

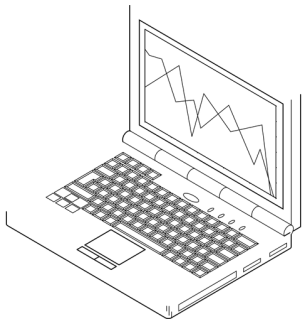
Progetto di basi di Dati

Gestione Ordini di un'azienda di logistica



Anno Accademico 2022 / 2023
Giovanni Spera

Specifiche del Progetto



Vogliamo creare una piattaforma web che permetta di sapere l'evoluzione dei vari ordini che sono stati gestiti dal nostro committente.

La piattaforma sarà accessibile dalle diverse aziende, in modo da poter sempre essere aggiornate sui loro ordini.



Ordine e Trasporto

Per l'azienda è importante dividere i vari viaggi che effettua un ordine

Un viaggio è il movimento di uno o più ordini da una locazione ad un'altra

Un trasporto è l'insieme di viaggi che portano un singolo ordine dall'origine alla destinazione



Deposito e Archivio

È importante, inoltre, distinguere tra deposito e archivio.

Il deposito è dove vengono conservate le merci che verranno gestite a breve.

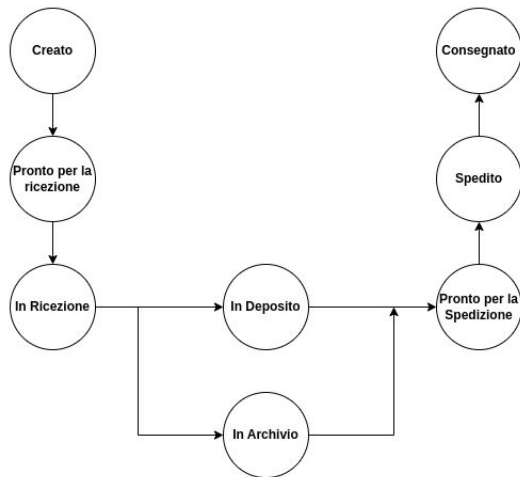


L'archivio è dove vengono conservate le merci a lungo termine.

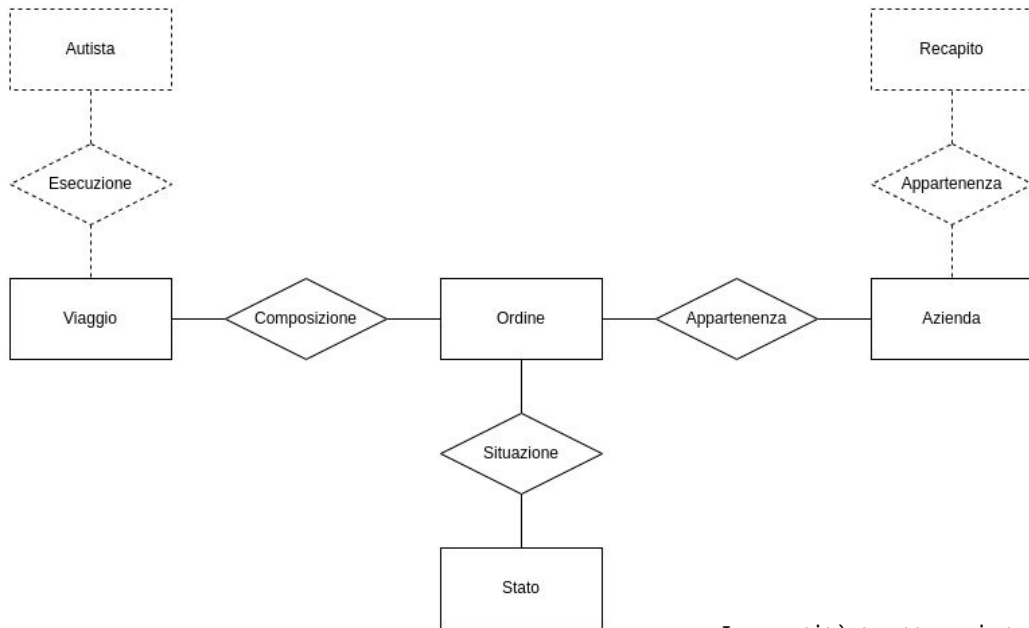


Uno stato

Si vuole conservare tutte le informazioni possibili riguardo un ordine, per una consultazione successiva. Uno stato rappresenta cosa succedeva ad un ordine in uno specifico momento.



