata.

Grazie a questo sistema è stato possibile isolarla come template in un file a parte.

### Headers

Per semplificare la creazione delle pagine abbiamo creato un template di header che viene richiamato sempre da ogni pagina all’interno dell’applicativo. Questa view rappresenta

la parte in comune di tutte le pagine, ovvero l’head.

Oltre ai meta tag, all’interno dell’header abbiamo importato anche la favicon e le librerie grafiche come bootstrap che ci servono in tutte le views.

Figura 50 Template EJS header

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta name="description" content="Progetto terzo anno secondo semestre">

<meta name="author" content="Gioele Cappellari, Amir Kawsarani, Davide Branchi">

<link rel="icon" href="/img/fav/favicon.ico" type="image/ico">

<!-- bootstrap -->

<link rel="stylesheet" href="/bootstrap/css/bootstrap.min.css">

<script src="/bootstrap/js/bootstrap.min.js"></script>

<!-- boxicons -->

<link rel="stylesheet" href="/boxicons/css/boxicons.css">

<script src="/boxicons/dist/boxicons.js"></script>

Per le altre parti di head più specifiche come librerie legate alle views, … sono stati creati altri template di header legate ad un tipo di view comune, ad esempio: quello delle tabelle e quello dei dettagli.

Per implementare la parte dell’header nelle view bisogna aggiungere la seguente parte di codice:

Figura 51 Implementazione header nelle view

<head>

  <title>Gestionale Magazzino - Dashboard</title>

  <%- include('../headers/header'); -%>

  <%- include('../headers/headerTablePage'); -%>

  <link rel="stylesheet" href="/css/dashboard.css">

</head>

### Alberatura view

Di seguito è riportato il contenuto delle cartelle delle view:

#### Archivio

La cartella archivio contiene la view che serve per visualizzare tutti i noleggi chiusi e archiviati e la view per vedere i dettagli di questi noleggi archiviati.

#### Categoria

La cartella categoria contiene la view di aggiunta di una categoria e quella per visualizzare tutte le categorie.

#### Dashboard

La cartella dashboard contiene la view della dashboard con tutte le sue funzionalità.

#### Login

La cartella login contiene la semplice view di login.

#### Manuale

La cartella manuale contiene la view per mostrare i link ai manuali.

#### Noleggio

La cartella noleggio contiene la view di visualizzazione dei noleggi, la view per visualizzare i dettagli di un noleggio, la view per creare un noleggio e la view per chiudere un noleggio.

#### Prodotto

La cartella prodotto contiene la view di visualizzazione di tutti i prodotti, la view per visualizzare i dettagli di un prodotto, la view per creare un prodotto e la view di modifica di un prodotto.

#### Utente

La cartella utente contiene la view per visualizzare tutti gli utenti, la view per visualizzare i dettagli di un utente, la view per l’aggiunta di un utente e la view per la rimozione di un utente.

## Middlewares

### Log middleware

In questo middleware viene configurato il modulo “morgan” che permette di loggare sulla console del server tutte le richieste che arrivano.

Il middleware è stato configurato per stampare le informazioni riguardanti l’indirizzo remoto della richiesta, l’utente, la data formattata con fuso orario Europe/Zurich, il metodo, l’URL, la versione HTTP, lo stato e la lunghezza della risposta.

Figura 52 Configurazione log middleware

morgan.token('date', (req, res, tz) => {

const date = new Date().toLocaleString('it-CH', { timeZone: 'Europe/Zurich' });

return date.replace(/,/, '');

});

const morganMiddleware = morgan(

':remote-addr - :remote-user [:date[clf]] ":method :url HTTP/:http-version" :status :res[content-length]',

{

stream: {

write: (message) => console.log(message.trim()),

},

}

);

### Auth middleware

Questo middleware si occupa di controllare l’autenticazione degli utenti e di gestire i relativi permessi per gli utenti normali, i gestori e gli amministratori

#### isAuthenticated()

Questa funzione controlla se l’utente è autenticato, per farlo va a verificare se la variabile *user* nella sessione è settata.

Se l’utente è autenticato la richiesta viene passata al middleware successivo tramite la funzione *next()* altrimenti la richiesta viene reindirizzata alla pagina di login.

Figura 53 Funzione isAuthenticated() - Auth middleware

function isAuthenticated (req, res, next) {

if (req.session.user){

next();

}else{

res.redirect("/login");

}

}

#### isGestore()

Questa funzione controlla se l’utente che è loggato è un gestore oppure se è un amministratore, per farlo va a controllare la variabile *user.ruolo* nella sessione.

Se l’utente è un gestore o amministratore la richiesta viene passata al middleware successivo tramite la funzione *next()* altrimenti viene renderizzata una pagina che comunica all’utente che non è autorizzato ad accedere a quella determinata schermata.

Figura 54 Funzione isGestore() - Auth middleware

function isGestore(req, res, next) {

if (req.session.user.ruolo === "gestore" || req.session.user.ruolo === "amministratore"){

next();

}else{

res.status(403).render("\_templates/error.ejs", { error: { status: 403 } });

}

}

#### isAmministratore()

Questa funzione controlla se l’utente che è loggato è un amministratore, per farlo va a controllare la variabile *user.ruolo* nella sessione.

Se l’utente è un amministratore la richiesta viene passata al middleware successivo tramite la funzione *next()* altrimenti viene renderizzata una pagina che comunica all’utente che non è autorizzato ad accedere a quella determinata schermata.

Figura 55 Funzione isAmministratore() - Auth middleware

function isAmministratore(req, res, next) {

if (req.session.user.ruolo === "amministratore"){

next();

}else{

res.status(403).render("\_templates/error.ejs", { error: { status: 403 } });

}

}

### Upload middleware

Questo middleware si occupa tramite il modulo “multer” di gestire l’upload delle immagini, ad esempio per i prodotti, noleggi o utenti.

In questo middleware è presente la funzione *uploadImg()* che riceve come parametro il percorso in cui va salvata l’immagine, configura multer ed effettua un controllo sul tipo di file che viene caricato.

fs.mkdirSync(uploadPath, {recursive: true});

const storage = multer.diskStorage({

destination: function (req, file, cb) {

cb(null, uploadPath);

},

filename: function (req, file, cb) {

let fileName = crypto.randomUUID() + path.extname(file.originalname);

cb(null, fileName);

}

});

Figura 56 Configurazione multer

Durante il controllo dell’estensione del file e del MIME type viene anche impostata una variabile (*req.body.fileUploadTry*) questa variabile viene utilizzata nei controller per poter controllare se c’è stato un tentativo di upload, poi tramite la variabile *req.body.file* si può capire se l’upload è andato a buon fine o no.

