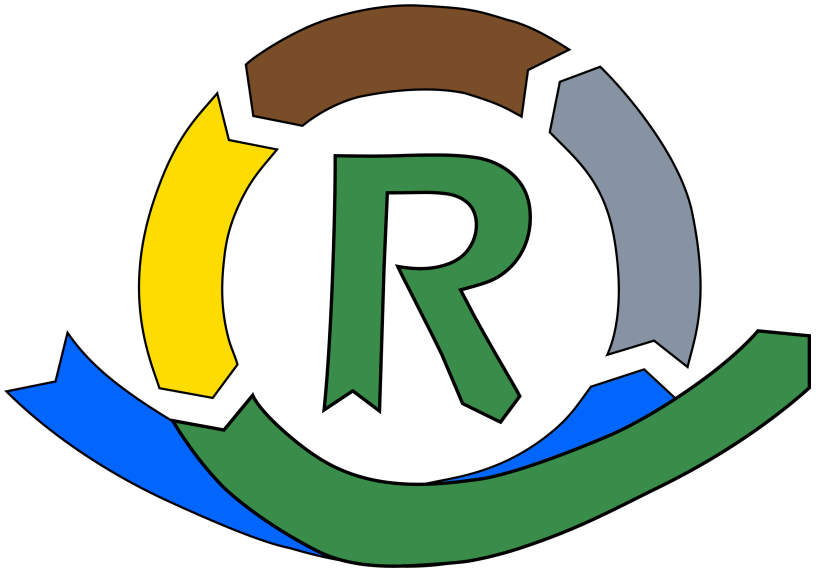
Rius.Co.

L'e-commerce per il RIUSo COllaborativo



Mauro Pellonara

Elaborato di Informatica e Sistemi e Reti

I.I.S. Volterra Elia

31/05/2021

Indice

1 Introduzione

L'accelerazione del progresso tecnologico, la progressiva liberalizzazione degli scambi commerciali e il grande sviluppo delle comunicazioni hanno portato alla nascita del mercato globale. Le aziende realizzano nuovi prodotti a ritmi sempre più elevati con prezzi sempre più competitivi ed economici; questo spinge i consumatori a credere che i prodotti in loro possesso diventino obsoleti nell'arco di pochi mesi e li invita quindi ad acquistare sempre di più. Questo continuo ciclo d'acquisto è alla base del consumismo moderno che negli ultimi anni sta producendo sempre più spazzatura mettendo in pericolo il pianeta intero. Numerosi studi [1] rivelano che il problema spesso risiede nel fatto che i consumatori e le aziende, non sapendo che fare dei prodotti obsoleti, li buttano via anche se funzionanti.

Rius.Co. cerca di risolvere questa problematica attraverso una piattaforma per il riuso online che permette a chiunque di poter acquisire prodotti di seconda mano funzionanti in modo del tutto gratuito e facilmente accessibile. Rius.Co. si ispira ai già esistenti centri del riuso presenti nella Regione Marche, ma innova portando questa iniziativa solidale nelle mani di tutti attraverso il sito web sulla quale si appoggerà la piattaforma. Il sito web consiste in un e-commerce customer-to-customer in cui tutti possono offrire e richiedere prodotti usando la moneta di scambio virtuale coniata a questo scopo: il Green Coin. Spendendo un Green Coin gli utenti potranno richiedere ed ottenere un prodotto da un altro utente appartenente alla piattaforma. Ogni prodotto, qualsiasi esso sia, se funzionante e in buone condizioni potrà essere inserito nel Marketplace dove gli verrà assegnato il "prezzo" di un Green Coin a prescindere dal suo valore economico che è trascurato in quanto si tratta di prodotti che altrimenti sarebbero stati buttati in discarica.

(a) Logo Rius.Co. (b) Logo esteso Rius.Co. (c) Green Coin

Figura 1: Loghi ed icone dell'azienda

Un importante punto di forza della piattaforma è la possibilità d'integrazione con i già esistenti centri del riuso locali facilitandone il flusso dei prodotti e aiutandoli quindi a costruire un inventario completo o ad alleviare la mole di lavoro e la quantità di prodotti che questi centri, spesso di modeste dimensioni, gestiscono con difficoltà. I centri del riuso locali possono quindi beneficiare in modo considerevole da una collaborazione con la piattaforma Rius.Co. che ne guadagnerebbe l'opportunità di ingrandire il proprio bacino d'utenza. La piattaforma può anche servire come rete di collegamento dei centri del riuso già esistenti, aiutandoli nella comunicazione e negli scambi di prodotti in modo efficiente. Il progetto risulta quindi semplice dal punto di vista operativo, economico dal punto di vista finanziario e sostenibile dal punto di vista ambientale.

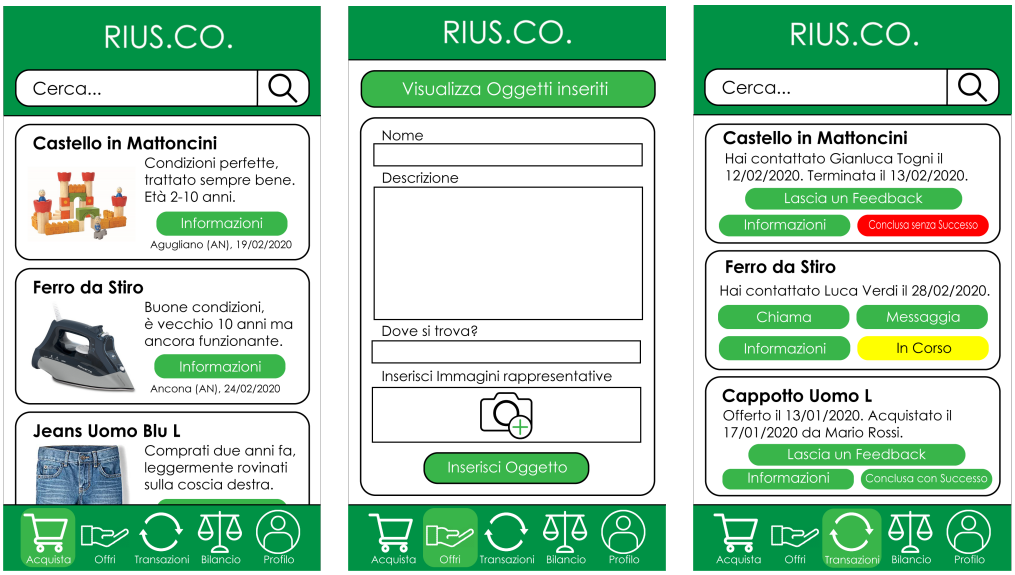
Rius.Co. ha in comune numerosi obiettivi con l'importante Agenda 2030, un programma d'azione per le persone, il pianeta e la prosperità sottoscritto nel 2015 dai Paesi membri dell'ONU [2], questi obiettivi sono:

- 1° Obiettivo – "Porre fine alla povertà in tutte le sue forme": chiunque può commerciare gratuitamente sulla piattaforma a prescindere dal proprio reddito o situazione economica;
- 11° Obiettivo – "Rendere le città e gli insediamenti umani inclusivi, sicuri, duraturi e sostenibili": Rius.Co. rende le città più sostenibili e sicure grazie alla riduzione dei rifiuti e l'aumento dei prodotti riutilizzati;
- 12° Obiettivo – "Garantire modelli sostenibili di produzione e di consumo": la piattaforma offre un modello sostenibile di consumo congruo alla vita d'oggi;
- 13° Obiettivo – "Promuovere azioni, a tutti i livelli, per combattere il cambiamento climatico": promuovendo il riutilizzo dei prodotti e, di conseguenza, scoraggiando l'accumulo di rifiuti nelle discariche, Rius.co. supporta la lotta contro il cambiamento climatico.