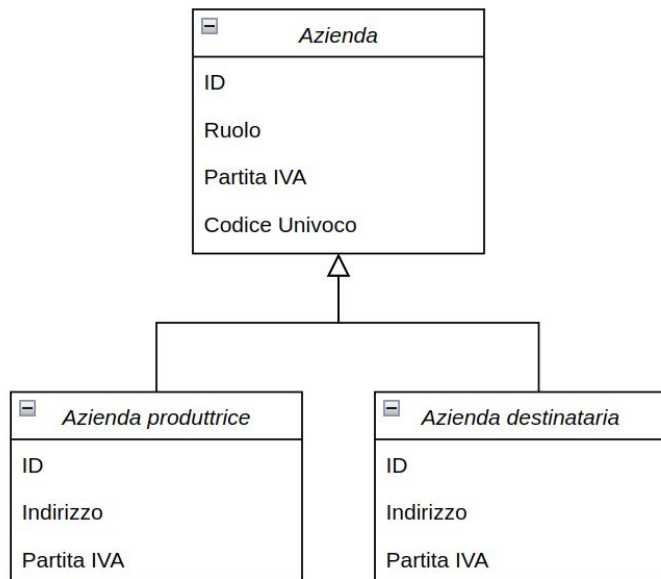
Primo raffinamento e schema a scheletro



L'ordine è l'elemento principe, composto da diversi viaggi, appartenente a un'azienda e con la raccolta di stati che l'hanno coinvolto.

Le entità tratteggiate sono opzionali e soggette a revisioni

Miglioramento e secondo affinamento



In prima battuta c'era una distinzione tra azienda produttrice e destinataria, dovuta alle operazioni possibili.

Si è risolto tramite un'unica tabella e l'aggiunta del membro ruolo, che distingue le azioni possibili.

Il codice univoco è stato un elemento richiesto successivamente, viene usato per la fatturazione

Dimensionamento

Entità	Stima delle occorrenze
Ordini	Meno di una centinaia al mese
Viaggi	Tra il doppio e il triplo del numero di ordini
Aziende	Poche decine
Stati	Circa sette volte il numero di ordini

Alcune stime per la quantità di dati all'interno della basi di dati,

viene tenuto conto che gli ordini vengono divisi per anni, e quindi all'inizio dell'anno è possibile spostare i dati in un archivio.

Vincoli

Alcuni vincoli che sono stati redatti:

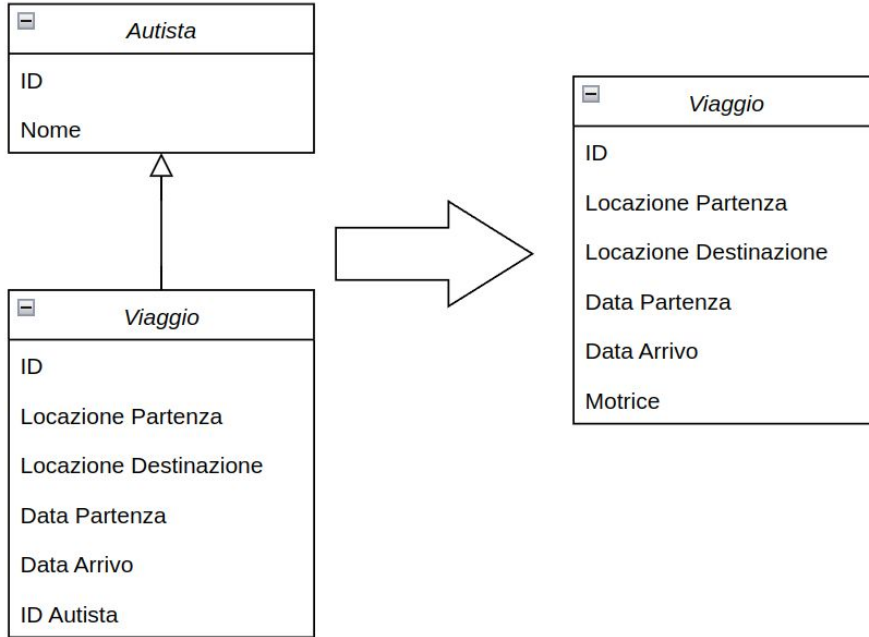
- Un' **azienda** non può avere un *nome* vuoto, ma potremmo non conoscere la *partita iva* o il *codice univoco*
- Un **ordine** deve essere composto da almeno un *collo*
- Un **ordine** non può avere uguali *produttore* e *destinatario*
- Un **viaggio** avrà una *data di arrivo* successiva a quella di *ritorno* (non ci dovrebbero essere problemi nel caso di diversi fusi, che in ogni caso sono trascurabili)
- Un **viaggio** non può avere uguali *partenza* e *destinazione*
- Lo **stato** deve avere un valore valido