Figura 57 Controllo dell'immagine durante l'upload

const imageFilter = (req, file, cb) => {

//Variabile per poter controllare nel controller se c'è stato un

//tentativo di upload di un'immagine

req.body.fileUploadTry = true;

const acceptedExtensions = ['.jpg', '.jpeg', '.png'];

const acceptedMimeTypes = ['image/jpeg', 'image/png'];

if(acceptedMimeTypes.includes(file.mimetype.toLowerCase()) && acceptedExtensions.includes(path.extname(file.originalname.toLowerCase()))){

cb(null, true); //File valido

}else{

cb(null, false) //File non valido (non viene salvato)

// Nel controller req.body.file sarà undefined

}

};

## Routes

Le routes sono essenziali per il nostro applicativo perché ci permettono di mappare i vari URL, aggiungerci i permessi tramite middlewares e aggiungerci l’interattività tramite i controllers.

Abbiamo separato tutti i file di routes per ogni entità presente nell’applicativo, ad esempio: prodottiRoutes, utentiRoutes, …

Tutte queste rotte alla fine vanno a finire in una rotta “index” che sarà poi quella che verrà messa nell’app.js.

## Sistema di notifica via email

Per inviare le email di notifica è stato creato un indirizzo gratuito con il provider Infomainak: “[gestionale-magazzino@ik.me](mailto:gestionale-magazzino@ik.me)”.  
Lato server è stato utilizzato il modulo “nodemailer”, che grazie ad una configurazione molto semplice permette di inviare facilmente delle email.

### Modulo mailer

Questo modulo è presente nella cartella “mail”, all’interno viene configurato nodemailer con le impostazioni del server SMTP.

Figura 58 Configurazione nodemailer

const transporter = nodemailer.createTransport({

host: process.env.SMTP\_HOST,

port: process.env.SMTP\_PORT,

secure: true,

auth: {

user: process.env.SMTP\_USER,

pass: process.env.SMTP\_PASS,

}

});

Per l’invio delle email è stata creata la funzione *sendMail()* che riceve come parametri l’indirizzo email del destinatario, il soggetto e il testo (in formato html) della email da inviare.

Figura 59 Funzione sendMail()

async function sendMail(receiverAddr, subject, text){

try{

const message = await transporter.sendMail({

from: '"Gestionale Magazzino" <gestionale-magazzino@ik.me>',

to: receiverAddr,

subject: subject,

html: text

});

return message.response;

}catch(ex){

return ex.message

}

}

### Modulo mailer worker

In questo modulo è contenuta tutta la logica per controllare quali noleggi sono scaduti o in scadenza, da queste funzioni viene chiamata la funzione *sendMail(()* del modulo mailer per inviare le email di notifica.

#### initializeMailerWorker()

Questa funzione inizializza grazie al modulo “node-cron” un CronJob che si occupa di avviare le due funzioni per inviare le email dei noleggi in scadenza e dei noleggi scaduti.

Questa funzione viene chiamata all’interno del file principale *app.js* in modo che il mailer worker venga avviato quanto parte l’intero server.

Figura 60 Funzione initializeMailerWorker() - Mailer worker

function initializeMailerWorker(){

log("Initializing Mailer Worker");

// MailWorker programmato ogni giorno alle 00:00

cron.schedule('0 0 \* \* \*', async() => {

await sendNotificationEmails();

await sendExpiredNotificationEmails();

});

log("Mailer Worker Initialized");

}

#### log()

Questa funzione viene utilizzata per loggare sulla console del server tutto quello che fa il mailer worker durante la sua esecuzione.

Figura 61 Funzione log() - Mailer worker

function log(message){

let date = new Date();

date = date.toLocaleString("it-CH");

date = date.replace(/,/, '');

console.log(`[${date}]-[MAILERWORKER]: ${message}`);

}

#### sendNotificationEmails()

Questa funzione si occupa di inviare le email per i noleggi che sono in scadenza, ovvero a cui mancano 24 ore o meno prima di scadere.

Vengono controllati tutti i noleggi di ogni utente, e se la data di fine del noleggio è nelle prossime 24 ore viene inviata una email di notifica.

Per ogni email da inviare tramite EJS viene renderizzato un template che poi viene passato come stringa alla funzione *sendMail()* del modulo mailer.

Figura 62 Funzione sendNotificationEmails() - Mailer worker

async function sendNotificationEmails(){

log("Send notification emails task started");

let users = await userMapper.getAll();

for(let user of users){

let noleggi = await noleggioMapper.getNoleggiOfUtente(user.id);

let toNotify = [];

for(let noleggio of noleggi){

if(isDateWithin24Hours(noleggio.dataFine)){

toNotify.push(noleggio);

}

}

if(toNotify.length > 0){

let templateRendered = await ejs.renderFile("mail/\_emailTemplates/normal.ejs", {noleggi: toNotify, utente: user});

let mail = await mailer.sendMail(user.email, "Gestionale Magazzino - Notifica noleggi in scadenza", templateRendered)

log("Notification email sent to " + user.email + " | STATUS: " + mail)

}

}

log("Send notification emails task ended");

}

#### sendExpiredNotificationEmails()

Questa funzione si occupa di inviare agli utenti le email di notifica per i noleggi che sono scaduti da almeno 24 ore. Inoltre la funzione invia anche ai gestori e agli amministratori un report giornaliero con gli eventuali noleggi che sono scaduti e gli autori di quest’ultimi.

Figura 63 Invio del report giornaliero noleggi scaduti

let gestori = await userMapper.getAllGestoriAndAmministratori();

gestoriToNotify = await noleggioMapper.changeIdUtenteToNome(gestoriToNotify);

for(let gestore of gestori){

let templateRendered = await ejs.renderFile("mail/\_emailTemplates/gestoreReport.ejs", {noleggi: gestoriToNotify, utente: gestore});

let mail = await mailer.sendMail(gestore.email, "Gestionale Magazzino - Report noleggi scaduti", templateRendered)

log("Report email sent to " + gestore.email + " | STATUS: " + mail)

}



La funzionalità di invio delle email di notifica funziona unicamente su una rete in cui non è presente un server proxy.