Figura 2: Obiettivi 1, 11, 12 e 13 dell'Agenda 2030.

2 Analisi dei requisiti

La registrazione alla piattaforma Rius.Co. sarà obbligatoria per poter accedere al Marketplace, gli utenti dovranno inserire la propria e-mail identificativa, il nome dell'account, la password, un'immagine di profilo e la città in cui vivono. La piattaforma inizialmente sarà accessibile esclusivamente tramite il sito web, verrà successivamente introdotta un'applicazione mobile della quale è già presente il prototipo visualizzabile qui [3].



(a) Acquista (b) Offri (c) Transazioni

Figura 3: Prototipo dell'applicazione mobile.

Gli utenti potranno ottenere i Green Coin necessari per commerciare sul Marketplace attraverso le seguenti modalità:

- **Registrazione sulla piattaforma**: verrà fornito un Green Coin alla registrazione di un nuovo account;
- **Condivisione della piattaforma sui propri canali Social**: gli utenti che sceglieranno di condividere e pubblicizzare online l'applicazione otterranno un Green Coin. Questa modalità sarà rinnovabile mensilmente, perciò gli utenti potranno ottenere un Green Coin ogni mese semplicemente pubblicizzando la piattaforma che ne guadagnerà in popolarità ed utenti, fidelizzando anche i propri clienti;

- **Offrire prodotti nel Marketplace**: l'utente potrà inserire prodotti a patto che siano funzionanti e in buone condizioni, otterrà un Green Coin per ogni prodotto inserito nel Marketplace e ceduto ad un altro utente;

- **Dimostrare di essere in una situazione di difficoltà economica** attraverso la dichiarazione dei redditi o l'attestazione ISEE, a tali utenti verranno forniti mensilmente dai 2 ai 5 Green Coin.

Il Green Coin è stato pensato per risolvere una problematica importante comune a tutti gli attuali centri del riuso fisici: la presenza di utenti che abusano della piattaforma richiedendo una grande mole di prodotti, senza mai contribuire al sistema offrendone altrettanti.

Gli utenti dopo essersi registrati potranno accedere alla piattaforma ed usufruire dei servizi che essa offre, sarà quindi possibile:

- **Acquistare un prodotto di cui si necessita attraverso la pagina "Acquista".** Nella pagina saranno presenti tutti i prodotti offerti da utenti membri della piattaforma nella vicinanza dell'utente. Verrà data anche la possibilità di cercare un determinato prodotto all'interno del Marketplace. L'utente interessato ad un prodotto potrà spendere un Green Coin per contattare l'utente offerente e accordarsi per lo scambio. Nel caso in cui lo scambio non si sia concluso con successo l'utente richiedente otterrà indietro il Green Coin che aveva speso;

- **Offrire un prodotto attraverso la pagina "Offri".** Nella pagina sarà possibile inserire un prodotto da offrire nel Marketplace, saranno richieste una breve descrizione e delle foto rappresentative del prodotto, ne verrà resa pubblica anche la posizione. Nel caso un utente sia interessato al prodotto il venditore verrà contattato da quest'ultimo e otterrà un Green Coin nel caso lo scambio si concluda con successo;

- **Controllare le proprie transazioni attraverso la pagina "Transazioni".** L'utente potrà visualizzare tutte le sue transazioni, i dettagli relativi ad esse e lo stato attuale: "In Corso", "Conclusa senza Successo" e "Conclusa con Successo". Saranno anche visualizzabili i movimenti di Green Coin in entrata ed uscita;

- **Visualizzare e modificare il proprio profilo attraverso la pagina "Profilo".** L'utente potrà visualizzare il proprio profilo e modificare i dati relativi ad esso mantenendo aggiornate informazioni importanti come la città di residenza.

3 Analisi funzionale

aspnet core 5.0
- c#
- mvc
- riepilogo generico delle cose che si vedono dopo

4 Database

Nello sviluppo di un database la prima scelta da compiere è quella relativa al DBMS che si occupa di gestire e garantire l'accesso al database; per questo progetto ho scelto SQLite3 [4] in quanto in questo caso il DBMS non si basa sull'architettura client-server ma viene direttamente incluso nel programma che s'interfaccia con il database, ciò rende la configurazione quasi automatica. Nel caso specifico di Rius.Co. il database è generato tramite la tecnologia Entity Framework Core sviluppata da Microsoft che pone un ulteriore livello di virtualizzazione tra l'utente e il DBMS che quindi non comunicheranno direttamente tra loro. SQLite3 è conosciuto soprattutto per la sua portabilità in quanto il database è contenuto in un singolo file che risulta molto leggero e compatto, inoltre è anche compatibile con la maggior parte dei sistemi operativi presenti sul mercato.

Per garantire la semplicità di gestione e la velocità di esecuzione delle query nel database sono presenti solo 3 tabelle: Users, Products e Transactions. Queste sono il minimo indispensabile per la creazione di un e-commerce funzionante. Quella riportata nel modello E/R è la prima versione del database che permette di caricare un'unica immagine per prodotto, nelle versioni successive verranno introdotti aggiornamenti graduali paralleli allo sviluppo completo del sito web. Il database è visualizzabile insieme a tutto il progetto scaricando la repository di GitHub disponibile qui [5].

La tabella Users contiene le informazioni base necessarie per l'utilizzo della piattaforma: codice identificativo, nome utente, password, email identificativa, città di residenza, immagine di profilo, salt per la generazione dell'hash della password, api key per eseguire richieste tramite l'API e quantità di green coin posseduti.

La tabella Products contiene invece i seguenti campi: codice identificativo, nome, breve descrizione, immagine rappresentativa, disponibilità (per tenere traccia dei prodotti anche dopo la loro vendita o rimozione dal Marketplace), data dell'ultimo aggiornamento apportato e codice identificativo dell'utente proprietario del prodotto.

La tabella Transactions contiene le transazioni effettuate tra gli utenti descritte attraverso i seguenti campi: codice identificativo della transazione, codice identificativo del prodotto scambiato, codice identificativo dell'utente proprietario del prodotto al momento dello scambio (campo necessario per tenere traccia dei cambi di proprietà e delle transazioni anche dopo che queste si sono concluse), codice identificativo dell'utente acquirente, stato della transazione ("In Corso", "Completata Con Successo" o "Completata Senza Successo") e data dell'ultimo aggiornamento relativo alla transazione.