Numero delle operazioni

Una raccolta delle varie operazioni effettuate, tutte di tipo interattivo

Azione	Frequenza
Visione degli ordini in corso	10 Volte/Giorno (Diviso tra varie aziende)
Aggiornamento stato di un ordine	5 Volte/Giorno
Consultazione stati di un ordine	15 Volte/Giorno (Diviso tra ordini in corso e ordini consegnati e tra varie aziende)
Inserimento o modifica di un' azienda, con aggiunta di utente	1 Volta/Mese

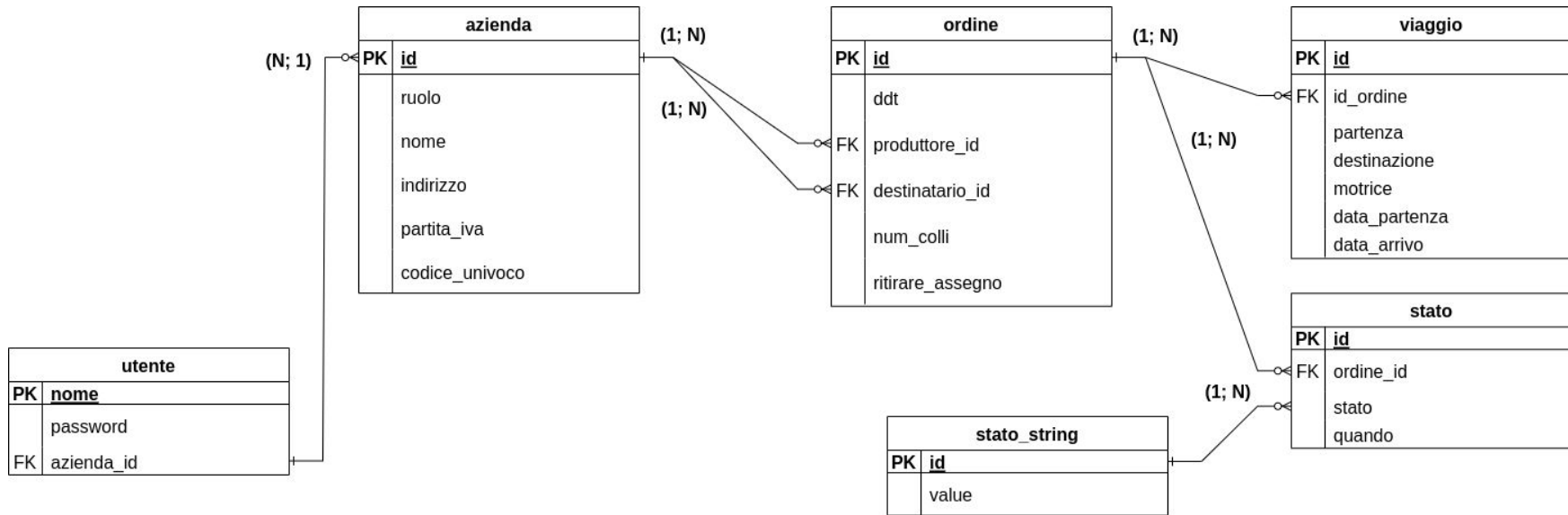
Ristrutturazione



L'entità autista è risultata poco utile, in quanto è preferibile conoscere la matrice, e quindi il camion che ha effettuato il viaggio rispetto a chi l'ha effettuato.

