SAPIENZA
UNIVERSITÀ DI ROMA

iMenuManager - Progettazione e sviluppo di un'applicazione web per la gestione di attività di ristorazione

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Dipartimento di Informatica
Corso di laurea in Informatica

Alberto Cotumaccio
Matricola 1852040

Responsabile
Emiliano Casalicchio

Corresponsabile
Manuel Covuccia

A.A. 2020-2021

Indice

1. Introduzione

1.1 Le motivazioni ed il problema affrontato

IMenuManager è un'applicazione web di gestione sviluppata e pensata per un'Organizzazione che gestisce più mense all'interno di realtà industriali (da ora in avanti chiamate impianti) su tutto il territorio nazionale.

L'oggetto del tirocinio è stato quello di creare un'applicazione web, costituita da un'interfaccia web multimodale, un database e un insieme di funzionalità fruibili via browser indipendentemente dal dispositivo utilizzato (computer, telefono, tablet), con lo scopo di **automatizzare i task abituali all'interno dell'organizzazione** ed ottimizzare i loro tempi di esecuzione.

Questo è un momento in cui l'informatica è diventata un elemento chiave per molte aziende, necessaria per essere sempre più presenti e competitive in un mercato in continua evoluzione tecnologica. Non basta infatti curare solo la presenza dell'azienda sul web ma è anche indispensabile intervenire con sistemi interni mirati a gestire e controllare i propri costi di gestione per rendere più agevoli ed economiche le attività svolte al proprio interno.

La progettazione del portale "IMenuManager" ha avuto come idea di partenza, e quindi di primaria importanza per l'architettura del sistema, quella di garantire la possibilità di gestire i menu settimanali di tutti gli impianti nelle diverse aree d'Italia.

Prima della creazione di questa applicazione web, l'organizzazione effettuava le operazioni di gestione in maniera difficoltosa e macchinosa, ad esempio tramite fogli excel o tramite scambio di email, senza avere quindi un'infrastruttura centralizzata accessibile in remoto da tutti comportando varie difficoltà per tutti gli utenti che ne fanno parte.

La creazione del portale va a **ridurre estremanente i costi e tempi di amministrazione** da parte della direzione e degli chef, aumentando facilità e immediatezza delle procedure e riducendo la quantità di errori commessi.

Il sistema deve garantire anche una politica di sicurezza adeguata per mantenere integrità e confidenzialità dei dati e dovrà essere mantenuto aggiornato sotto questo aspetto per ridurre eventuali attacchi esterni od interni.

Il portale nel suo complesso deve risultare:

- **Usabile**, tanto da permettere a tutti gli utenti di comprendere in ogni istante in che pagina si trovano e l'operazione che stanno eseguendo.
- **Veloce**, rendendo la consultazione e l'utilizzo di informazioni immediata, ottimizzando il più possibile le query al database.
- **Sicuro**, la sicurezza del sistema è stato un elemento alla base del progetto. Ogni utente che naviga nel portale deve essere registrato ed essersi autenticato, evitando l'accesso e la divulgazione di informazioni private, come ad esempio ricette e menu, da parte di estranei.

I principali obiettivi che sono stati perseguiti per la realizzazione del portale possono essere riassunti nei seguenti punti:

1. Creare una **infrastruttura centralizzata**, ovvero un punto di accesso unico e trasversale per tutte le utenze appartenenti ai vari impianti nelle diverse regioni d'Italia. Questo

comporta sicuramente molteplici benefici, tra questi la possibilità di avere una maggiore organizzazione tra i vari impianti garantendo anche un collegamento efficiente ed efficace tra gli stessi, avendo per esempio un database unico e consultabile da tutti.

2. Fornire un punto di incontro tra gli chef e gli utenti amministratori

3. Ottimizzare i processi, e permettere quindi di svolgere la maggior parte delle attività caratterizzanti dell'organizzazione senza doversi recare in sede, velocizzando e semplificando le procedure esistenti e la possibilità di essere contemporaneamente fruibile su diversi tipi di dispositivi.

Le figure, ovvero coloro che possono usufruire dei servizi del portale e che possono utilizzarlo, sono solamente utenti autenticati e registrati, questi si dividono nelle seguenti categorie: **chef, chef manager, capi area e utenti direzionali** *(solamente alla fine della fase di testing sono stati aggiunti anche i direttori d'impianto).*

Tutti gli utenti hanno funzionalità simili, possono cioè:

- *Accedere agli impianti* a cui sono stati associati: gli chef ne hanno uno solo, un capoarea può accedere agli impianti che si trovano nella sua area di competenza ed infine un utente direzionale può accedere e modificare qualunque impianto. Quando si entra nella pagina di un impianto si possono visualizzare le 52 settimane dell'anno corrente, e queste possono avere le tre seguenti condizioni:
  - Il menu della settimana ancora non esiste;
  - Il menu della settimana esiste ma deve essere approvato dalla direzione;
  - Il menu della settimana esiste ed è approvato;
- *Creare le ricette*, queste possono essere associate ad uno chef proprietario rispettando i seguenti criteri: uno chef può assegnarla solamente a sè stesso; gli chef manager possono assegnarla ad un qualunque chef che si trova all'interno dell'impianto; un capoarea invece può associarla agli chef manager presenti negli impianti che gestisce. Infine un utente direzionale può assegnarla ad un qualunque chef manager presente nel sistema, o anche a nessuno, rendendo la ricetta anonima.
- *Creare i menu settimanali* all'interno di un impianto a cui è possibile accedere. Tutti gli utenti possono inserire qualunque ricetta approvata che si trova all'interno del sistema, inoltre è possibile inserire anche le ricette non approvate ma solo se lo chef proprietario appartiene al medesimo impianto. Esistono anche le ricette speciali, che però possono essere inserite nei menu degli impianti in base alla classificazione di quest'ultimi.

Figura 1.1: Diagramma degli stati per un menu di un certo impianto

Gli utenti direzionali possiedono molte funzionalità di sistema aggiuntive, come per esempio registrare gli utenti, modificare i listini degli ingredienti dei vari impianti, aggiungere nuovi fornitori ecc.

1.2 La struttura della tesina

Il portale IMenuManager è stato realizzato in **fasi sequenziali** e a stretto contatto con lo staff dell'organizzazione, definendo e sviluppando man mano le funzionalità e le caratteristiche che sono state ritenute importanti.

La tesina, di conseguenza segue l'andamento del lavoro svolto ed è divisa in **5 parti fondamentali** che sono:

1. Nel capitolo 2 viene discussa la selezione delle migliori tecnologie adatte per la creazione di questa applicazione web, raccogliendo informazioni su differenti framework ed estraendo pregi e difetti di ognuno.

2. La raccolta di tutti i requisiti dell'applicazione, concordati di volta in volta con lo staff dell'azienda, mantenendo quindi un contatto costante per questi approfondimenti, sono descritti nel capitolo 3.

3. La progettazione della soluzione, che è stato un processo che ha visto l'evolversi della creazione dell'architettura del sistema, partendo da diagramma ER dell'applicazione fino ad arrivare allo schema della base dati e la progettazione delle funzionalità, ciò è descritto nel capitolo 4.

4. Nel capitolo 5 viene descritta l'implementazione delle funzionalità, utilizzando il linguaggio di programmazione Python e di varie sue librerie.

5. Infine, dopo il deployment su un server linux remoto, è stata effettuata una fase di testing direttamente dagli attori del sistema, che ha portato all'aggiunta di nuovi requisiti, alla correzione di alcune funzionalità e all'aggiunta di altre nuove. Ciò è descritto nel capitolo 5.