Abbiamo provato ad utilizzare diversi servizi per l’invio delle email tramite API, ma nessuno di quelli testati funzionava correttamente

(Per maggiori informazioni consultare diario del 19.04.2024, Davide Branchi)

## Tests

Per testare i models dell’applicativo abbiamo utilizzato la libreria “jest”, abbiamo eseguito i test per ogni funzione di ogni model che si occupa di andare a prendere i dati dal database.

I file di test sono memorizzati nella cartella *test*s per eseguire i test viene utilizzato un database chiamato *gestionaleMagazzinoTEST*.

Per evitare che un test vada a compromettere il funzionamento di quello successivo (ad esempio eliminando dei dati che vengono utilizzati in un altro test), prima di ogni test viene avviata una transazione e alla fine del test viene fatto un rollback alla situazione iniziale.

Figura 64 Configurazione della transazione per eseguire i test

beforeEach(async () => {

    await db.query("START TRANSACTION");

});

afterEach(async () => {

    await db.query("ROLLBACK");

});

Qui di seguito è presente un esempio di come sono strutturati i test:

Figura 65 Test get per il materiale mapper

test("\_01\_getAll", async() => {

    let result = await materialeMapper.getAll();

    expect(result.length).toBeGreaterThan(0);

    for(let item of result){

        expect(item instanceof Materiale).toBeTruthy();

        expect(item.nome).toBeDefined();

    }

});

test("\_02\_getByCodice\_Exists", async() => {

    let result = await materialeMapper.getByCodice(4);

    expect(result).not.toBeNull();

    expect(result instanceof Materiale).toBeTruthy();

    expect(result.codice).toBe(4);

});

test("\_03\_getByCodice\_NotExists", async() => {

    let result = await materialeMapper.getByCodice(200);

    expect(result).toBeNull();

});

I test vengono avviati con il comando *npm run test*, i risultati sono i seguenti