Schema Finale

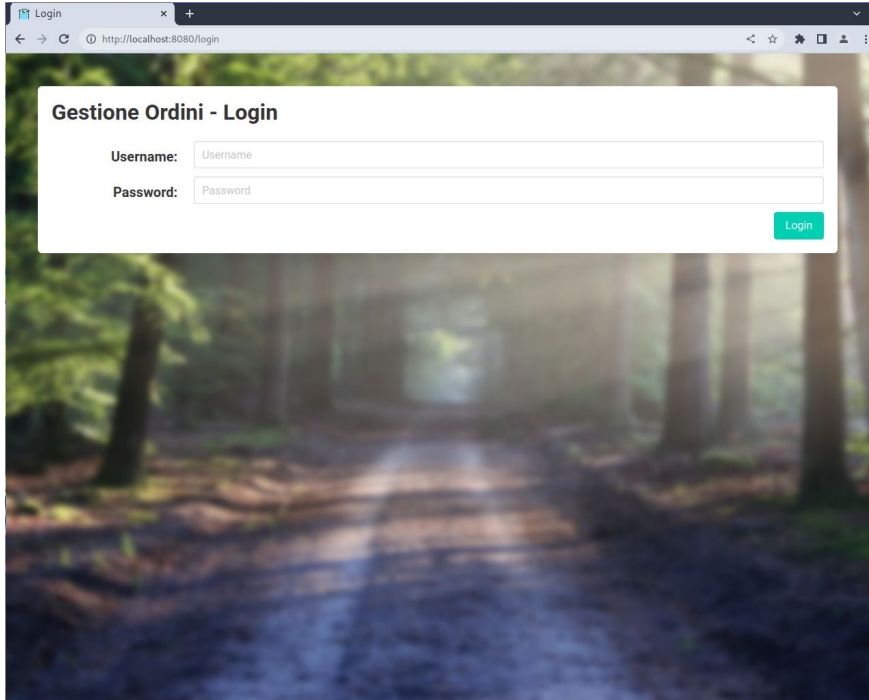
La struttura finale della basi di dati



Gli utenti sono necessari per accedere alla piattaforma web

L'indicazione di uno stato è un intero per evitare ridondanze

Interfaccia Applicazione



L'applicazione sarà ospitata su un sito web accessibile da internet,



Un utente non autenticato vede solamente la pagina di login

Non è possibile registrarsi in quanto gli account vengono creati dall'amministratore aziendale

La gestione ordini

Una volta inserite le credenziali
l'utente accede alla gestione degli
ordini

Qui ha una panoramica completa di
tutti gli ultimi ordini

Gestione Ordini							Spera Logistica ▾
Ordini							
	Aggiungi ordine						
	Aggiungi azienda						
DDT	Produttore	Destinatario	n° Colli	Assegno	Ultimo aggiornamento	Data aggiornamento	
Scarpe/1	Facciamo scarpe	Vendiamo scarpe	1		✓ Consegnato	⌚	5/1/2022
Scarpe/2	Vendiamo scarpe	Facciamo scarpe	2	✓	📦 Pronto per essere spedito	⌚	11/1/2022

Ogni ordine mostra le sue generalità e il suo ultimo
aggiornamento

E' inoltre possibile, per chi ne è autorizzato
registrare un nuovo ordine o una nuova azienda

Aggiunta di un Ordine

Gestione Ordini Scera Logistica

Ordini

Aggiungi ordine

Aggiungi azienda

DDT	Produttore	Destinatario	n° Colli	Assegno	Ultimo aggiornamento	Data aggiornamento
Scarpe/1	Facciamo scarpe					5/1/2022
Scarpe/2	Vendiamo scarpe					11/1/2022

Aggiungi Ordine

Mittente Facciamo scarpe

Destinatario Vendiamo scarpe

DDT Scarpe/3

Numero Colli 10

Ritirare Assegno ☒

Aggiungi Chiudi

Cliccando sull'apposito pulsante il popup permette di aggiungere le generalità di un nuovo ordine