4.1 Modello E/R

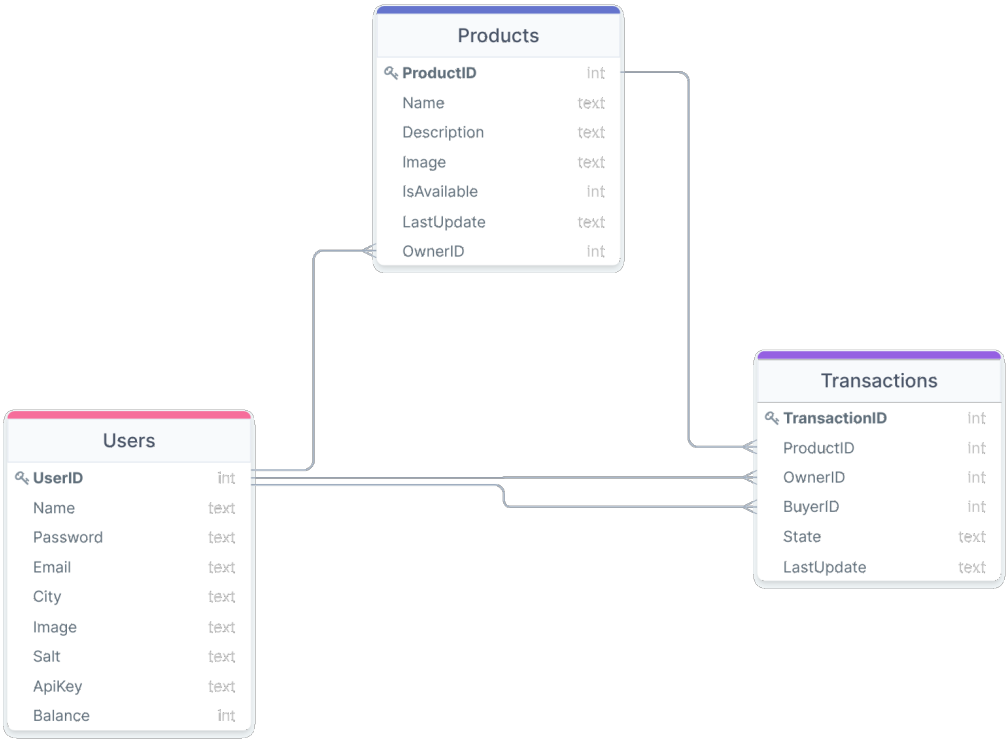


Figura 4: Modello E/R sviluppato tramite la web app disponibile qui [6].

4.2 Modello logico

Users		
Nome Campo	**Tipo**	**Note**
UserID	INTEGER	NOT NULL, PK, AUTOINCREMENT
Name	TEXT	NOT NULL
Password	TEXT	NOT NULL
Email	TEXT	NOT NULL
City	TEXT	NOT NULL
Image	TEXT	NOT NULL
Salt	TEXT	NOT NULL
ApiKey	TEXT	NOT NULL
Balance	INTEGER	NOT NULL

Products		
Nome Campo	**Tipo**	**Note**
ProductID	INTEGER	NOT NULL, PK, AUTOINCREMENT
Name	TEXT	NOT NULL
Description	TEXT	NOT NULL
Image	TEXT	NOT NULL
IsAvailable	INTEGER	NOT NULL
LastUpdate	TEXT	NOT NULL
OwnerID	INTEGER	NOT NULL, FK(Users)

Transactions		
Nome Campo	**Tipo**	**Note**
TransactionID	INTEGER	NOT NULL, PK, AUTOINCREMENT
ProductID	INTEGER	NOT NULL, FK(Products)
OwnerID	INTEGER	NOT NULL, FK(Users)
BuyerID	INTEGER	NOT NULL, FK(Users)
State	TEXT	NOT NULL
LastUpdate	TEXT	NOT NULL

4.3 Entity Framework Core

Per la creazione del database ho scelto Entity Framework Core, una versione semplice, estendibile, open source e cross-platform della tecnologia di accesso ai dati di grande diffusione chiamata Entity Framework [7]. Questa tecnologia sviluppata direttamente da Microsoft permette agli sviluppatori di creare e gestire database tramite oggetti scritti in C# che prendono il nome di modelli, è inoltre compatibile con numerosi DBMS tra cui SQLite3 che è quello adottato in questo progetto. Entity Framework Core supporta anche il versioning del database attraverso l'uso delle migrazioni: gli sviluppatori possono fare modifiche allo schema del database per poi eseguirne successivamente il rollback nel caso non fossero più desiderate.

I database vengono quindi generati automaticamente partendo dai modelli inseriti nel DbContext che descrive il database e le tabelle che lo compongono. Il DbContext da usare viene indicato nel file Startup.cs insieme alla stringa di connessione da Entity Framework Core per collegarsi al database. I modelli invece si occupano di descrivere le entità che formano il database, all'interno di questi vengono quindi riportati tutti i campi appartenenti alle tabelle. La nomenclatura dei modelli comprende l'espressione DTO (Data Transfer Object [8]) in quanto questi oggetti vengono usati per il trasferimento dei dati dagli utenti al database all'interno del sistema distribuito sul quale è basata la piattaforma.

MainDbContext.cs

```
using riusco_mvc.Models;
using Microsoft.EntityFrameworkCore;

namespace riusco_mvc.Data
{
    public class MainDbContext : DbContext
    {
        public DbSet<UserDTO> Users { get; set; }
        public DbSet<ProductDTO> Products { get; set; }
        public DbSet<TransactionDTO> Transactions { get; set; }
        public MainDbContext(DbContextOptions<MainDbContext>
            options) : base(options) { }
    }
}
```

UserDTO.cs

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace riusco_mvc.Models
{
    public class UserDTO
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        // chiave primaria con autoincrement
        public int UserID { get; set; }
        public string Name { get; set; }
        public string Password { get; set; }
        public string Email { get; set; }
        public string City { get; set; }
        public string Image { get; set; }
        public string Salt { get; set; }
        public string ApiKey { get; set; }
        public int Balance { get; set; }

        // metodi costruttori scritti con overloading
```

```
        public UserDTO(string name, string password, string
            email, string image, string salt, string apiKey,
            int balance, string city)
        {
            Name = name;
            Password = password;
            Email = email;
            Image = image;
            Salt = salt;
            ApiKey = apiKey;
            Balance = balance;
            City = city;
        }

        public UserDTO() { }
    }
}
```

ProductDTO.cs

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace riusco_mvc.Models
{
    public class ProductDTO
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        // chiave primaria con autoincrement
        public int ProductID { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public string Image { get; set; }
        public bool IsAvailable { get; set; }
        public DateTime LastUpdate { get; set; }
        [ForeignKey("Owner")]
        public int OwnerID { get; set; }
        public UserDTO Owner { get; set; }
```

```
        // metodi costruttori scritti con overloading
        public ProductDTO(string name, string description,
            string image, DateTime lastUpdate, bool
            isAvailable, int ownerId)
        {
            Name = name;
            Description = description;
            Image = image;
            LastUpdate = lastUpdate;
            IsAvailable = isAvailable;
            OwnerID = ownerId;
        }

        public ProductDTO() { }
    }
}
```