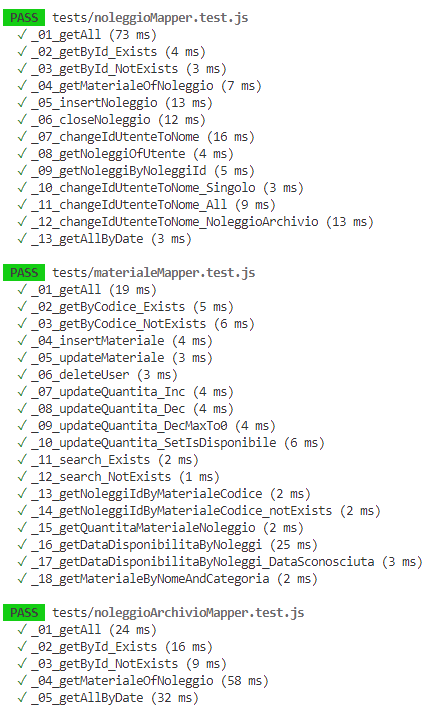


Figura 66 Risultati prima parte test jest

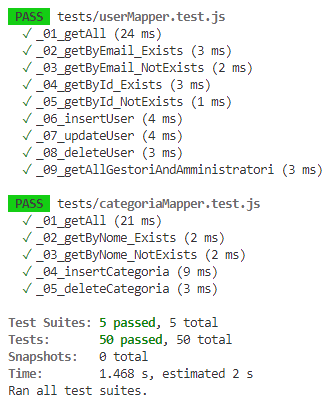


Figura 67 Risultati seconda parte test jest

# Test

## Protocollo di test

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-001  REQ-002 | **Nome** | Accesso utente |
| **Descrizione** | Si proverà a fare l’accesso all’applicativo con un utente normale. | | |
| **Prerequisiti** | Pagina di login completata | | |
| **Procedura** | 1. Cercare sul proprio browser l’indirizzo ip del server che ospita l’applicativo 2. Inserire le credenziali di login e cliccare sul pulsante login | | |
| **Risultati attesi** | Ci si dovrebbe ritrovare nella pagina home dell’applicativo con le funzionalità limitate a quelle di un utente normale. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-002  REQ-002 | **Nome** | Accesso utente con credenziali non corrette |
| **Descrizione** | Si proverà a fare l’accesso all’applicativo con un utente ma con delle credenziali non corrette o addirittura non esistenti. | | |
| **Prerequisiti** | Pagina di login completata | | |
| **Procedura** | 1. Cercare sul proprio browser l’indirizzo ip del server che ospita l’applicativo 2. Inserire delle credenziali di login errate e cliccare sul pulsante di login | | |
| **Risultati attesi** | Dovrebbe mostrare un messaggio di errore che le credenziali non sono corrette. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-003  REQ-002 | **Nome** | Accesso come gestore magazzino |
| **Descrizione** | Si proverà a fare l’accesso al sito come “gestore magazzino”. | | |
| **Prerequisiti** | Pagina di login completata | | |
| **Procedura** | 1. Cercare sul proprio browser l’indirizzo ip del server che ospita l’applicativo 2. Inserire delle credenziali di login di un “gestore magazzino” e cliccare sul pulsante di login | | |
| **Risultati attesi** | Ci si dovrebbe ritrovare nella pagina home dell’applicativo con le funzionalità di un “gestore magazzino”. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-004  REQ-002 | **Nome** | Accesso come amministratore |
| **Descrizione** | Si proverà a fare l’accesso al sito come “amministratore”. | | |
| **Prerequisiti** | Pagina di login completata | | |
| **Procedura** | 1. Cercare sul proprio browser l’indirizzo ip del server che ospita l’applicativo 2. Inserire delle credenziali di login di un “amministratore” e cliccare sul pulsante login | | |
| **Risultati attesi** | Ci si dovrebbe ritrovare nella pagina home dell’applicativo con le funzionalità di un “amministratore”. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-005  REQ-004 | **Nome** | Visualizzazione articoli |
| **Descrizione** | Si proverà ad andare a visualizzare gli articoli presenti nel database. | | |
| **Prerequisiti** | Pagina di visualizzazione articoli completata e devono essere presenti degli articoli nel DB | | |
| **Procedura** | 1. Effettuare l’accesso al sito con qualsiasi utente 2. Cliccare nel menu sull’icona per visualizzare gli articoli | | |
| **Risultati attesi** | Si dovrebbero visualizzare degli articoli. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-006  REQ-003 | **Nome** | Registrazione di un articolo |
| **Descrizione** | Si proverà a registrare un articolo nel database. | | |
| **Prerequisiti** | Pagina di registrazione articolo e pagina di visualizzazione articolo completate | | |
| **Procedura** | 1. Effettuare l’accesso al sito come “gestore magazzino” 2. Cliccare nel menu sull’icona per aggiungere un articolo 3. Compilare i campi necessari e cliccare sul pulsante per aggiungere l’articolo 4. Cliccare nel menu sull’icona per visualizzare gli articoli | | |
| **Risultati attesi** | Come ultimo articolo si dovrebbe visualizzare quello appena registrato. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-007  REQ-003 | **Nome** | Eliminazione di un articolo |
| **Descrizione** | Si proverà ad eliminare un articolo presente nel database. | | |
| **Prerequisiti** | Pagina di visualizzazione articolo completata | | |
| **Procedura** | 1. Effettuare l’accesso al sito come “gestore magazzino” 2. Cliccare nel menu sull’icona per visualizzare gli articoli 3. Cliccare sul pulsante per eliminare l’articolo | | |
| **Risultati attesi** | L’articolo eliminato dovrebbe sparire dalla lista. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-008  REQ-006 | **Nome** | Visualizzazione categorie |
| **Descrizione** | Si proverà ad andare a visualizzare le categorie di prodotti presenti nel database. | | |
| **Prerequisiti** | Pagina visualizzazione categorie completata e devono essere presenti delle categorie di prodotti nel DB | | |
| **Procedura** | 1. Effettuare l’accesso come “gestore magazzino” 2. Cliccare nel menu sull’icona per visualizzare le categorie | | |
| **Risultati attesi** | Si dovrebbero visualizzare delle categorie. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-009  REQ-005 | **Nome** | Registrazione di una nuova categoria |
| **Descrizione** | Si proverà ad andare a registrare una nuova categoria di prodotti. | | |
| **Prerequisiti** | Pagina visualizzazione categorie e pagina di registrazione categorie completate. | | |
| **Procedura** | 1. Effettuare l’accesso al sito come “gestore magazzino” 2. Cliccare nel menu sull’icona per aggiungere una categoria 3. Compilare i campi necessari e cliccare sul pulsante per aggiungere la categoria 4. Cliccare nel menu sull’icona per visualizzare le categorie | | |
| **Risultati attesi** | Si dovrebbe visualizzare la nuova categoria appena registrata. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-010  REQ-005 | **Nome** | Eliminazione categoria prodotti |
| **Descrizione** | Si proverà ad eliminare una categoria. | | |
| **Prerequisiti** | Pagina visualizzazione categorie completata. | | |
| **Procedura** | 1. Effettuare l’accesso al sito come “gestore magazzino” 2. Cliccare nel menu sull’icona per visualizzare le categorie 3. Cliccare sul pulsante di eliminazione della categoria | | |
| **Risultati attesi** | Dovrebbe sparire dalla lista la categoria appena eliminata. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-011  REQ-007 | **Nome** | Visualizzazione informazioni singolo articolo |
| **Descrizione** | Si proverà ad andare a visualizzare le informazioni dell’articolo selezionato. | | |
| **Prerequisiti** | Pagina scheda articolo e pagina visualizzazione articolo completate. | | |
| **Procedura** | 1. Effettuare l’accesso al sito con qualsiasi utente 2. Cliccare nel menu sull’icona per visualizzare gli articoli 3. Cliccare su un articolo | | |
| **Risultati attesi** | Si dovrebbero visualizzare le informazioni sull’articolo. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-012  REQ-008 | **Nome** | Visualizzazione scheda articolo tramite QR |
| **Descrizione** | Si proverà ad andare a visualizzare la scheda dell’articolo scansionando il suo codice QR. | | |
| **Prerequisiti** | Applicativo web funzionante. | | |
| **Procedura** | 1. Effettuare l’accesso al sito con qualsiasi utente 2. Cliccare nel menu sull’icona per scansionare un articolo 3. Scansionare il codice QR | | |
| **Risultati attesi** | Si dovrebbe visualizzare la scheda dell’articolo. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-013  REQ-009 | **Nome** | Noleggio articolo tramite codice QR |
| **Descrizione** | Si proverà ad andare a noleggiare un articolo scansionando il suo codice QR. | | |
| **Prerequisiti** | Applicativo web funzionante. | | |
| **Procedura** | 1. Effettuare l’accesso al sito con qualsiasi utente 2. Cliccare nel menu sull’icona per scannerizzare l’articolo 3. Scannerizzare il codice QR presente sull’articolo 4. Cliccare sul pulsante noleggia | | |
| **Risultati attesi** | Si dovrebbe noleggiare l’articolo, quindi deve risultare noleggiato nel DB e nella lista dei noleggi. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-014  REQ-009 | **Nome** | Ritorno di un articolo tramite codice QR |
| **Descrizione** | Si proverà a ritornare l’articolo noleggiato scansionando il codice QR. | | |
| **Prerequisiti** | Applicativo web funzionante. | | |
| **Procedura** | 1. Effettuare l’accesso al sito con qualsiasi utente 2. Cliccare nel menu sull’icona per ritornare l’articolo 3. Scannerizzare il codice QR presente sull’articolo 4. Cliccare sul pulsante di ritorno articolo | | |
| **Risultati attesi** | L’articolo dovrebbe sparire dai noleggi e lo si dovrebbe poter visualizzare nella lista di articoli presenti. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-015  REQ-011 | **Nome** | Visualizzazione lista utenti |
| **Descrizione** | Si proverà ad andare a visualizzare tutti gli utenti registrati nell’applicativo. | | |
| **Prerequisiti** | Pagina visualizzazione utenti completata e presenti utenti nel DB. | | |
| **Procedura** | 1. Effettuare l’accesso al sito come “amministratore” 2. Cliccare nel menu sull’icona per visualizzare gli utenti | | |
| **Risultati attesi** | Si dovrebbero visualizzare gli utenti registrati nell’applicativo. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-016  REQ-010 | **Nome** | Registrazione di un nuovo utente |
| **Descrizione** | Si proverà a registrare un nuovo utente. | | |
| **Prerequisiti** | Pagina di visualizzazione utenti e pagina registrazione utenti completate. | | |
| **Procedura** | 1. Effettuare l’accesso al sito come “amministratore” 2. Cliccare nel menu sull’icona per registrare gli utenti 3. Compilare i campi necessari e cliccare sul pulsante per aggiungere l’utente 4. Spostarsi nella pagina di visualizzazione utenti | | |
| **Risultati attesi** | Si dovrebbe visualizzare l’utente appena registrato. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-017  REQ-010 | **Nome** | Eliminazione utente |
| **Descrizione** | Si proverà ad eliminare un utente. | | |
| **Prerequisiti** | Pagina visualizzazione utenti completata e presenti utenti nel DB. | | |
| **Procedura** | 1. Effettuare l’accesso al sito come “amministratore” 2. Cliccare nel menu sull’icona per visualizzare gli utenti 3. Cliccare sul pulsante per eliminare l’utente | | |
| **Risultati attesi** | L’utente dovrebbe essere eliminato e quindi anche sparito dalla lista. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-018  REQ-012 | **Nome** | Stampa etichetta con codice QR articolo |
| **Descrizione** | Si proverà a stampare l’etichetta con il codice QR sull’etichettatrice. | | |
| **Prerequisiti** | Applicativo web funzionante. | | |
| **Procedura** | 1. Effettuare l’accesso al sito come “gestore magazzino” 2. Cliccare nel menu sull’icona per visualizzare gli articoli 3. Cliccare su un articolo per visualizzarlo 4. Cliccare sul pulsante stampa QR 5. Scansionare l’etichetta appena stampata | | |
| **Risultati attesi** | Quando si scansiona l’etichetta stampata, si dovrebbe visualizzare l’articolo corrente. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-019  REQ-013 | **Nome** | Inventario articoli |
| **Descrizione** | Si proverà a fare il controllo dell’inventario. | | |
| **Prerequisiti** | Applicativo web funzionante. | | |
| **Procedura** | 1. Effettuare l’accesso al sito come “gestore magazzino” 2. Cliccare nel menu sull’icona per fare l’inventario 3. Scannerizzare tutti i codici degli articoli a parte uno | | |
| **Risultati attesi** | Dovrebbe dire quale articolo non è stato scannerizzato, quindi quale articolo manca. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-020  REQ-014 | **Nome** | Allerta giorno prima della scadenza del noleggio |
| **Descrizione** | Si proverà a testare il sistema di allerta del giorno prima della restituzione del prodotto. | | |
| **Prerequisiti** | Applicativo web e sistema di allerta funzionante. | | |
| **Procedura** | 1. Effettuare l’accesso al sito con qualsiasi utente 2. Noleggiare un articolo con scadenza a due giorni di distanza 3. Aspettare il giorno dopo e controllare le e-mail | | |
| **Risultati attesi** | Si dovrebbe ricevere un’e-mail che ti avvisa che il prodotto deve essere riconsegnato. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-021  REQ-014 | **Nome** | Allerta giorno della scadenza del noleggio |
| **Descrizione** | Si proverà a testare il sistema di allerta del giorno della restituzione del prodotto. | | |
| **Prerequisiti** | Applicativo web e sistema di allerta funzionante. | | |
| **Procedura** | 1. Effettuare l’accesso al sito con qualsiasi utente 2. Noleggiare un articolo con scadenza il giorno dopo 3. Aspettare il giorno dopo e controllare le e-mail | | |
| **Risultati attesi** | Dovrebbe essere arrivata un’e-mail che avvisa che il prodotto deve essere riconsegnato e dovrebbe essere stato allertato anche il gestore del magazzino. | | |