2. Tecnologie fondamentali

Per lo sviluppo del portale è richiesta *un'interfaccia web* con cui gli utenti interagiscono con la piattaforma, *un web server* per gestire e soddisfare le richieste di questi ultimi ed infine un *database relazionale* per mantenere e gestire i dati. Prima di trovare le tecnologie adeguate sono state effettuate delle ricerche per estrapolare i pregi ed i difetti di ognuna per capire quali fossero le migliori soluzioni da adottare. In particolare ho sfruttato la tecnologia **Django** per la progettazione del back-end, associato al framework **Bootstrap** per il lato del front-end e per la creazione delle pagine html. Per quanto riguarda l'infrastruttura, visto l'approccio centralizzato, è bastato un server per rendere disponibile l'accesso al servizio, utilizzando un database **SQLite**.

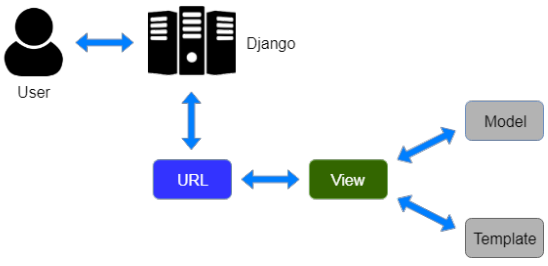
2.1 Django

Django è un web framework con licenza open source per lo sviluppo di applicazioni web, scritto in linguaggio **Python**, seguendo il paradigma di progettazione *"Model-View-Template"* che consiste in una raccolta di tre componenti importanti:

- **Model**. È un'insieme di classi Python per la rappresentazione del modello dei dati. Queste classi forniscono una rappresentazione per le tabelle del database e questo ci consente di utilizzare gli oggetti per effettuare le operazioni *CRUD*[1] sui dati, al posto delle classiche query SQL. Si parla in questo caso di ORM, Object-Relational Mapping. Tutte le definizioni delle classi sono contenute nel file *models.py* nella cartella dell'applicazione.

- **View**. Si tratta di funzioni Python che gestiscono il flusso dell'applicazione. Grazie a queste funzioni possiamo definire le pagine all'interno dell'applicazione e i comportamenti che queste pagine avranno in funzione dell'interazione con l'utente. In una View è possibile prendere una serie di dati dal Model, elaborarli e decidere quale Template utilizzare per mostrare i dati all'utente finale.

- **Template**. E' costituito da parti statiche dell'output HTML desiderato e da una sintassi speciale che descrive come verrà inserito il contenuto dinamico.



[1] Con il nome CRUD (Create, Read, Update, Delete) si intendono le operazioni di inserimento, cancellazione e aggiornamento di record.

Quando si costruisce un'applicazione web, si ha sempre bisogno di un insieme di *componenti simili*: un sistema per gestire l'autenticazione dell'utente (registrazione, accesso, logout), un pannello di amministrazione, un sistema per caricare i file, ecc.

L'idea di partenza degli sviluppatori di Django è stata quella di fornire agli sviluppatori web, componenti già pronte all'uso lasciandoli concentrare sulle parti importanti che devono sperimentare. D'altronde I frameworks consistono in un insieme di componenti, ed esistono per evitare di reinventare la ruota e per velocizzare e facilitare il lavoro quando si crea un nuovo sito.

Il web server legge le richieste in entrata, ed ognuna di queste viene passata a Django che prova a capire che cosa è stato chiesto creando il contenuto ed inviando una risposta sotto forma di pagina web. Django prende quindi l'indirizzo della pagina web e cerca di capire cosa deve fare. Questa parte viene svolta da **Django urlresolver** (il nome urlresolver acquista significato dal fatto che l'indirizzo di una pagina web si chiama URL - Uniform Resource Locator).

Django ha la necessità di possedere una serie di **schemi** ed in seguito prova a far corrispondere l'URL. Quindi controlla questi modelli o schemi e se trova corrispondenze passa alla richiesta della funzione associata (che si chiama view).

Nella funzione view avvengono tutte le cose più interessanti: è possibile ad esempio consultare il database per la ricerca di qualche informazione, ha inoltre la capacità di poter controllare se l'utente che ha fatto la richiesta è effettivamente autorizzato ed infine la view genera una risposta e Django la può inviare al browser dell'utente.

Attualmente ci sono molte aziende che utilizzano Django come framework per lo sviluppo della loro applicazione web, tra le più importanti:

- **Instagram**. E' un'app di social network per la condivisione di foto e video basata su Python che elabora enormi quantità di dati e gestisce un numero ancora maggiore di interazioni tra più utenti ogni secondo. Questo Framework aiuta Instagram a gestire tutto questo lavoro mantenendolo semplice e non reinventando il lavoro.

- **Spotify**. Ha portato l'industria della musica al livello successivo cambiando il modo in cui le persone ascoltano la musica e rendendola accessibile a chiunque su qualsiasi dispositivo. Spotify utilizza Python sia per i servizi di backend che per l'apprendimento automatico in combinazione con Django Framework.

- **YouTube**. E' una delle piattaforme di condivisione di contenuti popolari. YouTube era un progetto basato su PHP ma per migliorare le sue prestazioni YouTube si è trasferito a Django, il che li aiuta ad agire in modo impeccabile.

- **Dropbox**. E' uno dei rinomati servizi di archiviazione cloud per documenti, video e immagini. Si basa su Python per software client desktop e server. Dropbox utilizza Django Framework per abilitare l'archiviazione, la sincronizzazione e fornire opzioni per la condivisione di vari tipi di file.

- **Mozilla**. E' tra i browser più popolari ed ha milioni di utenti in tutto il mondo. I loro vecchi componenti non sono scritti in Python ma i nuovi componenti sono implementati usando Django. Si sono spostati da PHP (CakePHP) a Python (Django Framework) che li ha aiutati a gestire decine, centinaia e milioni di visualizzazioni al mese e più hit API al giorno.

- **Disqus**. Questo è il più grande progetto realizzato usando Django. Gli sviluppatori di Disqus hanno creato l'app da zero usando Django per ridimensionarla per gestire milioni di utenti al giorno. Hanno anche usato Django in uno dei loro progetti chiamato Sentry, uno strumento di segnalazione degli errori che è anche famoso con gli sviluppatori di questi giorni.

Le caratteristiche fondamentali di Django sono:

Portabilità E' scritto con il linguaggio di programmazione Python e ne eredita la sua portabilità e permette quindi di utilizzarlo senza problemi in tutti i sistemi operativi più comunemente diffusi. Semplice da imparare e facile da implementare, è possibile gestire molto in poche righe grazie all'utilizzo di librerie di grandi dimensioni. Django e Python sono soluzioni chiave per le aziende Fintech nella Silicon Valley, giganti IT, società blue chip e Internet delle cose.

Manutenibilità Rispetta il principio di scrittura software **DRY** ovvero *"Don't Repeat Yourself"* e con la sua architettura modulare permette di mantenere il codice robusto e aggiornato.

Sicurezza Protezione automatica per vulnerabilità come *sql injection, cross-site scripting e cross-site request forgery.*

Versatilità Utile per la creazione di qualsiasi tipologia di sito. E fornisce contenuto in tanti formati come *HTML, JSON, RSS* e supporta nativamente database quali *Mysql, postgresql e sqlite.*

Interfaccia amministrativa L'interfaccia di amministrazione fornita da Django è semplice da creare e uno dei principali vantaggi di un framework. Possiede un'interfaccia di amministrazione completa.

Completezza Fornisce ampie librerie di supporto che includono operazioni su stringa, servizi Web, interfaccia del sistema operativo e protocollo standard.

Scalabilità Per gestire il traffico più intenso, viene infatti utilizzato per soddisfare le molteplici richieste nei siti più trafficati. Consente di eseguire diverse azioni relative alla scalabilità come l'esecuzione di server separati per il database e persino l'uso di clustering o bilanciamento del carico per distribuire l'applicazione su più server. Ha quindi la possibilità di accogliere milioni di utenti grazie all'architettura utilizzata.

Documentazione Nel sito web è possibile consultare molta ducomentazione ed è organizzata in sezioni.