TransactionDTO.cs

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Text.Json;

namespace riusco_mvc.Models
{
    public class TransactionDTO
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        // chiave primaria con autoincrement
        public int TransactionID { get; set; }
        [ForeignKey("Product")]
        public int ProductID { get; set; }
        public ProductDTO Product { get; set; }
        [ForeignKey("Owner")]
        public int OwnerID { get; set; }
        public UserDTO Owner { get; set; }
        [ForeignKey("Buyer")]
        public int BuyerID { get; set; }
        public UserDTO Buyer { get; set; }
```

```
        public DateTime LastUpdate { get; set; }
        public string State { get; set; }

        // metodi costruttori scritti con overloading
        public TransactionDTO(int productId, int ownerID, int
            buyerId, DateTime lastUpdate, string state)
        {
            ProductID = productId;
            OwnerID = ownerID;
            BuyerID = buyerId;
            LastUpdate = lastUpdate;
            State = state;
        }

        public TransactionDTO() { }

        // metodo override ToString()
        public override string ToString()
        {
            return JsonSerializer.Serialize(this);
        }
    }
}
```

4.4 Query

Usando Entity Framework Core per la creazione e la gestione del database le query possono essere scritte con la sintassi sviluppata da Microsoft chiamata LINQ (Language Integrated Query), questa sintassi ha il vantaggio che è direttamente integrata nel linguaggio C# [9]. Alternativamente possono essere usati anche i metodi appositi inclusi nella libreria Microsoft.EntityFrameworkCore [10], questi metodi si occupano di recuperare i dati dal database, modificare quelli già presenti ed eseguire query complesse. Entrambe le modalità per l'esecuzione delle query possono lavorare in modo asincrono, ciò rende quindi Entity Framework Core perfetto per l'ambiente web. Nel progetto ho scelto di usare la libreria Microsoft dedicata in quanto è la più veloce e quella che richiede meno l'intervento del programmatore, evitando quindi errori semplici di distrazione o tempo perso ad ideare query inutilmente complesse.

Query scritta con la sintassi LINQ

```
using Microsoft.Linq;
using Microsoft.EntityFrameworkCore;
using riusco_mvc.Models;

// query che ottiene il prodotto con chiave primaria uguale a 2
List<ProductDTO> products = _context.Products.ToList();
ProductDTO product = (from product in products
                      where product.ProductID == 2
                      select product).First();

// equivalente SQLite3
// SELECT * FROM Products WHERE Products.ProductID = 2 LIMIT 1;
```

Query con i metodi appartenenti alla libreria Microsoft

```
using Microsoft.EntityFrameworkCore;
using riusco_mvc.Models;

// query che ottiene il prodotto con chiave primaria uguale a 2
ProductDTO product = await _context.Products.FindAsync(2);

// equivalente SQLite3
// SELECT * FROM Products WHERE Products.ProductID = 2 LIMIT 1;


// query che ottiene l'utente con email uguale
   mauro.pellonara@gmail.com, nel caso piu' utenti abbiano la
   stessa email ritorna un errore
UserDTO user = _context.Users.Single(x => x.Email ==
   "mauro.pellonara@gmail.com");