## Risultati test

La seguente tabella contiene i vari test case eseguiti all’applicativo e il loro esito finale.

Le comprove si trovano in allegato nel file: “Test case.docx”.

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Risultato ottenuto** | **Stato** |
| TC-001 | L’utente riesce ad accedere all’applicativo e vengono visualizzate unicamente le funzionalità che ha il permesso di utilizzare. | Passato |
| TC-002 | Viene visualizzato un messaggio di errore che comunica che le credenziali utilizzate per accedere sono errate. | Passato |
| TC-003 | Il gestore del magazzino riesce ad accedere all’applicativo e vengono visualizzate unicamente le funzionalità che ha il permesso di utilizzare. | Passato |
| TC-004 | L’amministratore riesce ad accedere all’applicativo e vengono visualizzate unicamente le funzionalità che ha il permesso di utilizzare. | Passato |
| TC-005 | Si riescono a visualizzare tutti gli articoli presenti nella pagina del catalogo. | Passato |
| TC-006 | È possibile registrare un nuovo articolo. | Passato |
| TC-007 | L’articolo viene eliminato e sparisce dal catalogo. | Passato |
| TC-008 | Si riesce a visualizzare tutte le categorie | Passato |
| TC-009 | La categoria viene creata ed è possibile visualizzarla nel catalogo. | Passato |
| TC-010 | La categoria viene eliminata e non è più visibile nel catalogo. | Passato |
| TC-011 | Si riesce a visualizzare i dettagli di un articolo. | Passato |
| TC-012 | Scansionando un prodotto si riesce ad accedere alla pagina con i suoi dettagli. | Passato |
| TC-013 | Si riesce a noleggiare un articolo tramite codice QR, e in seguito il noleggio è visibile nel catalogo. | Passato |
| TC-014 | Il codice QR viene scansionato e il noleggio viene rimosso dal catalogo, la quantità del prodotto viene aggiornata. | Passato |
| TC-015 | Si possono visualizzare tutti gli utenti nel catalogo. | Passato |
| TC-016 | Si riesce a registrare un nuovo utente, ed è possibile vederlo nel catalogo di tutti gli utenti. | Passato |
| TC-017 | L’utente viene eliminato, e non è più visibile nel catalogo. | Passato |
| TC-018 | È possibile stampare un foglio con i codici QR del prodotto, ed è anche possibile scansionarli, tuttavia la stampa su etichettatrice non è stata implementata. | Parzialmente passato |
| TC-019 | Non è possibile effettuare l’inventario dei prodotti.  Questa funzionalità non è stata implementata, di conseguenza il test non è passato. Per ulteriori informazioni consultare il capitolo “Mancanze”. | Fallito |
| TC-020 | L’utente riceve l’email che avvisa che il noleggio è in scadenza, i gestori e amministratori ricevono il report con nessun noleggio scaduto. | Parzialmente passato |
| TC-021 | L’utente riceve una email di allerta con il noleggio scaduto, gli amministratori e gestori ricevono il report che contiene il noleggio scaduto dell’utente | Parzialmente passato |

I Test case TC-020 e TC-021 sono parzialmente passati perché l’invio delle email funziona solo su una rete in cui non è presente un server proxy. Per maggiori informazioni consultare il capitolo [5.10 Sistema di notifica via email](#_Sistema_di_notifica).