2.2 Bootstrap

Il Web si basa su **HTML**, **CSS** e **JavaScript**: qualsiasi sito li usa e chiunque voglia realizzarne uno completamente personalizzato deve occuparsi di scrivere codice per delinearne il contenuto (se html), per tracciarne l'aspetto (se CSS) o per permettere l'interazione dell'utente (se JavaScript).

Proprio qui entra in gioco Bootstrap: creato dai programmatori di Twitter (prima di diventare open-source era infatti conosciuto come Twitter Blueprint), è una libreria di componenti front-end pronte all'uso usata per creare pagine web reattive e mobile-first utilizzando HTML, CSS e JavaScript.

L'idea alla base di questo framework è quella di permettere agli sviluppatori di realizzare siti e applicazioni Web di alta qualità in un tempo ridotto. Questi, infatti, non devono più preoccuparsi (o quasi) di numerose aspetti, primi fra tutti la grafica e la user interaction. Il suo scopo è proprio quello di uniformare tutti i componenti di un'interfaccia Web per renderli riutilizzabili con facilità.

Bootstrap usa tecnologie flexbox (sistema di griglie flessibili) e mixin (classi che contengono metodi utilizzabili da altre classi, senza che esista un rapporto genitore-figlio tra queste due classi).

Tra le caratteristiche principali ci sono:

- **Semplicità di utilizzo**, basta poco per iniziare a padroneggiare le funzioni, anche quelle più evolute. Ed è quindi adatto a realizzare pressoché qualsiasi sito.
- **Supporto al design responsive**, introdotto con la versione 2.0, permette di creare pagine con un layout che si adatta automaticamente al tipo di dispositivo usato. Per esempio il contenuto visualizzato su uno smartphone è presentato in modo differente rispetto a quello mostrato su un PC.
- **completamente gratuito**, si tratta di un progetto open source, disponibile all'utilizzo a chiunque lo desideri.
- **Documentazione completa**, semplice da consultare e tradotta in più lingue. Presenta anche degli esempi d'uso pratico per rendere la comprensione ancora più agevole.

2.3 SQLite

SQLite è un database utilizzato in milioni di software e dispositivi mobili. E' stato inventato da D.Richard Hipp nell'agosto 2000. Lite in SQLite significa leggerezza in termini di installazione, amministrazione del database e risorse necessarie.

SQLite è un pacchetto software di dominio pubblico che fornisce un sistema di gestione di database relazionale o RDBMS. I sistemi di database relazionali vengono utilizzati per archiviare record definiti dall'utente in tabelle di grandi dimensioni.

Oltre all'archiviazione e alla gestione dei dati, un motore di database può elaborare comandi di query complessi che combinano dati da più tabelle per generare report e riepiloghi di dati.

Un database SQLite viene conservato in un singolo file nel file system. La libreria gestisce l'accesso al file, incluso il lock per prevenire corruzione di dati quando viene usato in scrittura da utenti multipli. Il database viene creato la prima volta in cui si accede al file, ma l'applicazione è responsabile per la gestione delle definizioni della tabella, note come schema.

Mantenere i dati in un singolo file di testo (ad esempio con excel) non è molto efficiente; leggere e archiviare dati potrebbe esserlo ancora meno. I database organizzano i dati nell'ordine corretto in modo che la lettura e la scrittura degli stessi siano molto più veloci.

I dati possono essere strutturati, semi-strutturati o non strutturati. I database vengono utilizzati principalmente per archiviare dati strutturati e semi-strutturati. I database possono essere suddivisi come segue in base al tipo di struttura di dati che verrà archiviata:

- **Database relazionale**. Tipo di database comunemente usato che ha una struttura a tabella. Le tabelle possono avere relazioni con altre tabelle. Il linguaggio SQL è utilizzato per manipolare i dati su questo tipo di database.

- **Database dei chiave-valore**. I dati vengono memorizzati assieme a una chiave associata. I dati possono essere recuperati con la chiave. I database main memory ne sono un classico esempio.

- **Database di oggetti**. La struttura dei dati è più simile a un oggetto che a una tabella.

- **Database a grafo**. Consiste in una raccolta di nodi e vertici utilizzata principalmente nelle applicazioni di data mining e social media.

Sqlite non presenta alcuni costrutti SQL quali RIGHT JOIN e FULL OUTER JOIN, la scrittura diretta nelle viste ma presenta molti pregi:

Dimensioni Supporta database di dimensioni terabyte anche molto grandi; attualmente il limite è 140TB con una riga di dimensione massima pari a 1GB;

Velocità E' molto veloce, in molti casi più di MySQL e PostgreSQL. Inoltre SQLite è in alcuni casi più veloce dell'I/O diretto del filesystem.

Documentazione Il codice sorgente è liberamente disponibile, chiaro e ben commentato. Inoltre il team di sviluppo di SQLite prende molto sul serio il test e la verifica del codice.

Serverless SQLite è progettato per essere integrato nelle applicazioni, anzichè utilizzare un programma server di database separato tipo MySQL, PostgreSQL ed Oracle. E' veloce, rigorosamente testato e flessibile, rendendolo adatto per prototipazione e distribuzione in produzione per alcune applicazioni. La libreria SQLite accede direttamente ai suoi file di archiviazione. Le applicazioni interagiscono con il database SQLite in lettura e scrittura direttamente dai file di database memorizzati sul disco.

Zero Configuration La creazione di un'istanza del database SQLite è semplice come l'apertura di un file. Grazie all'architettura senza server, non è necessario "installare" SQLite prima di utilizzarlo. Non esiste alcun processo server che deve essere configurato, avviato e arrestato. Inoltre, SQLite non utilizza alcun file di configurazione.

Cross-Platform L'intera istanza del database risiede in un singolo file multi-piattaforma, che non richiede amministrazione. Android, *BSD, iOS, Linux, Mac, Solaris, VxWorks e Windows (Win32, WinCE, WinRT) sono supportati immediatamente. Facile da trasferire ad altri sistemi.

Autosufficiente Una singola libreria contiene l'intero sistema di database, che si integra direttamente in un'applicazione host. Ciò significa che richiede un supporto minimo dal sistema operativo o dalla libreria esterna e questo rende SQLite utilizzabile in qualsiasi ambiente, specialmente in dispositivi integrati come iPhone, telefoni Android, console di gioco, lettori multimediali portatili, ecc.

Ingombro di runtime ridotto La build predefinita è inferiore a un megabyte di codice e richiede solo pochi megabyte di memoria. Con alcune regolazioni, sia la dimensione della libreria che l'uso della memoria possono essere ulteriormente e significativamente ridotti.

Transactional Tutte le transazioni in SQLite sono completamente conformi ACID. Significa che tutte le query e le modifiche sono Atomic, Coerent, Isolated e Durable. In altre parole, tutte le modifiche all'interno di una transazione avvengono completamente o per nulla anche quando si verifica una situazione imprevista come un arresto anomalo dell'applicazione, un'interruzione dell'alimentazione (blackout) o un arresto anomalo del sistema operativo.

Full-Featured SQLite supporta la maggior parte delle funzionalità del linguaggio di query presenti nello standard SQL92 (SQL2).

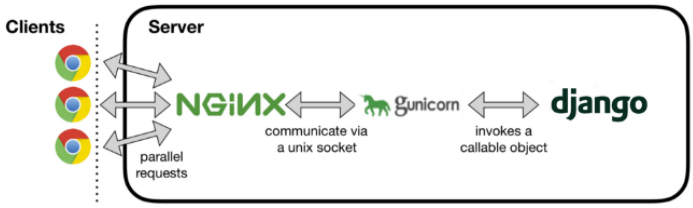
Tipi dinamici Una peculiarità di SQLite è la gestione flessibile dei tipi di dati: utilizza tipi dinamici per le tabelle, ed ogni campo può contenere qualsiasi tipo di dato (o quasi; gestito differentemente nella versione 2 e 3).

Database in memoria SQLite è in grado di creare database in memoria e quindi maggiormente performanti.