Analogamente possono essere registrate e modificate nuove aziende

Informazioni su un Ordine

Di tutti gli ordini è possibile avere un riepilogo

Qui sono mostrati tutti gli aggiornamenti riguardanti l'ordine, anche a distanza di mesi

The screenshot displays a web interface for 'Gestione Ordini' (Order Management). A modal window titled 'Info Ordine: Scarpe/1' is open, showing the following details:

- Mittente:** Facciamo scarpe
- Destinatario:** Vendiamo scarpe
- Numero Colli:** 2

Aggiornamenti:

- Il 7/1/2022 - 7/1/2022
Viaggio da **DP Spera Logistica** a **Negozio**
Con motrice: **GV123NI**
- Il: 5/1/2022 l'ordine è ✓ **Consegnato**
- Il: 4/1/2022 l'ordine è 📦 **Spedito**
- Il: 3/1/2022 l'ordine è 📦 **In deposito**
- Il: 3/1/2022 l'ordine è 📦 **Pronto per essere spedito**
- Il 3/1/2022 - 3/1/2022
Viaggio da **DP Facciamo Scarpe** a **DP Spera Logistica**
Con motrice: **SP312RA**
- Il: 2/1/2022 l'ordine è 📦 **Pronto per essere ricevuto**

At the bottom of the modal is a 'Chiudi' (Close) button. In the background, a table titled 'Ordini' is visible with columns 'DDT' and 'Produttore', showing two rows: 'Scarpe/1' from 'Facciamo scarpe' and 'Scarpe/2' from 'Vendiamo scarpe'. On the right side of the background interface, there is a 'Data aggiornamento' section with a list of dates: 5/1/2022 and 11/1/2022, each preceded by a circular icon.

Interfaccia Web

L'interfaccia web funziona tramite comunicazioni tra il client e il server.

Il primo interpreta gli input dell'utente e crea la richiesta,

Il server, dopo essersi assicurato che la richiesta sia valida, trasmette le modifiche al DBMS

Il client usa JavaScript, abbinato a ReactJS per interagire con l'utente

Il server usa Go, linguaggio sviluppato da Google per gestire le richieste

Oracle è usato per il DBMS

JavaScript



ORACLE
DATABASE



Creazione Tabelle

Passiamo alla creazione delle tabelle e delle sequence accessorie:

```
CREATE TABLE utente (  
    nome varchar(20) primary key,  
    password char(64) not null,  
    azienda_id integer not null  
);
```


```
CREATE TABLE azienda (  
    id integer primary key,  
    ruolo integer,  
    nome varchar(20) not null,  
    indirizzo varchar(50),  
    partita_iva char(11),  
    codice_univoco char(6)  
);
```

```
CREATE TABLE stato_string (  
    id integer primary key,  
    value varchar(30)  
);
```

```
CREATE TABLE ordine (  
    id integer primary key, -- Autoincrement  
    ddt varchar(20) not null,  
    produttore_id integer not null,  
    destinatario_id integer not null,  
    num_colli integer default 1 not null,  
    ritirare_assegno integer not null -- Boolean  
);
```

```
CREATE TABLE viaggio (  
    id integer not null,  
    id_ordine integer,  
    partenza varchar(20),  
    destinazione varchar(20),  
    motrice char(7),  
    data_partenza date,  
    data_arrivo date  
);
```

Le sequenze sono usate per dare un
identificativo univoco, quando necessario



```
CREATE SEQUENCE ordine_seq INCREMENT BY 1 NOCACHE NOCYCLE ORDER;
```

Creazione Foreign Key

```
ALTER TABLE utente ADD CONSTRAINT fk_utente_azienza  
FOREIGN KEY(azienda_id) REFERENCES azienda(id);
```

```
ALTER TABLE ordine ADD CONSTRAINT fk_ordine_produuttore  
FOREIGN KEY(produttore_id) REFERENCES azienda(id);
```

```
ALTER TABLE ordine ADD CONSTRAINT fk_ordine_destinatario  
FOREIGN KEY(destinatario_id) REFERENCES azienda(id);
```

```
ALTER TABLE viaggio ADD CONSTRAINT fk_viaggio_ordine  
FOREIGN KEY(id_ordine) REFERENCES ordine(id);
```

```
ALTER TABLE stato ADD CONSTRAINT fk_stato_ordine  
FOREIGN KEY(ordine_id) REFERENCES ordine(id);
```