## Mancanze/limitazioni conosciute

Attualmente non è stato implementato il sistema per effettuare l’inventario del magazzino, e nemmeno la stampa del codice QR tramite etichettatrice, per mancanza di tempo.

Il sistema di notifica vie email è limitato a funzionare su una rete senza server proxy.

# Consuntivo

Di seguito è presente il diagramma di GANTT consuntivo:

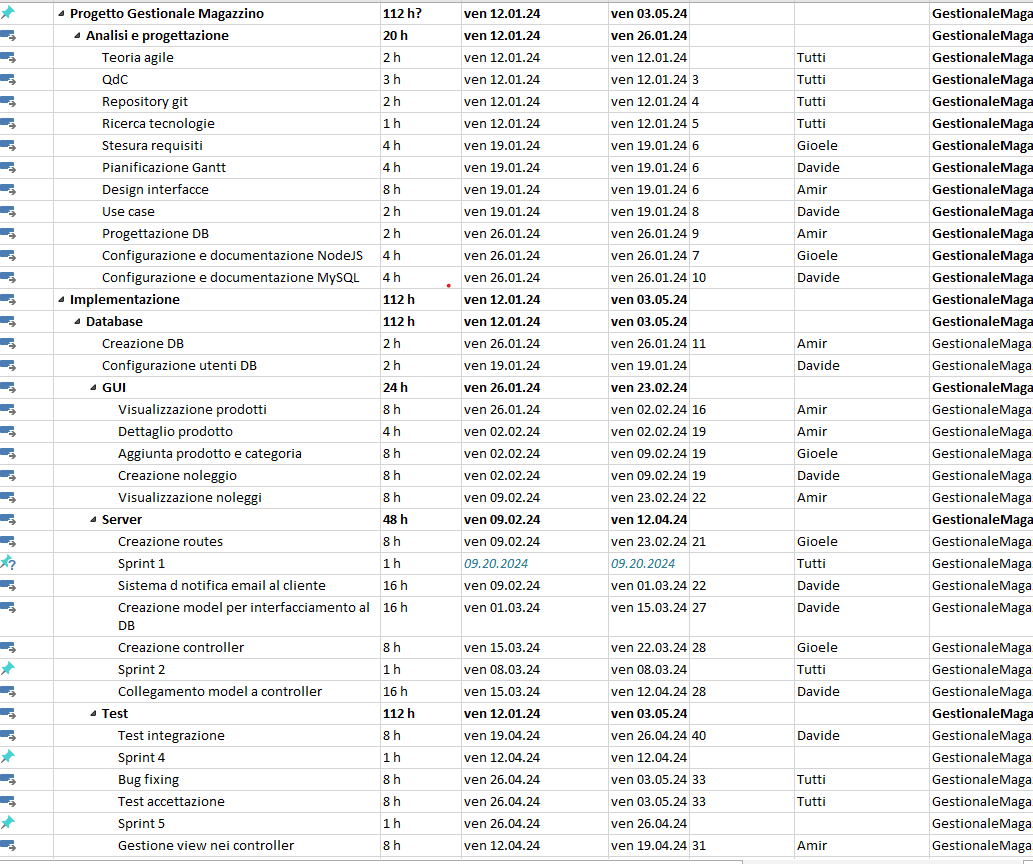


Figura 68 diagramma di GANTT consuntivo parte 1

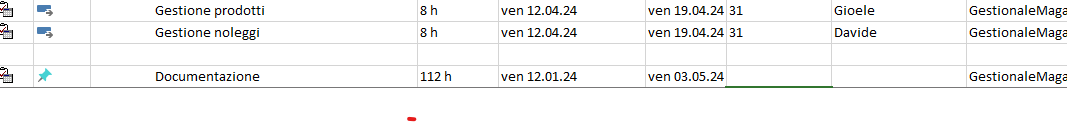


Figura 69 diagramma di GANTT consuntivo parte 2

# Conclusioni

## Sviluppi futuri

### Integrazione di robot per il picking e la movimentazione

Potremmo considerare l'integrazione di robot capaci di raccogliere e spostare merci all'interno del magazzino, al fine di migliorare l'efficienza e ridurre il lavoro manuale.

### Sviluppo di sistemi di navigazione avanzati

Potremmo investire nello sviluppo di sistemi di navigazione e controllo avanzati per consentire ai robot di muoversi in modo sicuro ed efficiente all'interno del magazzino, garantendo al contempo la sicurezza del personale.

### Integrazione con il sistema di gestione del magazzino (WMS)

Potremmo esplorare l'opportunità di integrare i robot con il nostro sistema di gestione del magazzino esistente, per garantire una comunicazione fluida e aggiornamenti in tempo reale sullo stato delle operazioni.

### Implementazione di monitoraggio remoto e manutenzione programmata

Potremmo valutare l'implementazione di sistemi di monitoraggio remoto per tenere traccia delle prestazioni dei robot e per eseguire la manutenzione preventiva in modo tempestivo, garantendo il funzionamento ottimale nel tempo.

### Utilizzo di intelligenza artificiale e apprendimento automatico

Potremmo esplorare l'utilizzo di tecniche di intelligenza artificiale e di apprendimento automatico per ottimizzare le operazioni dei robot nel tempo, consentendo loro di apprendere e migliorare continuamente le loro prestazioni.

### Analisi delle statistiche sui prodotti

Potremmo aggiungere la capacità di raccogliere e analizzare statistiche riguardanti i prodotti prelevati dal magazzino, al fine di comprendere la domanda e adattare l'inventario e le strategie di approvvigionamento di conseguenza.

## Considerazioni personali

### Amir Kawsarani

Questo progetto non solo mi ha permesso di acquisire nuove competenze tecniche, ma mi ha anche aiutato a crescere come sviluppatore, ad affrontare le mie debolezze e ad ottenere fiducia nelle mie capacità. È importante per me continuare a cercare opportunità di apprendimento e a sfidarmi con nuovi progetti per continuare a crescere e migliorare come professionista nel campo dello sviluppo software.   
CSS è sempre stato una mia grande debolezza è sono contento di essere riuscito a migliorare e comprenderlo meglio. Nonostante tutto il tempo passato a usarlo sono comunque certo che avrei potuto fare di meglio. Riguardando come ho strutturato le pagine posso chiaramente vedere che qualche mese fa ero molto meno capace. Infatti se dovessi rifare il CSS da zero uscirebbe non solo molto più pulito ma ci metterei anche meno tempo.

In un certo senso, le ore che ho passato in questo progetto per la maggior parte le ho passate a imparare a usare il CSS invece di sfruttarlo.