2.4 Nginx e Gunicorn

Per il **deployment** dell'applicazione sono stati utilizzati Nginx e Gunicorn, opzioni solide e popolari e che lavorano insieme. Ciascuno può fare qualcosa che l'altro non può e quindi si completano a vicenda. Nginx è il primo punto in cui arrivano le richieste da Internet e le gestisce molto rapidamente filtrandole. Gunicorn traduce le richieste che riceve da Nginx in un formato che l'applicazione web può gestire assicurandosi che venga eseguito quando necessario.

Figura 2.1: flusso di richieste



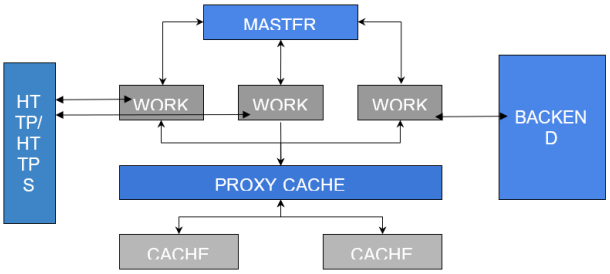
Nginx E' un software open source ideato da Igor Sysoev, che nasce come web server con lo scopo di garantire le massime performance e la massima stabilità tramite una sofisticata architettura con un approccio basato su eventi che gli permette di elaborare le richieste in maniera asincrona, risparmiando memoria locale e tempo. Oggi viene usato anche per operazioni di reverse proxy, caching e load balancing. Tra le aziende che utilizzano questa tecnlogia ci sono Google, Twitter, Facebook, LinkedIn, Apple, Microsoft, Intel, Cisco, Adobe, IBM, Xerox.

NGINX oltre ad essere user friendly, facile da installare e da configurare, è molto performante ad offrire file statici come file zip, immagini e pdf. Offre inoltre anche un servizio di cache in modo da mantenere ancora più elevate le sue prestazioni.

I web server tradizionali, come Apache, presentano una criticità: utilizzano un modello per il quale ogni richiesta viene trattata con un nuovo processo, o thread, e bloccano le operazioni di I/O fino al loro compimento. Tale modello può, in base al tipo di applicazione, risultare inefficiente, sia dal punto di vista del consumo della memoria sia della CPU.

La soluzione proposta da NGINX è un'architettura event-driven, asincrona, single-threaded e non-blocking, che permette di mantenere al minimo l'utilizzo delle risorse senza dover creare un nuovo thread per ogni richiesta garantendo una scalabilità non lineare sia sul numero di connessioni simultanee che per richieste al secondo.

Figura 2.2: Architettura di NGINX



NGINX è stato pensato per essere uno strumento specializzato per raggiungere il massimo delle performance e massimizzare l'economia delle risorse permettendo di gestire migliaia di richieste tramite un numero limitato di processi single-thread.

Si divide in quattro processi principali: Master, Workers, Cache Loader e Cache Manager. Tutti i processi sono single-thread e comunicano tra di loro tramite una memoria condivisa.

1. **Master**. È il processo principale che si occupa di creare e orchestrare tutti gli altri, ed è responsabile della lettura e validazione della configurazione, creazione, binding e chiusura dei socket, gestione dei processi Worker, riconfigurazione senza interruzioni del servizio.

2. **Cache Loader**. È il processo che gestisce il caricamento in memoria della cache disk-based, popolando il database in memoria con metadati e la preparazione dell'istanza di NGINX per lavorare con file storicizzati sul disco in una determinata struttura di directory.

   Quando ha completato il proprio compito, termina la propria esecuzione mantenendo al minimo le quantità di risorse necessarie.

3. **Cache Manager**. Processo che viene eseguito periodicamente, e si occupa di pulire la cache affinché resti all'interno dei limiti di grandezza specificata in configurazione. In caso di fallimento il Master si occupa di riavviarlo.

4. **Workers**. Sono processi che non richiedono l'accesso a elementi privilegiati del sistema (tipicamente uno per ogni core o CPU presenti nel sistema) e che vengono creati all'avvio del Master. Gestiscono le connessioni con i client e le relative richieste.

NGINX è basato su una struttura modulare, questo significa che le diverse funzioni vengono messe a disposizione attraverso diversi moduli attivabili e disattivabili dagli amministratori. Di seguito sono riportate le sue principali caratteristiche:

- Possiede una struttura leggera che gli attribuisce prestazioni elevate, ed è versatile: funziona su Unix, Linux, macOS, Solaris, Windows e fornisce contenuti statici senza gravare sulle risorse del sistema, che possono essere utilizzate per svolgere altre operazioni.

- Supporta il protocollo di Rete IPv6, quest'ultimo ha una struttura di indirizzi diversa da IPv4, ed è lunga 128 bit e scritta con 8 gruppi esadecimali.

- Supporta i protocolli di messaggistica WebSocket. Questo consente di gestire le comunicazioni asincrone e full-duplex (quindi bidirezionali e in contemporanea) su connessione TCP (Transmission Control Protocol), vengono così superati i limiti delle connessioni HTTP con cui le trasmissioni dei dati sono invece half-duplex e quindi unidirezionali.

- Il load balancing (bilanciamento del carico) è una tecnica per distribuire i carichi di traffico su due o più linee di connessione in modo bilanciato. Alleggerisce quindi il server principale, distribuendo le richieste.

- La possibilità di agire come reverse proxy, ciò offre ulteriori garanzie dal punto di vista del bilanciamento dei carichi di lavoro perché una configurazione di questo genere consente di collegare un medesimo URL con più server all'interno di una VPN (Virtual Private Network), permettendo di distribuire le richieste su di essi. Quando agisce come reverse proxy NGinx è in grado di allocare temporaneamente le risposte inviate dai server e di riutilizzarle per non dover elaborare nuovamente le richieste. A ciò si aggiunga il supporto per la compressione dei dati, sia in entrata che in uscita, che determina un ulteriore beneficio per le prestazioni.

- Consente di utilizzare connessioni crittografate per un trasferimento dei dati sicuro, con la possibilità di archiviare i certificati SSL, di filtrare i package veicolati sulla base di direttive definite dall'amministratore di sistema, di installare servizi che operano in background come per esempio gli antivirus e, sempre tramite reverse proxy, di anonimizzare i server di destinazione delle richieste.

Gunicorn. Dal momento che si sta utilizzando Django, che è un framework web, c'è bisogno di qualcosa che lo colleghi con il web server. Nel mondo Python, una cosa del genere si chiama server WSGI (Web Server Gateway Interface), all'interno del quale è presente anche Gunicorn.

Quando gestisce una richiesta, Nginx la inoltra a Gunicorn, che a sua volta chiama il codice Django e restituisce la risposta http.

Gunicorn creerà un socket Unix e fornirà le risposte a nginx tramite il protocollo wsgi, il socket passa i dati in entrambe le direzioni.