// equivalente SQLite3
// SELECT * FROM Users WHERE Users.Email =
   "mauro.pellonara@gmail.com" LIMIT 1;
```

5 Infrastruttura di rete

L'infrastruttura di rete dell'azienda Rius.Co. comprende ben due reti diverse: la rete aziendale, della quale fanno parte le sottoreti degli uffici amministrativi, operativi e di segreteria, e la rete dei server che comprende i 4 server sulla quale è costruito il Marketplace. La rete dei server sarà esterna in quanto collocata all'interno di una rete VPC (Virtual Private Cloud) disponibile tramite il Google Cloud [11].

L'affitto di reti VPC è uno dei tanti servizi che offre Google per ospitare server e applicazioni web, in particolare questo servizio consiste nella creazione di una rete basata su la topologia virtuale fornita dagli utenti. Ho scelto questo servizio in quanto in passato avevo già avuto esperienze pratiche con la piattaforma cloud di Google; proprio grazie ai corsi online offerti dal programma Qwiklabs ho acquistato familiarità con le reti VPC e tutti i dispositivi che ne fanno parte [12]. Google inoltre fornisce una grande quantità di strumenti che permettono ai programmatori di concentrarsi più sui servizi che stanno sviluppando che sull'infrastruttura che li circonda. Un esempio importante è la tecnologia HA Cloud VPN che permette di collegare tramite un tunnel VPN ad alta affidabilità (99.99% garantito) la rete locale di un'azienda alle reti VPC collocate sul cloud [13]. La tecnologia Cloud VPN si basa sul protocollo IPSec ed implementa una Secure VPN di tipo site-to-site che non richiede una complessa configurazione manuale dell'utente. Le reti VPC fanno parte del nuovo approccio alle architetture di rete chiamato Software-defined networking (SDN) [14], si basano quindi sulla virtualizzazione delle reti e se ne guadagna perciò in termini di affidabilità, scalabilità, flessibilità e riduzione dei costi.

Attraverso l'HA Cloud VPN i dipendenti dell'azienda potranno collegarsi da remoto alla rete VPC che contiene i server, la sede aziendale sarà quindi collegata ad Internet tramite un ISP e il traffico dei dipendenti collegati dall'azienda passerà per un Firewall che filtrerà i messaggi in entrata ed uscita. Nella sede aziendale saranno presenti un Campus Distributor che si interfaccerà con la rete pubblica dell'ISP per ottenere l'accesso ad Internet, sarà poi presente un Building Distributor in quanto si prevede la presenza di un solo edificio e infine ci saranno due Floor Distributor, uno per il piano che ospiterà la segreteria e l'ufficio operativo e un altro per il piano che ospiterà l'ufficio amministrativo. I collegamenti saranno costruiti con cavi ethernet di categoria 6A con schermatura completa SFTP che garantiscono la stabilità della connessione grazie alla doppia schermatura, un buon bandwidth (circa 10Gbps su 100 metri) e un risparmio considerevole visto il prezzo di circa 1 euro al metro [15].

5.1 Apparati di rete

All'interno della rete aziendale saranno presenti un numero contenuto di apparati di rete fisici in quanto la parte più importante ed esposta dell'infrastruttura è contenuta nella rete VPC. I seguenti apparati di rete costituiranno quindi l'infrastruttura:

- **Due router di bordo con Firewall**, il primo sarà un router virtuale posto all'interno della rete dei server nella quale regolerà il traffico con regole statiche molto rigide implementate attraverso il Packet Filter Firewall. Il secondo sarà invece fisicamente collocato nella rete aziendale e si occuperà di ricevere la connessione dal Campus Distributor per poi filtrare i messaggi implementando delle politiche intelligenti che comprendono l'analisi dei pacchetti che attraversano la rete. Per fare ciò viene quindi implementato un Stateful Packet Inspection Firewall, in questo caso i messaggi ritenuti validi verranno inoltrati al router interno che si occuperà di recapitarli alla destinazione;

- **Un router con DHCP** che si occuperà dell'instradare i pacchetti all'interno della rete aziendale e di assegnare automaticamente gli indirizzi IP ai dispositivi. Ho scelto di non includere un server DHCP per ragioni economiche e di manutenzione, l'azienda non dispone di numerose postazioni perciò un semplice router è in grado di gestire l'instradamento dei pacchetti e il servizio DHCP senza grossi problemi;

- **Quattro switch** con ventiquattro porte ciascuno necessari in vista di futuri aggiornamenti della rete o l'acquisizione di ulteriore personale.

Non ho incluso nessun access point in quanto offrire la copertura totale di un edificio di due piani comporta una spesa non indifferente e difficile da giustificare per l'azienda che può comunque essere operativa anche senza una rete Wi-Fi, in questo caso è quindi preferibile non disporre di connettività wireless ma concentrare le spese sullo sviluppo di un'infrastruttura robusta ed espandibile. È qua riportata la tabella contenente tutte le sottoreti previste nell'infrastruttura di rete.

Sottoreti					
Nome	**N. Host**	**Indirizzo di rete**	**Gateway**	**Subnet Mask**	**Broadcast**
Segreteria	16	192.168.0.0	192.168.0.1	255.255.255.240	192.168.0.15
Sede Amm.	16	192.168.0.16	192.168.0.17	255.255.255.240	192.168.0.31
Sede Op.	16	192.168.0.32	192.168.0.33	255.255.255.240	192.168.0.47
Server	16	192.168.1.0	192.168.1.1	255.255.255.240	192.168.1.15

5.2 Server

L'infrastruttura prevede l'implementazione di ben 4 diversi tipi di server divisi su 3 strati o tier, ogni strato è in grado di comunicare esclusivamente con lo strato direttamente superiore o inferiore al proprio. Questa architettura è considerata ottimale in quanto rispetta i principi di scalabilità verticale ed orizzontale delle applicazioni, è inoltre alla base dei sistemi distribuiti con interfaccia web. L'architettura a 3 tier è particolarmente adatta per l'ambiente web in quanto è molto comune avere diversi componenti slegati tra loro che concorrono e cooperano per il funzionamento della web app [16].

Il primo strato è chiamato Presentation Layer e comprende sia il Web server che l'API server, questo è l'unico strato in cui i dispositivi sono esposti all'esterno e raggiungibili direttamente tramite il proprio indirizzo IP. Gli utenti nel richiedere una pagina web o fare una richiesta all'API interagiscono direttamente con questo strato che si occupa di eseguire una prima elaborazione di queste richieste prima di inoltrare i dati al livello sottostante. Sono presenti entrambi i server in questo strato in quanto entrambi si occupano di rispondere alle richieste di client che sono però di tipo diverso. L'API server è stato inserito anche per rendere più facile il futuro sviluppo dell'applicazione mobile che potrebbe essere sviluppata come un semplice front-end che dialoga con l'API tramite le richieste HTTPS.

Subito dopo troviamo invece l'Application Layer, da alcuni chiamato Business Logic Layer, questo contiene l'Application server nel quale sono presenti tutti i programmi e i processi che rappresentano e rispettano la logica dell'applicazione o sito web. Le funzioni appartenenti all'Application Layer vengono invocate direttamente dal Presentation Layer che invia anche i dati necessari per completare l'elaborazione delle richieste degli utenti. Dopo aver elaborato i dati, l'Application server ha il compito di decretare cosa farne, nel caso fosse necessario salvarli localmente li invierà allo strato inferiore, altrimenti saranno inviati al Presentation Layer che si occupa di rispondere alle richieste degli utenti. Questo è forse lo strato più importante perché senza di esso il resto sarebbe totalmente inutile in quanto non conterrebbe nulla della logica con la quale è stata ideata la web app.