Creazione Viste

La tabella con gli ultimi aggiornamenti ha frequenti letture, è utile allora creare una vista materializzata, in questo modo non è necessario ricalcolare tutti i valori ad ogni lettura

```
CREATE MATERIALIZED VIEW ultimi_statì AS
SELECT ordine.ddt, produttore.nome as produttore_nome, destinatario.nome as destinatario_nome, ordine.num_colli,
ordine.ritirare_assegno, MAX(stato.stato) as stato, stato_string.value as stato_string, MAX(stato.quando) as quando
FROM ordine
    JOIN stato ON ordine.id = stato.ordine_id
    JOIN azienda produttore ON ordine.prodotto_id = produttore.id
    JOIN azienda destinatario ON ordine.destinatario_id = destinatario.id
    JOIN stato_string ON stato_string.id = ( SELECT MAX(stato.stato) FROM stato WHERE ordine_id = ordine.id)
GROUP BY ordine.ddt, produttore.nome, destinatario.nome, ordine.num_colli, ordine.ritirare_assegno, stato_string.value
ORDER BY quando DESC;
```

Creazione Trigger

I trigger sono usati per automatizzare alcune importanti operazioni:

Il trigger *order_new* si attiva quando viene inserito un nuovo ordine e ne registra il suo stato di “Creato”

```
CREATE TRIGGER order_new AFTER INSERT ON ordine FOR EACH ROW
INSERT INTO stato VALUES (stato_seq.nextval, :new.id, 0, CURRENT_TIMESTAMP)
```

Il trigger *ultimi_stati_update* si occupa di tenere sempre aggiornata la vista *ultimi_stati*, in questo modo la vista viene calcolata solo quando veramente necessario

```
CREATE TRIGGER ultimi_stati_update AFTER INSERT OR UPDATE OR DELETE ON stato
```

```
BEGIN
```

```
CREATE OR REPLACE MATERIALIZED VIEW ultimi_stati
SELECT ordine.ddt, produttore.nome as produttore_nome, destinatario.nome as destinatario_nome, ordine.num_colli, ordine.ritirare_assegno, MAX(stato.stato as stato_string, MAX(stato.quando as quando
FROM ordine
JOIN stato ON ordine.id = stato.ordine_id
JOIN azienda produttore ON ordine.prodotto_id = produttore.id
JOIN azienda destinatario ON ordine.destinatario_id = destinatario.id
JOIN stato_string ON stato_string.id = (SELECT MAX(stato.stato) FROM stato WHERE ordine_id = ordine.id)
GROUP BY ordine.ddt, produttore.nome, destinatario.nome, ordine.num_colli, ordine.ritirare_assegno, stato_string.value
ORDER BY quando DESC;
```

```
END;
```

Dichiarazione Procedure

Sono state aggiunte alcune procedure per facilitare la gestione del DBMS, in particolare la gestione degli utenti e delle loro password, dato il basso numero non è stato creato un package apposito:

```
CREATE PROCEDURE aggiungi_utente(utente IN varchar(20), password IN varchar(50), azienda_nome IN varchar(20))
```

```
CREATE PROCEDURE cambia_password_utente(utente IN varchar(20), nuova_password IN varchar(50))
```

```
CREATE PROCEDURE ricalcola_ultimi_ordini
```

Lato Server

Il server comunica con il client attraverso il web, sono quindi stati impostati diversi percorsi per le diverse richieste:

```
http.Handle("/", server.LoggedInMiddleware(server.HandleHome, "/login"))
http.HandleFunc("/login", server.HandleLogin)
http.HandleFunc("/logout", server.HandleLogout)
http.Handle("/api/info-order", server.LoggedInMiddleware(server.HandleApiInfoOrder, "/login"))
http.Handle("/api/orders", server.LoggedInMiddleware(server.HandlerApiGetOrders, "/login"))
http.Handle("/api/avaible-receivers", server.LoggedInMiddleware(server.HandlerApiReceivers, "/login"))
http.Handle("/api/new-order", server.LoggedInMiddleware(server.HandleApiNewOrder, "/login"))
http.Handle("/api/me", server.LoggedInMiddleware(server.HandlerApiAboutMe, "/login"))

http.Handle("/static/", http.StripPrefix("/static/", http.FileServer(http.Dir("static"))))
```