3. Descrizione dei requisiti dell'applicazione

In questo capitolo descriveremo i requisiti del nostro sistema. Al fine di garantire una buona organizzazione dei requisiti e di poter codificare le dipendenze, identificheremo ogni requisito mediante un indicatore numerico incrementale. Inoltre, i requisiti **funzionali** e **non funzionali** sono codificati da:

`REQ_tipo_nome`

dove:

- *REQ* indica che si tratta di un requisito
- Il tipo può essere:
  - *F* indica se si tratta di un requisito funzionale
  - *NF* indica se si tratta di un requisito non funzionale
- Il nome serve ad identificare la funzionalità descritta dal requisito

3.1 Requisiti sui dati

1. Di ogni Utente registrato si vuole mantenere:
   - 1.1 Username
   - 1.2 Nome
   - 1.3 Cognome
   - 1.4 Email
   - 1.5 Password (hash)
   - 1.6 Ogni utente può essere (disgiunzione e completezza):
     - 1.6.1 Utente direzionale
     - 1.6.2 Chef, in questo caso è di interesse sapere:
       - 1.6.2.1 L'impianto di appartenenza (cfr. req. 2)
       - 1.6.2.2 Se è lo chef manager dell'impianto
     - 1.6.3 Capoarea, di cui interessa:
       - 1.6.3.1 L'area che gestisce (cfr. req. 10)
2. Di ogni Impianto interessa:
   - 2.1 Il nome
   - 2.2 L'area in cui si trova (cfr. req. 10)
   - 2.3 La classificazione (classico, premium o plus)
   - 2.4 Gli ingredienti disponibili (cfr. req. 6), di ognuno dei quali interessa:

2.4.1 Il costo al kg definito dall'impianto
2.4.2 Il fornitore definito dall'impianto (cfr. req. 8)

3. Di ogni Ricetta interessa:
   3.1 Il nome, non esistono ricette che hanno lo stesso nome
   3.2 La portata (cfr. req. 4)
   3.3 Il tag (cfr. req. 5)
   3.4 Lo chef responsabile (cfr. req. 1). Solo gli utenti direzionali possono non specificarlo.
   3.5 Una foto (opzionale)
   3.6 Sapere se è una ricetta special. Se una ricetta è special può essere visualizzata solo dagli chef di un impianto Plus o Premium.
   3.7 Sapere se è stata approvata dalla direzione
   3.8 L'insieme degli ingredienti (cfr. req. 6), di ognuno interessa sapere:
      3.8.1 La quantità in grammi
4. Di ogni Portata interessa:
   4.1 Il nome (primo piatto, secondo piatto, contorno o dessert)
5. Di ogni Tag interessa:
   5.1 Il nome
   5.2 Il valore di un tag è vincolato dal tipo di portata. I valori ed i vincoli sono i seguenti:
      5.2.1 Se la portata è "Primo piatto": zuppa, primo classico, primo speciale.
      5.2.2 Se la portata è "Secondo piatto": secondo di carne rossa, secondo di carne bianca, secondo di pesce, vegetariano, vegano.
      5.2.3 Se la portata è "Contorno": contorno.
      5.2.4 Se la portata è "Dessert": dessert.
6. Di ogni Ingrediente interessa:
   6.1 Il nome generico (univoco)
   6.2 Il nome specifico
   6.3 Prezzo al kg di riferimento
   6.4 L'insieme degli allergeni contenuti (cfr. req. 9)
   6.5 L'apporto calorico con le info di carboidrati, proteine e grassi
7. Di ogni Menu Giornaliero interessa:
   7.1 L'impianto in cui verrà servito (cfr. req. 2)
   7.2 L'anno in cui verrà servito
   7.3 La settimana dell'anno in cui verrà servito, un intero tra 1 e 52
   7.4 Il giorno della settimana in cui verrà servito (lunedi,martedi,mercoledi,giovedi,venerdi)
   7.5 L'utente creatore (cfr. req. 1)
   7.6 L'istante di creazione

7.7 Se è stato approvato, in questo caso interessa sapere:

7.7.1 L'utente direzionale approvatore (cfr. req. 1)

7.8 Lo storico delle modifiche effettuate, di ognuna interessa sapere:

7.8.1 quando è stata effettuata

7.8.2 quali sono le modifiche effettuate (eliminazione o aggiunta di ricette)

7.8.3 L'utente che l'ha effettuata (cfr. req. 1)

7.9 L'insieme delle ricette ordinate per tag (cfr. req. 3). L'ordinamento è definito dallo staff.

8. Di ogni Fornitore interessa:

8.1 Il nome

9. Di ogni Allergene interessa:

9.1 Il nome

9.2 Foto (opzionale)

10. Di ogni Area interessa:

10.1 Il nome

3.2 Requisiti Funzionali

11. `REQ_F_Autenticazione`

    11.1 `REQ_F_EffettuaLogin`
    - Funzione: Primaria
    - Descrizione: la piattaforma deve permette agli utenti registrati di iniziare la propria sessione autenticata

    11.2 `REQ_F_EffettuaLogout`
    - Funzione: Primaria
    - Descrizione: la piattaforma deve permettere agli utenti autenticati di uscire dalla propria sessione

12. `REQ_F_GestisciRicette`

    12.1 `REQ_F_InserisciRicetta`
    - Funzione: Primaria
    - Descrizione: permette agli utenti di creare una nuova ricetta da far approvare

    12.2 `REQ_F_AggiungiIngredienteRicetta`
    - Funzione: Primaria
    - Descrizione: permette agli utenti (che ne hanno la possibilità) di aggiungere un nuovo ingrediente ad una ricetta

    12.3 `REQ_F_EliminaIngredienteRicetta`
    - Funzione: Primaria
    - Descrizione: permette agli utenti (che ne hanno la possibilità) di eliminare un ingrediente da una ricetta

    12.4 `REQ_F_DuplicaRicetta`
    - Funzione: Primaria
    - Descrizione: permette agli utenti di duplicare una ricetta

    12.5 `REQ_F_InserisciImmagineRicetta`
    - Funzione: Primaria
    - Descrizione: permette agli utenti (che ne hanno la possibilità) di associare un'immagine ad una ricetta

    12.6 `REQ_F_RimuoviImmagineRicetta`
    - Funzione: Primaria
    - Descrizione: permette agli utenti (che ne hanno la possibilità) di eliminare la foto di una ricetta

    12.7 `REQ_F_RicercaGoogleImmagine`
    - Funzione: Primaria
    - Descrizione: permette agli utenti di cercare una foto su google di una ricetta che si è intenzionati a caricare

    12.8 `REQ_F_EliminaRicetta`
    - Funzione: Primaria

- Descrizione: permette agli utenti direzionali,allo chef creatore, allo chef manager e direttore dell'impianto di eliminare una ricetta esistente

12.9 `REQ_F_ApprovaRicetta`

- Funzione: Primaria
- Descrizione: permette agli utenti direzionali di approvare una ricetta in stato di approvazione

12.10 `REQ_F_ImportaRicetteExcel`

- Funzione: Primaria
- Descrizione: permette agli utenti direzionali di caricare ricette tramite un file excel

12.11 `REQ_F_VisualizzaRicetteDaApprovare`

- Funzione: Primaria
- Descrizione: permette agli utenti direzionali di visualizzare la lista delle ricette che devono essere approvate

13. `REQ_F_ConsultaRicette`

13.1 `REQ_F_RicercaPerNomeRicette`

- Funzione: Primaria
- Descrizione: permette agli utenti di consultare le ricette, visualizzando solo quelle a cui hanno accesso

13.2 `REQ_F_RicercaFiltrataRicette`

- Funzione: Primaria
- Descrizione: permette agli utenti di ricercare le ricette filtrando per tag, portata ed impianto.

14. `REQ_F_GestisciMenu`

14.1 `REQ_F_CreaMenuSettimanale`

- Funzione: Primaria
- Descrizione: permette agli utenti di creare un menu settimanale per un certo impianto (con all'interno almeno una ricetta), composto da 5 menù giornalieri

14.2 `REQ_F_VisualizzaMenu`

- Funzione: Primaria
- Descrizione: permette agli utenti che hanno accesso all'impianto di visualizzare un menu settimanale

14.3 `REQ_F_VisualizzaMenuCosti`

- Funzione: Secondaria
- Descrizione: permette agli utenti che hanno accesso all'impianto di visualizzare i costi di un menu settimanale

14.4 `REQ_F_VisualizzaMenuAllergeni`

- Funzione: Secondaria
- Descrizione: permette agli utenti che hanno accesso all'impianto di visualizzare gli allergeni di un menu settimanale

14.5 `REQ_F_VisualizzaStoricoModifiche`

- Funzione: Secondaria
- Descrizione: permette agli utenti che hanno accesso all'impianto di visualizzare lo storico delle modifiche effettuate ad un menù giornaliero

14.6 `REQ_F_AggiungiRicettaMenu`

- Funzione: Primaria
- Descrizione: permette agli utenti che hanno accesso all'impianto e che possono modificare il menu di aggiungerci ricette