Abbiamo infine lo Storage Layer che comprende il data server, quest'ultimo conserva e gestisce l'accesso a tutti i dati relativi al programma e salvati all'interno di database o cartelle locali. Nel caso di Rius.Co. oltre al database il data server contiene anche le immagini di profilo degli utenti e quelle legate ai prodotti. In questo strato viene condotta un ulteriore rifinitura dei dati per accertarsi che gli utenti non stiano provando a danneggiare l'infrastruttura attraverso attacchi come l'SQL injection.

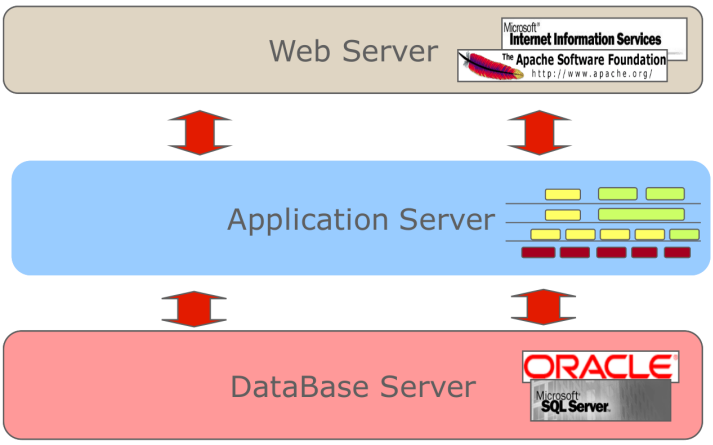
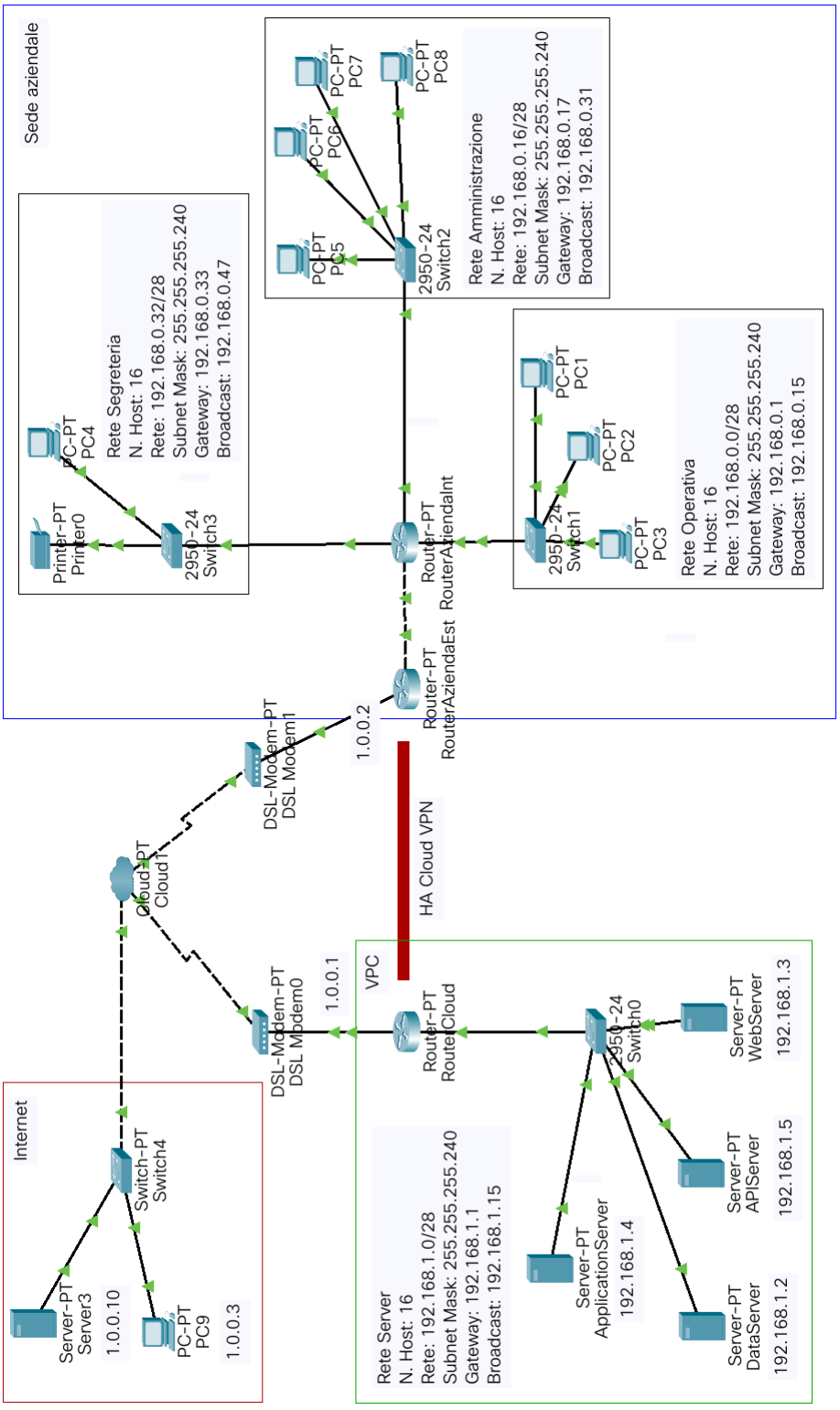


Figura 5: Rappresentazione grafica dell'architettura a 3 tier.

Questi server non esistono fisicamente all'interno dell'infrastruttura in quanto sono delle macchine virtuali generate tramite il Google Kubernetes Engine e collocate all'interno della rete VPC, ciò offre numerosi vantaggi in quanto è possibile affidare alla piattaforma Google la scalabilità delle risorse, la manutenzione dei server e la gestione dei dati. Il Google Kubernetes Engine consiste in un servizio cloud di virtualizzazione a metà tra l'Iaas (Infrastructure-as-a-Service) e il Paas (Platform-as-a-Service) attraverso il quale gli utenti possono creare dei cluster di container Docker all'interno dei quali viene eseguito del codice fornito dagli sviluppatori [17]. Questi container di fatto lavorano come delle macchine virtuali ma garantiscono maggiore virtualizzazione e quindi più efficienza e versatilità. Tramite il Google Kubernetes Engine è possibile creare dei cluster di container identici (stesso sistema operativo e hardware) che vengono replicati a seconda del workload globale e dei criteri impostati dall'amministratore. È anche possibile inserire dei Load Balancer che si occupino di dividere equamente la mole di lavoro all'interno dei cluster di macchine virtuali, insieme a questi è consigliato integrare anche degli Health Checks che in caso di disservizi eseguono azioni predefinite come l'invio istantaneo di un email di notifica all'amministratore.

5.3 Progetto di rete

Ho progettato la rete aziendale insieme alla rete VPC usando il programma di simulazione Cisco Packet Tracer 8.0, nelle etichette da me inserite sono riportati gli indirizzi IP degni di nota e le informazioni relative alle 4 sottoreti create. Il file è stato caricato assieme a tutto il progetto nell'apposita repository creata su GitHub e raggiungibile qui [5].


Sede aziendale
Rete Segreteria
N. Host: 16
Rete: 192.168.0.32/28
Subnet Mask: 255.255.255.240
Gateway: 192.168.0.33
Broadcast: 192.168.0.47
Rete Amministrazione
N. Host: 16
Rete: 192.168.0.16/28
Subnet Mask: 255.255.255.240
Gateway: 192.168.0.17
Broadcast: 192.168.0.31
Rete Operativa
N. Host: 16
Rete: 192.168.0.0/28
Subnet Mask: 255.255.255.240
Gateway: 192.168.0.1
Broadcast: 192.168.0.15
Printer-PT Printer0
PC-PT PC4
2950-24 Switch3
PC-PT PC5
PC-PT PC6
PC-PT PC7
PC-PT PC8
2950-24 Switch2
PC-PT PC1
PC-PT PC2
PC-PT PC3
2950-24 Switch1
Router-PT RouterAziendaInt
Router-PT RouterAziendaEst
1.0.0.2
DSL-Modem-PT DSL Modem1
Cloud-PT Cloud1
HA Cloud VPN
VPC
1.0.0.1
DSL-Modem-PT DSL Mpdem0
Router-PT RouterCloud
2950-24 Switch0
Server-PT WebServer
192.168.1.3
Server-PT APlServer
192.168.1.5
Server-PT ApplicationServer
192.168.1.4
Server-PT DataServer
192.168.1.2
Rete Server
N. Host: 16
Rete: 192.168.1.0/28
Subnet Mask: 255.255.255.240
Gateway: 192.168.1.1
Broadcast: 192.168.1.15
Internet
Server-PT Server3
1.0.0.10
Switch-PT Switch4
PC-PT PC9
1.0.0.3

6 Politiche di sicurezza

Gli e-commerce, più di molti altri siti web, devono garantire un livello di sicurezza più che eccellente in quanto le conseguenze causate da eventuali bug o vulnerabilità sono potenzialmente catastrofiche per gli utenti che usufruiscono della piattaforma. All'interno del ciclo di vita del software l'azienda deve definire delle rigide politiche di sicurezza volte a garantire la continuità operativa e la riservatezza delle informazioni associate agli utenti. Ho quindi definito ed adottato le seguenti politiche di sicurezza:

- **Rinominare i file caricati dagli utenti**, degli eventuali attaccanti possono caricare file con nomi particolari volti ad accedere a cartelle contenenti file sensibili che quindi risulterebbero compromessi;
- **Controllare il contenuto dei file caricati dagli utenti**, gli unici file che gli utenti possono caricare sono le immagini, viene quindi fatto un doppio controllo (lato client e lato server) per assicurarsi che i file caricati rispettino il corretto formato;
- **Uso del pattern Post/Redirect/Get**, questo pattern particolare adottato in maniera estensiva nel web permette di ovviare al problema della ricarica o navigazione temporale delle pagine e richieste già inviate al server [18]. Seguendo questo pattern tutte le richieste POST verranno prima elaborate e poi reindirizzate su la stessa o un'altra pagina che verrà ottenuta tramite richiesta GET, in questo modo l'utente nella sua navigazione non incontrerà mai messaggi relativi alla celebre "Conferma reinvio modulo";
- **HTTPS** [19], il sito usa il protocollo TLS/SSL insieme al protocollo HTTP per la navigazione sicura all'interno del sito, i dati trasmessi da e verso il sito attraversano un tunnel end-to-end dentro il quale sono crittografati e non leggibili da possibili attaccanti;
- **Autenticazione degli utenti**, gli utenti si devono autenticare per poter compiere azioni sul sito come inserire un prodotto;
- **Password hashing**, le password usate dagli utenti non vengono salvate in chiaro dal database che invece le memorizza già crittografate;
- **Uso di un Firewall**, i dispositivi essendo connessi in rete possono anche essere soggetti di attacchi informatici basati sull'invio di pacchetti maliziosi, l'uso di un Firewall permette di bloccare questi pacchetti e garantire una connessione sicura.

6.1 Autenticazione

La piattaforma ha due modalità di accesso: il sito web e l'API. Entrambe le modalità devono effettuare l'autenticazione dell'utente che s'interfaccia con la piattaforma Rius.Co. verificandone l'identità.

Il sito web si occupa dell'autenticazione richiedendo agli utenti di impostare una password di lunghezza variabile tra gli 8 e i 24 caratteri. I clienti possono anche richiedere la modifica della password che è comunque una pratica da fare periodicamente per evitare problemi relativi alla sicurezza del profilo.

L'API invece esegue l'autenticazione dell'utente in tutt'altro modo, gli utenti alla registrazione ricevono una chiave univoca privata (API key) di 32 byte che viene generata in modo casuale e non è modificabile [20]. Questa chiave dovrà quindi essere inserita nel form di tutte le richieste fatte all'API per permettere l'identificazione dell'utente associato alla richiesta. Alcune richieste come ottenere un file in formato JSON contenente tutti i prodotti presenti sul Marketplace non necessitano dell'autenticazione e vengono eseguite anche senza l'API key in quanto sono informazioni pubbliche visualizzabili da chiunque.

Esempio di richiesta con autenticazione all'API tramite curl