Le richieste che iniziano con **/api/** sono interne e usate per la comunicazione tra il client e il server, **/static/** invece contiene i file statici, come gli stili e il codice javascript

La maggior parte delle richieste devono essere autenticate, altrimenti riportano alla pagina di login

Lato Server: Connessione con il DBMS

La connessione avviene tramite la libreria **sqlx** e il driver specifico **ora**, in questo modo le query possono facilmente essere adattate ad altri DBMS.

Creiamo poi la struttura **Server**, il vero nucleo che gestisce le richieste

```
log.Println("Connecting to database")
url := ora.BuildUrl("localhost", 1521, "XEPDB1", "gs", "gs", map[string]string{})
db, err := sqlx.Connect("oracle", url)
if err != nil {
    log.Fatalln("Cannot connect to database:", err)
}

log.Println("Initializing server")
server, err := NewServer(db, os.DirFS("./tpl"), log.NewEntry(log.StandardLogger()))
if err != nil {
    log.Fatalln("Cannot initialize server:", err)
}
```


Lato Server: Gestione delle password

Le password non vengono salvate direttamente nel database in chiaro, invece passano per un algoritmo di *hash*, per semplicità si è usato *sha256*

Una volta autenticato l'utente creiamo un token *JWT* contenente le sue informazioni essenziali e una specifica scadenza

```
err := row.Scan(&correctHash, &aziendaId, &azienda, &aziendaRole)
if err == sql.ErrNoRows {
    fmt.Fprintln(w, "Utente non trovato")
    log.Warningln("User not found")
    return
} else if err != nil {
    http.Error(w, "Internal server error", http.StatusInternalServerError)
    log.Errorln("Cannot query password:", err)
    return
}

hashBytes := sha256.Sum256([]byte(password))
hash := hex.EncodeToString(hashBytes[:])

if hash != correctHash {
    fmt.Fprintln(w, "Password sbagliata")
    log.Warningf("Wrong password: given: %s(%q), expected: %s\n", hash, password, correctHash)
    return
}

tok := jwt.NewWithClaims(jwt.SigningMethodHS256, UserCookie{
    Username:    username,
    CompanyID:   aziendaId,
    CompanyName: azienda,
    CompanyRole: aziendaRole,
    Expiration:  time.Now().AddDate(0, 0, 7),
}).Claims()
```

Lato Server: Connessione con il DBMS

La connessione avviene
tramite **sqlx** che concede una
comoda interfaccia per
eseguire query e raccoglierne i
risultati

```
type Order struct {
    ID            int           `db:"ID"`
    DDT           string        `db:"DDT"`
    ProducerName  string        `db:"PRODUTTORE_NOME"`
    RecipientName string        `db:"DESTINATARIO_NOME"`
    NumPackages   int           `db:"NUM_COLLI"`
    WithdrawBankCheck bool        `db:"RITIRARE_ASSEGNO"`
    StateID       int           `db:"STATO"`
    StateString   string        `db:"STATO_STRING"`
    When          time.Time   `db:"QUANDO"`
}

result := make([]Order, 0, 10)

orders, err := s.Database.Queryx(s.Database.Rebind(`
    SELECT * FROM ultimi_stat1
    WHERE (:1<0) or (produttore.id = :1 or destinatario.id = :1)
`), claims["aziendaId"].(float64))
if err != nil {
    log.Errorln("Cannot retrieve orders:", err)
    http.Error(w, "Internal server error", http.StatusInternalServerError)
    return
}
defer orders.Close()

for orders.Next() {
    var order Order
    err := orders.StructScan(&order)
    if err != nil {
        log.Errorln("Cannot scan row:", err)
        http.Error(w, "Internal server error", http.StatusInternalServerError)
        return
    }
    result = append(result, order)
}
```

Grazie per l'attenzione

Il codice sorgente del progetto è disponibile pubblicamente su GitHub:



<https://github.com/gSpera/progetto-basi>