14.7 `REQ_F_RimuoviRicettaMenu`

- Funzione: Primaria
- Descrizione: permette agli utenti che hanno accesso all'impianto e che possono modificare il menu di eliminare ricette

14.8 `REQ_F_ApprovaMenu`

- Funzione: Primaria
- Descrizione: permette agli utenti direzionali di approvare un menu creato e che è in attesa di approvazione

14.9 `REQ_F_GeneraOrdine`

- Funzione: Primaria
- Descrizione: permette agli utenti che hanno accesso al menu settimanale di un certo impianto di generare un ordine settimanale degli ingredienti

14.10 `REQ_F_EffettuaAnalisiIngredienti`

- Funzione: Primaria
- Descrizione: permette agli utenti che hanno accesso al menu di un certo impianto di effettuare un'analisi degli ingredienti, confrontando le quattro settimane precedenti

14.11 `REQ_F_EffettuaAnalisiPiatti`

- Funzione: Primaria
- Descrizione: permette agli utenti che hanno accesso al menu di un certo impianto di effettuare un'analisi delle ricette, confrontando le quattro settimane precedenti

14.12 `REQ_F_DownloadPdf`

- Funzione: Secondaria
- Descrizione: permette agli utenti che hanno accesso al menu di un certo impianto di stampare il pdf riassumente la lista delle ricette giornaliere

14.13 `REQ_F_DownloadRicettario`

- Funzione: Secondaria
- Descrizione: permette agli utenti che hanno accesso al menu di un certo impianto di stampare il ricettario riassumente gli ingredienti di ogni ricetta nel menu della settimana

14.14 `REQ_F_ImportaMenu`

- Funzione: Secondaria
- Descrizione: permette agli utenti che hanno accesso all'impianto di importare menu da altri impianti esistenti

15. `REQ_F_GestisciImpianti`

    15.1 `REQ_F_InserisciImpianto`

    - Funzione: Primaria
    - Descrizione: permette agli utenti direzionali di aggiungere nuovi impianti nel sistema

    15.2 `REQ_F_EliminaImpianto`

    - Funzione: Primaria
    - Descrizione: permette agli utenti direzionali di eliminare impianti esistenti

    15.3 `REQ_F_VisualizzaListinoImpianto`

    - Funzione: Primaria
    - Descrizione: permette agli utenti direzionali di visualizzare il listino dei prezzi di ogni impianto

    15.4 `REQ_F_DownloadListinoImpianto`

    - Funzione: Secondaria
    - Descrizione: permette agli utenti direzionali di scaricare il file excel con i listini dei prezzi di ogni impianto

    15.5 `REQ_F_CaricaFIleExcelListinoImpianto`

    - Funzione: Secondaria
    - Descrizione: permette agli utenti direzionali di caricare il file excel con i listini degli ingredienti

    15.6 `REQ_F_VisualizzaSettimaneImpianto`

    - Funzione: Primaria
    - Descrizione: permette agli utenti che hanno accesso all'impianto di visualizzare le 52 settimane di ogni anno

    15.7 `REQ_F_AssegnaClassificazioneImpianto`

    - Funzione: Primaria
    - Descrizione: permette agli utenti direzionali di associare ad un certo impianto una classificazione

16. `REQ_F_GestisciUtenti`

    16.1 `REQ_F_RicercaUtenti`

    - Funzione: Primaria
    - Descrizione: permette agli utenti direzionali di ricercare utenti registrati nel sistema

    16.2 `REQ_F_CreaUtente`

    - Funzione: Primaria
    - Descrizione: permette agli utenti direzionali di registrare utenti che avranno poi la possibilità di accedere il sistema

    16.3 `REQ_F_EliminaUtente`

    - Funzione: Primaria

- Descrizione: permette agli utenti direzionali di eliminare utenti non più attivi nel sistema

16.4 `REQ_F_ModificaDatiAccessoUtente`

- Funzione: Secondaria
- Descrizione: permette agli utenti direzionali di modificare i dati di accesso (password) di un utente registrato

17. `REQ_F_GestisciIngredienti`

17.1 `REQ_F_InserisciIngrediente`

- Funzione: Primaria
- Descrizione: permette agli utenti direzionali di inserire un nuovo ingrediente nel sistema

17.2 `REQ_F_RimuoviIngrediente`

- Funzione: Primaria
- Descrizione: permette agli utenti direzionali di rimuovere un ingrediente dal sistema

18. `REQ_F_GestisciFornitori`

18.1 `REQ_F_AggiungiFornitore`

- Funzione: Primaria
- Descrizione: permette agli utenti direzionali di aggiungere un nuovo fornitore al sistema

18.2 `REQ_F_EliminaFornitore`

- Funzione: Primaria
- Descrizione: permette agli utenti direzionali di eliminare un fornitore dal sistema

19. `REQ_F_GestisciAree`

19.1 `REQ_F_InserisciArea`

- Funzione: Primaria
- Descrizione: permette agli utenti direzionali di aggiungere una nuova area al sistema

19.2 `REQ_F_EliminaArea`

- Funzione: Primaria
- Descrizione: permette agli utenti direzionali di eliminare un'area esistente nel sistema

19.3 `REQ_F_NominaCapoArea`

- Funzione: Secondaria
- Descrizione: permette agli utenti direzionali di nominare un utente capoarea all'interno di una certa area