Oltre al CSS è stato anche interessante usare EJS e Node.js. Non li avevo mai usati prima d’ora e devo dire che sono molto comodi da usare. Soprattutto EJS, semplifica di gran lunga il processo di stampa su schermo delle informazioni.

Ho anche compreso l’importanza dell’utilizzare lo stesso materiale che usano gli altri sviluppatori. A un certo punto del progetto, ho usato la versione sbagliata di bootstrap. Quello ci ha fatto perdere qualche ora di tempo che avremmo potuto invece usare per sviluppare.

### Davide Branchi

Questo progetto mi è stato utile per capire com’è lavorare in un team, visto che non ne avevo mai affrontato uno. Mi è piaciuto utilizzare la metodologia Agile, credo sia migliore rispetto a quella Waterfall perché si ha più il controllo anche con il cliente di come sta procedendo il progetto, e si possono avere dei riscontri immediati per capire se le funzionalità che sono state implementate vanno bene al cliente.

Visto il progetto è stato realizzato in team ho anche imparato molte cose su come utilizzare al meglio git, ho capito che se viene imparato bene risulta essere uno strumento molto utile e potente per tenere traccia di tutte le versioni dei file, e inoltre integrandolo con GitHub si semplificano di molto alcune operazioni come il deploy dell’applicativo.

Per quanto riguarda la programmazione ho sicuramente approfondito le mie conoscenze di NodeJs e ho capito come si fa a integrare un database in un applicativo web.  
Inoltre ho anche appreso come utilizzare il pattern MVC con NodeJs. Il fatto di aver utilizzato i codici QR mi ha permesso di conoscere nuove librerie da utilizzare sia lato client che lato server per leggere e generare i codici.

### Gioele Cappellari

Questo progetto mi è stato utile per capire e imparare com’è lavorare in un team di sviluppo composto da più persone, cosa che non avevo mai provato prima d’ora. In più ho utilizzato la metodologia AGILE che non avevo mai utilizzato prima. Come primo impatto l’ho trovata un po’ strana come metodologia, ma andando avanti ho trovato solo pregi ad utilizzare questa metodologia. Uno di questi è quello che si può tornare indietro alla fase di progettazione e si possono aggiungere modifiche in maniera molto più facile e immediata come è capitato durante questo progetto quando qualcosa non andava bene.

Andando a guardare il lato più di sviluppo ho imparato meglio ad usare git e le sue branch (anche se ho avuto molto spesso dei problemi), capendo che è uno strumento super utile e potente. Per quanto riguarda la parte di programmazione ho imparato meglio a sviluppare un’applicazione web sfruttando il pattern MVC, questo secondo me è stata la svolta per il fatto che il codice rimane più ordinato ed è più facile separarsi le funzionalità da sviluppare. Poi ho imparato ad usare meglio NodeJS e addirittura ho imparato ad utilizzare nuovi moduli di Node (come quella della scansione dei codici QR) per affrontare le richieste per lo sviluppo dell’applicativo. In più grazie a questo progetto ho imparato dei concetti nuovi sulla sicurezza web e su come proteggere l’applicativo dalle minacce.

Per concludere posso aggiungere che è stato più bello e divertente dello scorso progetto perché farlo con altri è tutto più divertente, e poi posso dire che confronto lo scorso progetto ho imparato a gestire il mio tempo molto meglio e mi sono allenato di più a fare la documentazione, il tutto serve poi come allenamento per il progetto del quarto anno.

# Bibliografia

## Sitografia

* <https://nodejs.org/en>, 26-01-2024
* <https://expressjs.com/it/>, 26-01-2024
* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>, 26-01-2024
* <https://www.npmjs.com/package/jquery>, 02-02-2024
* <https://www.npmjs.com/package/bootstrap>, 02-02-2024
* <https://getbootstrap.com/docs/5.0/getting-started/introduction/>, 02-02-2024
* <https://www.npmjs.com/package/mysql2>, 23-02-2024
* <https://www.npmjs.com/package/express-session>, 23-02-2024
* <https://www.openssl.org/>, 15-03-2024
* <https://www.npmjs.com/package/multer>, 15-03-2024
* <https://www.nodemailer.com/>, 19-04-2024
* <https://www.npmjs.com/package/nodemailer>, 19-04-2024
* <https://www.infomaniak.com/en/free-email>, 19-04-2024
* <https://www.smtp2go.com/>, 19-04-2024
* <https://sendgrid.com/en-us>, 19-04-2024
* <https://www.mailjet.com/>, 19-04-2024

# Glossario

|  |  |
| --- | --- |
| **Termine** | **Significato** |
| JSON | Javascript Object Notation. È un formato per lo scambio dati basato sul linguaggio di programmazione JavaScript |
| GUI | L'interfaccia grafica (GUI) è un sistema che interagisce con l'utente a livello visivo. |
| SMTP | SMTP è l'acronimo di Simple Mail Transfer Protocol. È un protocollo di comunicazione utilizzato per l'invio e la ricezione di messaggi di posta elettronica su Internet. |
| SSL | SSL è l'acronimo di "Secure Sockets Layer", un protocollo che consente la trasmissione di informazioni in modo criptato e sicuro. |
| QR | Quick response code, un codice a barre bidimensionale, ossia a matrice, composto da moduli neri disposti all'interno di uno schema bianco di forma quadrata, impiegato in genere per memorizzare informazioni destinate a essere lette tramite un apposito lettore ottico o anche smartphone. |
| CRUD | CRUD è l'acronimo composto dalle iniziali delle parole inglesi Create, Read, Update e Delete., e descrive le operazioni di creazione, lettura, aggiornamento e eliminazione |
| URL | L'URL (Uniform Resource Locator) è una sequenza di caratteri che identifica univocamente l'indirizzo di una risorsa in Internet, come un documento o un'immagine. |
| Salt | Un salt è una sequenza casuale di bit che viene aggiunta alla password prima che passi attraverso l'algoritmo di hashing |
| EJS | EJS (Embedded JavaScript templating.) è un semplice linguaggio di template che ti consente di generare markup HTML con semplice JavaScript. |
| Blowfish | Blowfish è un cifrario a blocchi a chiave simmetrica sviluppato da Bruce Schneier, |
| HTTP | Sigla di hypertext transfer protocol, usata in informatica, e in particolare nella tecnica della rete Internet, che indica il protocollo di accesso alle informazioni. |
| POST | POST è un metodo utilizzato nelle richieste HTTP. |
| Case sensitive | L'espressione inglese case sensitivity, traducibile in italiano come sensibilità alle maiuscole, indica ogni operazione di analisi del testo in cui le lettere maiuscole e quelle minuscole vengono trattate come fossero caratteri completamente differenti. |
| NPM | NPM, abbreviazione di Node Package Manager, è il gestore di pacchetti ufficiale che viene installato con la piattaforma NodeJs. |
| LTS | Sigle. Long Term Support – supporto a lungo termine, usato per programmi che indicano che la versione di quel programma fornirà un supporto per un periodo più lungo di tempo. |