```
$ curl -X GET --insecure
   "https://0.0.0.0:6066/Transactions/GetTransactionsByUserID/9" -F
   "apiKey=aBQKfxy175kL1v5EdOwCkHWtgwAj9Kbzu339OkDIgxA="
```

Risposta del server in formato JSON contenente tutte le transazioni in cui ha partecipato l'utente con l'API key indicata


```
[
    {
        "transactionID": 8,
        "productID": 12,
        "ownerID": 9,
        "buyerID": 10,
        "lastUpdate": "2021-05-26T21:53:52.5569413",
        "state": "Completed"
    },
    {
        "transactionID": 7,
        "productID": 2,
        "ownerID": 10,
```

```
        "buyerID": 9,
        "lastUpdate": "2021-05-18T11:51:42.1961266",
        "state": "Pending"
    },
    {
        "transactionID": 4,
        "productID": 13,
        "ownerID": 9,
        "buyerID": 10,
        "lastUpdate": "2021-05-17T12:42:48.5219505",
        "state": "Closed"
    }
]
```


6.2 Password hashing

Per garantire la privacy degli utenti ed evitare che i profili vengano violati in caso di data breach o altri tipi di fughe di informazioni è necessario rendere praticamente impossibile ottenere le password impostate dagli utenti. Le password devono quindi essere crittografate con algoritmi che rendano impraticabile la decifrazione anche in caso di attacchi come il brute-force. Per la piattaforma Rius.Co. ho deciso di usare gli standard più recenti e robusti dal punto di vista informatico per eseguire l'hashing di tutte le password. L'hashing consiste nel produrre una stringa o digest correlata ad una serie di dati che sono in questo caso la password, una buona funzione di hashing rende praticamente impossibile risalire alla password partendo dal digest ma restituisce sempre lo stesso digest nel caso venga usata la stessa password [21]. In questo modo nel database è riportata solo l'elaborazione della password dalla quale nessuno può risalire all'originale.

Per eseguire l'hashing delle password ho implementato la funzione di derivazione della chiave consigliata sia da Microsoft [22] che dal RFC 8018 [23], questa è definita nel RFC 2898 [24] e prende il nome di PBKDF2 (Password-Based Key Derivation Function 2) [25]. Questa funzione è ritenuta la migliore dal punto di vista di semplicità, prestazioni e sicurezza; restituisce una chiave derivata (DK) che equivale all'hash della password ed è rappresentata in questo modo:

$$\mathrm{DK} = \mathrm{PBKDF2}(\mathrm{PRF},\ \mathrm{Password},\ \mathrm{Salt},\ \mathrm{c},\ \mathrm{dkLen})$$

Per ottenere la chiave derivata sono quindi richiesti i seguenti parametri:

- PRF, la funzione pseudocasuale che si occuperà ad ogni iterazione di eseguire l'hashing dei dati con la generazione del Message Authentication Code (HMAC);
- Password, la password della quale eseguire l'hashing e ottenerne la chiave derivata;
- Salt, una sequenza casuale di byte assegnata automaticamente all'utente e salvata in chiaro nel database. Il salt viene sommato alla password dell'utente prima dell'hashing per far sì che due password uguali di due utenti diversi producano hash altrettanto diversi, rendendo ancora più difficile la decifrazione tramite brute-force [26];
- c, il numero di iterazioni da svolgere per ottenere la chiave derivata, l'aumentare del numero di iterazioni comporta l'importante aumento sia del livello di sicurezza che del tempo di esecuzione della funzione che perde in velocità;
- dkLen, la lunghezza in byte della chiave derivata.

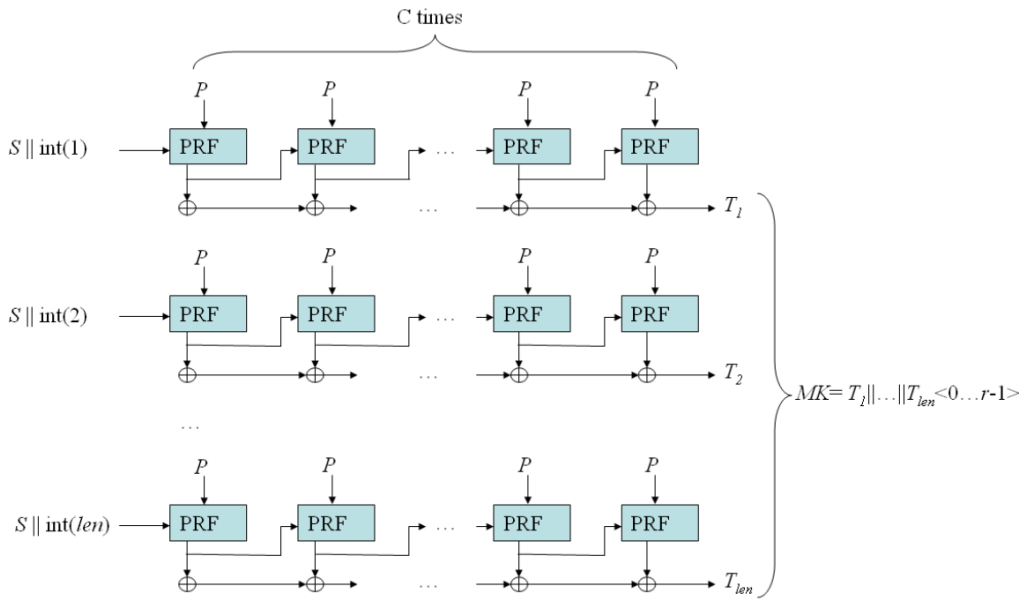


Figura 6: Rappresentazione grafica dell'algoritmo che compone un'iterazione.

Nel mio caso PBKDF2 esegue 10000 iterazioni con l'algoritmo di hashing HMACSHA256, usa un Salt di 128 byte, produce un'elaborazione della password lunga 32 byte ed è implementato all'interno della seguente funzione.