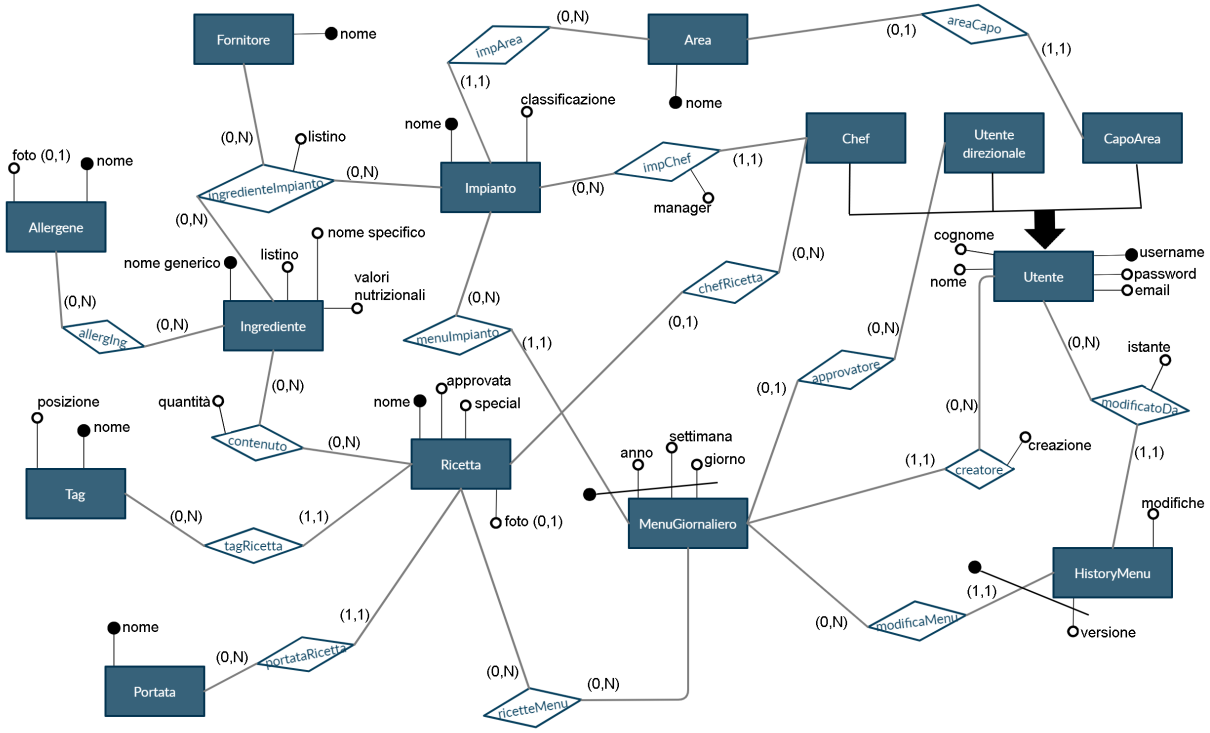
3.3 Requisiti Non Funzionali

20. `REQ_NF_ProtezioneDati`
    - Funzione: Primaria
    - Descrizione: la piattaforma deve garantire la riservatezza dei dati sensibili inseriti dagli utenti
21. `REQ_NF_UsabilitàInterfaccia`
    - Funzione: Primaria
    - Descrizione: la piattaforma deve avere un'interfaccia intuitiva e semplice per permettere agli utenti di utilizzare tutti i servizi a loro disposizione
22. `REQ_NF_SemplicitàRicercaContenuti`
    - Funzione: Primaria
    - Descrizione: la piattaforma deve disporre di metodi di ricerca intuitivi per permette di effettuare ricerche e query in tempo affidabile
23. `REQ_NF_SitoResponsive`
    - Funzione: Primaria
    - Descrizione: la piattaforma deve essere responsive in modo tale da poter essere consultata agevolmente da qualsiasi dispositivo
24. `REQ_NF_DisponibilitàPiattaforma`
    - Funzione: Primaria
    - Descrizione: la piattaforma deve soddisfare le richieste degli utenti in tempi ragionevoli (pochi secondi al massimo), e nel caso in cui questo non fosse momentaneamente possibile, è necessario comunicarlo all'utente precisando l'errore
25. `REQ_NF_CompatibilitàInterfaccia`
    - Funzione: Primaria
    - Descrizione: la piattaforma deve essere compatibile con i principali browser presenti nel mercato (Edge, Chrome, Firefox, Opera, Safari)
26. `REQ_NF_AffidabilitàDatabase`
    - Funzione: Primaria
    - Descrizione: la piattaforma deve offrire un alto livello di integrità del database per prevenire perdite di dati importanti

4. Progettazione della soluzione

4.1 Diagramma ER e specifiche dei dati

Nella figura che segue è rappresentato il diagramma ER che modella l'applicazione iMenuManager. Nel seguito forniamo una descrizione dettagliata delle entità, delle relationship e dei dati.



Entità Tag
Ogni istanza di questa entità rappresenta un tag

attributo	dominio	molteplicità	descrizione
nome	stringa		Il nome del tag
posizione	intero >0		La posizione del tag nei menu

Entità Allergene
Ogni istanza di questa entità rappresenta un allergene

attributo	dominio	molteplicità	descrizione
nome	stringa		Il nome dell'allergene
foto	image	(0,1)	La foto dell'allergene

Entità Ingrediente
Ogni istanza di questa entità rappresenta un ingrediente

attributo	dominio	molteplicità	descrizione
nome specifico	stringa		Il nome specifico dell'ingrediente
nome generico	stringa		il nome generico dell'ingrediente
listino	reale ≥ 0		il costo di riferimento dell'ingrediente
energiaKcal	reale ≥ 0		energia in kcal dell'ingrediente
energiaKj	reale ≥ 0		energia in kj dell'ingrediente
acqua	reale ≥ 0		acqua nell'ingrediente
proteine animali	reale ≥ 0		proteine animali nell'ingrediente
proteiene vegetali	reale ≥ 0		proteine vegetali nell'ingrediente
amido	reale ≥ 0		amido nell'ingrediente
glucidi solubili	reale ≥ 0		glucidi solubili nell'ingrediente
grassi saturi	reale ≥ 0		grassi saturi nell'ingrediente
grassi monoinsaturi	reale ≥ 0		grassi monoinsaturi nell'ingrediente
grassi polinsaturi	reale ≥ 0		grassi polinsaturi nell'ingrediente
acido oleico	reale ≥ 0		acido oleico nell'ingrediente
acido linoleico	reale ≥ 0		acido linoleico nell'ingrediente
acido linolenico	reale ≥ 0		acido linolenico nell'ingrediente
altri polinsaturi	reale ≥ 0		altri polinsaturi nell'ingrediente
colesterolo	reale ≥ 0		colesterolo nell'ingrediente
fibre animali	reale ≥ 0		fibre animali nell'ingrediente
alcool	reale ≥ 0		alcool nell'ingrediente
ferro (fe)	reale ≥ 0		ferro nell'ingrediente
calcio (ca)	reale ≥ 0		calcio nell'ingrediente
sodio (na)	reale ≥ 0		sodio nell'ingrediente
potassio (k)	reale ≥ 0		potassio nell'ingrediente
fosforo (p)	reale ≥ 0		fosforo nell'ingrediente
zinco (zn)	reale ≥ 0		zinco nell'ingrediente
vitamina b1	reale ≥ 0		vitamina b1 nell'ingrediente
vitamina b2	reale ≥ 0		vitamina b2 nell'ingrediente
vitamina b3	reale ≥ 0		vitamina b3 nell'ingrediente
vitamina c	reale ≥ 0		vitamina c nell'ingrediente
vitamina b6	reale ≥ 0		vitamina b6 nell'ingrediente
folico	reale ≥ 0		folico nell'ingrediente
retinolo	reale ≥ 0		retinolo nell'ingrediente
beta carotene	reale ≥ 0		beta carotene nell'ingrediente
vitamina e	reale ≥ 0		vitamina e nell'ingrediente
vitamina d	reale ≥ 0		vitamina d nell'ingrediente
parte edibile	reale ≥ 0		parte edibile nell'ingrediente

Entità Portata
Ogni istanza di questa entità rappresenta una portata

attributo	dominio	molteplicità	descrizione
nome	stringa		il nome della portata

Entità Area
Ogni istanza di questa entità rappresenta un'area

attributo	dominio	molteplicità	descrizione
nome	stringa		il nome dell'area

Entità Impianto
Ogni istanza di questa entità rappresenta un impianto

attributo	dominio	molteplicità	descrizione
nome	stringa		il nome dell'impianto
classificazione	{Classico, Plus, Premium}		la classificazione dell'impianto

Entità Ricetta
Ogni istanza di questa entità rappresenta una ricetta

attributo	dominio	molteplicità	descrizione
nome	stringa		il nome della ricetta
approvata	booleano		indica se la ricetta è stata approvata dalla direzione
foto	image	(0,1)	la foto della ricetta
special	booleano		indica se la ricetta è special

Entità MenuGiornaliero
Ogni istanza di questa entità rappresenta un menu giornaliero

attributo	dominio	molteplicità	descrizione
anno	intero ≥ 2021		l'anno in cui verrà servito
settimana	[1,52]		la settimana in cui verrà servito
giorno	{Lunedi,Martedi,Mercoledi,Giovedi,Venerdi}		il giorno in cui verrà servito

Entità HistoryMenu
Ogni istanza di questa entità rappresenta lo storico di un menu

attributo	dominio	molteplicità	descrizione
versione	intero ≥ 1		il numero della modifica
modifiche	stringa		testo delle modifiche effettuate

Entità Fornitore
Ogni istanza di questa entità rappresenta un fornitore

attributo	dominio	molteplicità	descrizione
nome	stringa		il nome del fornitore

Entità Utente
Ogni istanza di questa entità rappresenta un utente del sistema

attributo	dominio	molteplicità	descrizione
username	stringa		l'username dell'utente
password	stringa criptata		la password dell'utente
email	stringa		l'email dell'utente
nome	stringa		il nome dell'utente
cognome	stringa		il cognome dell'utente

Relationship impChef
Ogni istanza di questa relationship lega uno chef al suo impianto

attributo	dominio	molteplicità	descrizione
manager	booleano		indica se lo chef è il manager dell'impianto

Relationship creatore
Ogni istanza di questa relationship lega un menu al suo utente creatore

attributo	dominio	molteplicità	descrizione
creazione	dataora		l'istante di creazione del menu

Relationship modificatoDa
Ogni istanza di questa relationship lega una modifica di un menu all'utente che l'ha effettuata

attributo	dominio	molteplicità	descrizione
istante	dataora		l'istante di modifica di un menu

Relationship contenuto
Ogni istanza di questa relationship lega una ricetta ad un ingrediente

attributo	dominio	molteplicità	descrizione
quantità	intero ≥ 1		la quantità dell'ingrediente nella ricetta

Relationship ingredienteImpianto
Ogni istanza di questa relationship lega un ingrediente ad un certo impianto e ad un fornitore

attributo	dominio	molteplicità	descrizione
listino	intero ≥ 0		il listino dell'impianto per l'ingrediente associato al fornitore

4.2 Modello degli Use-Case

In questa sezione identifichiamo i casi d'uso presenti nel progetto e gli attori coinvolti nel loro utilizzo. Lo scopo è quello di fornire una descrizione dettagliata delle modalità di interazione tra le diverse entità presenti ed il sistema. L'analisi dei casi d'uso viene presentata tramite grafici che seguono lo standard UML.