# Indice delle figure

[Figura 1 Use Case 10](#_Toc165643778)

[Figura 2 Diagramma di Gantt 11](#_Toc165643779)

[Figura 3 Architettura di sistema 13](#_Toc165643780)

[Figura 4 Prima versione diagramma ER 14](#_Toc165643781)

[Figura 5 Seconda versione diagramma ER 15](#_Toc165643782)

[Figura 6 Design interfacce prodotti 18](#_Toc165643783)

[Figura 7 Desgin interfacce visualizzazione noleggi 19](#_Toc165643784)

[Figura 8 Design interfacce aggiunta e ritorno noleggio 20](#_Toc165643785)

[Figura 9 Design interfacce gestione utente 21](#_Toc165643786)

[Figura 10 Diagramma di flusso noleggio articoli 22](#_Toc165643787)

[Figura 11 Diagramma di flusso chiusura noleggio 23](#_Toc165643788)

[Figura 12 Diagramma di flusso sistema di notifica noleggio in scadenza 24](#_Toc165643789)

[Figura 13 Diagramma di flusso sistema di notifica noleggio scaduto 24](#_Toc165643790)

[Figura 14 Comandi installazione MySQL 25](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643791)

[Figura 15 Comandi installazione Nodejs 25](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643792)

[Figura 16 Comandi installazione PM2 26](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643793)

[Figura 17 Script auto\_deploy.py 26](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643794)

[Figura 18 CronJob auto\_deploy.py 26](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643795)

[Figura 19 installazione OpenSSL, generazione chiave privata e certificato SSL 27](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643796)

[Figura 20 implementazione certificato applicativo 27](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643797)

[Figura 21 Struttura di cartelle dell'applicativo 28](#_Toc165643798)

[Figura 22 Connessione al database 29](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643799)

[Figura 23 Classe Noleggio (model) 29](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643800)

[Figura 24 funzione getAll() - NoleggioMapper 30](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643801)

[Figura 25 funzione getMaterialeOfNoleggio() - NoleggioMapper 30](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643802)

[Figura 26 funzione deleteDatastoreElement() - datasotreManager (model util) 31](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643803)

[Figura 27 funzione datastoreElementExists() - datastoreManager (model util) 31](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643804)

[Figura 28 funzione createFulDatastorePath() - datastoreManager (model util) 31](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643805)

[Figura 29 funzione toBase64String() - QRGenerator (model util) 32](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643806)

[Figura 30 funzione sanitizeInput() - sanitizer (model util) 32](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643807)

[Figura 31 funzione validateEmail() - sanitizer (model util) 32](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643808)

[Figura 32 funzione login() - loginController 33](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643809)

[Figura 33 funzione loadViewEditProduct() - prodottiController 34](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643810)

[Figura 34 funzione showProductDetails() - prodottiController 35](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643811)

[Figura 35 funzione addProduct() sanificazione dati - prodottiController 36](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643812)

[Figura 36 funzione addProduct() controlli - prodottiController 36](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643813)

[Figura 37 funzione deleteProduct() - prodottiController 37](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643814)

[Figura 38 funzione addCategoria() - categorieController 38](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643815)

[Figura 39 funzione deleteCategoria() - categorieController 39](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643816)

[Figura 40 Controllo prodotti creazione nuovo noleggio 40](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643817)

[Figura 41 Controllo dei permessi per visualizzare i dettagli di un noleggio 41](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643818)

[Figura 42 Controllo per la chiusura forzata di un noleggio 41](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643819)

[Figura 43 funzione loadViewEditUtente() - utentiController 42](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643820)

[Figura 44 funzione showUserDetail() - utentiController 43](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643821)

[Figura 45 controlli vari funzione addNew() - utentiController 44](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643822)

[Figura 46 controllo funzione editUtente() - utentiController 44](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643823)

[Figura 47 controllo immagine funzione editProfilo() - utentiController 45](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643824)

[Figura 48 controlli funzione deleteUser() - utentiController 46](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643825)

[Figura 49 funzione showManualeUtente() - manualeController 47](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643826)

[Figura 50 Template EJS header 48](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643827)

[Figura 51 Implementazione header nelle view 48](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643828)

[Figura 52 Configurazione log middleware 50](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643829)

[Figura 53 Funzione isAuthenticated() - Auth middleware 51](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643830)

[Figura 54 Funzione isGestore() - Auth middleware 51](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643831)

[Figura 55 Funzione isAmministratore() - Auth middleware 52](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643832)

[Figura 56 Configurazione multer 52](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643833)

[Figura 57 Controllo dell'immagine durante l'upload 53](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643834)

[Figura 58 Configurazione nodemailer 54](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643835)

[Figura 59 Funzione sendMail() 54](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643836)

[Figura 60 Funzione initializeMailerWorker() - Mailer worker 55](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643837)

[Figura 61 Funzione log() - Mailer worker 55](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643838)

[Figura 62 Funzione sendNotificationEmails() - Mailer worker 56](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643839)

[Figura 63 Invio del report giornaliero noleggi scaduti 57](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643840)

[Figura 64 Configurazione della transazione per eseguire i test 58](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643841)

[Figura 65 Test get per il materiale mapper 58](file:///D:\M426\Progetto\GestionaleMagazzino\3_Documentazione\word\Documentazione_GestionaleMagazzino.docx#_Toc165643842)

[Figura 66 Risultati prima parte test jest 59](#_Toc165643843)

[Figura 67 Risultati seconda parte test jest 60](#_Toc165643844)

[Figura 68 diagramma di GANTT consuntivo parte 1 71](#_Toc165643845)

[Figura 69 diagramma di GANTT consuntivo parte 2 71](#_Toc165643846)

# Allegati

Assieme alla documentazione sono allegati:

* QdC
* Abstract
* Diari di lavoro
* Applicativo
* Database
* Catella degli Allegati
* Manuali