```
private static string GetHash(string password, string salt)
{
    return Convert.ToBase64String(KeyDerivation.Pbkdf2(
        password: password,
        salt: Encoding.UTF8.GetBytes(salt),
        prf: KeyDerivationPrf.HMACSHA256,
        iterationCount: 10000,
        numBytesRequested: 32));
}
```

6.3 Firewall

Un firewall consiste in un componente hardware o software che si occupa della difesa di una rete, questo analizza i pacchetti in entrata ed uscita e attraverso le regole fornite dall'amministratore di rete decide se permettere o bloccare il passaggio di questi pacchetti [27]. Le regole prendono il nome di access-list e definiscono il traffico che ha il permesso di attraversare la rete. Nel caso dell'infrastruttura sulla quale si appoggia Rius.Co. sono presenti due router che svolgono entrambi anche la funzione di firewall, questi sono collocati al bordo delle due reti, ovvero quella aziendale e quella contenente i server. Nel router di bordo collocato nella rete dei server il traffico nella sua completezza è permesso solo dalla rete aziendale, sono state però impostate due access-list per permettere l'accesso all'API server e al Web server che devono poter essere raggiungibili da chiunque.

Configurazione del firewall collocato sul router di bordo della rete dei server con indirizzo IP pubblico 1.0.0.1

```
interface GigabitEthernet0/0
   ip address 1.0.0.1 255.0.0.0
   ip access-group 101 in

access-list 101 permit ip any host 192.168.1.5
access-list 101 permit ip any host 192.168.1.3
access-list 101 permit ip 192.168.0.0 0.0.255.255 any
access-list 101 deny ip any any
```

Riferimenti bibliografici

[1] S2S Group. 10 fatti scioccanti riguardanti l'e-waste. URL: https://s2s.uk.com/news/10-shocking-facts-from-the-global-e-waste-monitor/.

[2] ONU Italia. L'Agenda 2030 presentata direttamente dall'ONU. URL: https://unric.org/it/agenda-2030/.

[3] Mauro Pellonara. Prototipo navigabile dell'app mobile. URL: https://mauro886267.invisionapp.com/console/share/7Z10U19EHJ/476334736.

[4] Wikipedia. Presentazione del DBMS SQLite3. URL: https://en.wikipedia.org/wiki/SQLite.

[5] Mauro Pellonara. Repository dedicata al progetto caricata su GitHub. URL: https://github.com/MauroPello/elaborato.

[6] DrawSQL. Web app per lo sviluppo di modelli E/R. URL: https://drawsql.app.

[7] Documentazione Microsoft. Descrizione della tecnologia Entity Framework Core. URL: https://docs.microsoft.com/en-us/ef/core/.

[8] Stack Overflow. Documentazione della libreria Microsoft.EntityFrameworkCore. URL: https://stackoverflow.com/questions/1051182/what-is-a-data-transfer-object-dto.

[9] Documentazione Microsoft. Descrizione della sintassi LINQ. URL: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/query-syntax-and-method-syntax-in-linq.

[10] Documentazione Microsoft. Documentazione della libreria Microsoft.EntityFrameworkCore. URL: https://docs.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore?view=efcore-5.0.

[11] Documentazione Google Cloud. Documentazione della tecnologia VPC. URL: https://cloud.google.com/vpc/docs/overview.

[12] Mauro Pellonara. Pagina pubblica della piattaforma Google Qwiklabs. URL: https://google.qwiklabs.com/public_profiles/9e3203d3-fe41-485a-b4a5-7a3b7b331bc1.

[13] Documentazione Google Cloud. Documentazione della tecnologia HA Cloud VPN. URL: https://cloud.google.com/network-connectivity/docs/how-to/choose-product#cloud-vpn.

[14] Wikipedia. Il Software-defined networking. URL: https://en.wikipedia.org/wiki/Software-defined_networking.

[15] Alibaba. Cavi di Categoria 6A e schermatura SFTP. URL: https://www.alibaba.com/showroom/cat6a-price.html.

[16] Altexsoft. Architettura a 3 tier per le web app. URL: https://www.altexsoft.com/blog/engineering/web-application-architecture-how-the-web-works/.

[17] Documentazione Google Cloud. Documentazione della tecnologia Kubernetes. URL: https://cloud.google.com/kubernetes-engine/docs.

[18] Stack Overflow. Descrizione del pattern Post/Redirect/Get. URL: https://stackoverflow.com/questions/4327236/stop-browsers-asking-to-resend-form-data-on-refresh.

[19] Cloudflare. Descrizione del protocollo HTTPS. URL: https://www.cloudflare.com/learning/ssl/what-is-https/.

[20] Wikipedia. Uso delle API key per l'autenticazione. URL: https://en.wikipedia.org/wiki/Application_programming_interface_key.

[21] Wikipedia. Funzione di hashing. URL: https://it.wikipedia.org/wiki/Funzione_di_hash.

[22] Documentazione Microsoft. Eseguire l'hashing delle password in ASP.NET Core. URL: https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/consumer-apis/password-hashing?view=aspnetcore-5.0.

[23] IETF. Indicazioni sul password hashing, versione aggiornata del RFC 2898. URL: https://datatracker.ietf.org/doc/html/rfc8018.

[24] IETF. Descrizione della funzione PBKDF2. URL: https://datatracker.ietf.org/doc/html/rfc2898#section-5.2.

[25] Wikipedia. Funzione di derivazione della chiave PBKDF2. URL: https://en.wikipedia.org/wiki/PBKDF2.

[26] Stack Exchange. Password hashing, l'utilizzo del Salt e le Rainbow tables. URL: https://crypto.stackexchange.com/questions/35275.

[27] Cisco. Cos'è un Firewall e come funziona. URL: https://www.cisco.com/c/en/us/products/security/firewalls/what-is-a-firewall.html.