Ogni attore del sistema è descritto dai seguenti campi:

- ID: identificativo univoco dell'attore.
- Nome: nome dell'attore, univoco.
- Genitore: identificativo dell'attore genitore.
- Descrizione: breve descrizione della posizione assunta dall'attore.

In questo sistema tutti gli attori possiedono funzionalità simili, pertanto gli attori che servono per rappresentare le funzioni use-case sono i seguenti:

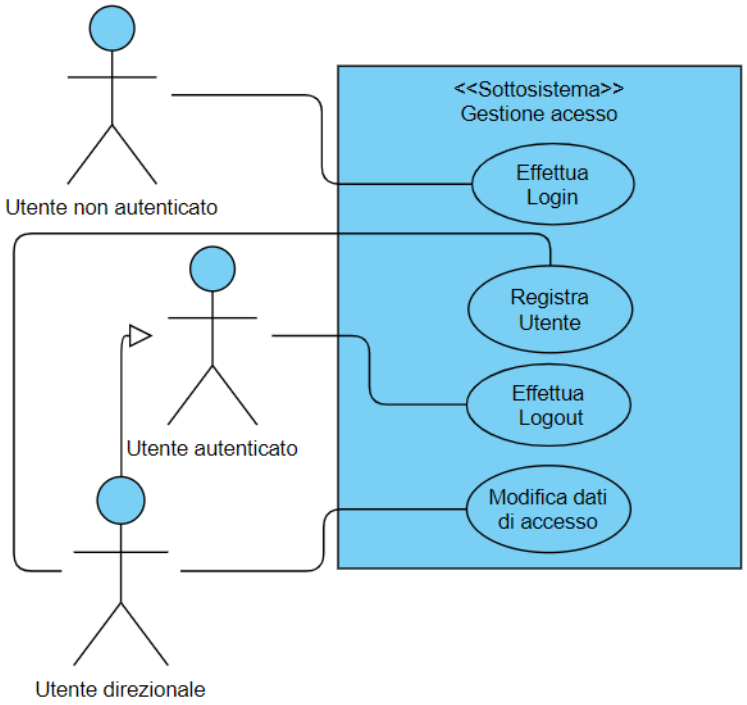
Id	01
Nome	Utente non autenticato
Genitore	-
Descrizione	Rappresente un utente che deve effettuare l'autenticazione nel sistema.

Id	02
Nome	Utente autenticato
Genitore	-
Descrizione	Rappresenta un utente che ha effettuato l'autenticazione e che accede alle funzionalità in base al suo ruolo. Può essere uno chef, un capoarea o un utente direzionale.

Id	03
Nome	Utente direzionale
Genitore	02
Descrizione	Rappresenta un utente amministratore del sistema, che possiede molteplici benefici rispetto agli altri utenti.

Nel seguito vengono descritti i **casi d'uso principali** del sistema, suddivisi per sottosistema, ognuno è indicato con *SUB.Nome* dove *SUB* indica il sottosistema di appartenenza del caso d'uso e *Nome* è il nome rappresentativo del caso.

1. **Gestione accesso (GA)**



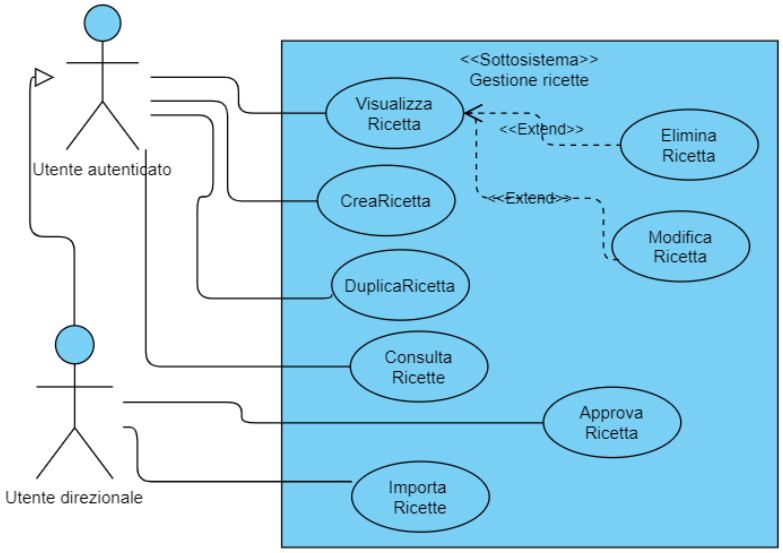
GA.EffettuaLogin	
Attore	Utente non autenticato
Precondizioni	L'utente non è ancora autenticato nel sistema
Flusso principale	1. L'utente visualizza la schermata di login 2. Il sistema richiede all'utente le credenziali per l'accesso 3. L'utente inserisce le sue credenziali e conferma 4. Il sistema verifica se le credenziali inserite corrispondono ad un account
Postcondizioni	Il sistema contiene una sessione attiva dell'utente

GA.RegistraUtente	
Attore	Utente direzionale
Precondizioni	-
Flusso principale	1. L'utente visualizza la schermata per la registrazione di nuovi utenti 2. L'utente inserisce i dati richiesti negli appositi campi 3. L'utente conferma i dati inseriti 4. Qualora non esista un utente con quelle credenziali allora ne viene creato uno nuovo 5. Se già esiste viene lanciato un errore
Postcondizioni	Il sistema contiene un nuovo utente registrato con i dati inseriti

GA.EffettuaLogout	
Attore	Utente autenticato
Precondizioni	Esiste una sessione attiva dell'utente
Flusso principale	1. L'utente richiede di effettuare il logout dal sistema 2. Il sistema effettua il logout dell'utente
Postcondizioni	La persona connessa al sito non è più autenticata

GA.ModificaDatiDiAccesso	
Attore	Utente direzionale
Precondizioni	-
Flusso principale	1. L'utente richiede di effettuare le modifiche delle credenziali di un certo utente registrato nel sistema 2. Il sistema permette all'utente di inserire i dati modificabili 3. L'utente compila e conferma i dati inseriti 4. Il sistema comunica all'utente che l'operazione è stata effettuata con successo
Postcondizioni	Il sistema contiene i dati aggiornati per l'accesso dell'utente

2. **Gestione ricette (GR)**



GR.VisualizzaRicetta	
Attore	Utente autenticato
Precondizioni	Il sistema contiene una sessione attiva dell'utente. L'utente può ricercare una ricetta solo se questa è approvata o se lo chef proprietario appartiene ad un suo stesso impianto
Flusso principale	1. L'utente richiede di visualizzare le info di una certa ricetta 2. Il sistema mostra la ricetta all'utente
Postcondizioni	Il sistema sta mostrando all'utente le info della ricetta desiderata

GR.CreaRicetta	
Attore	Utente autenticato
Precondizioni	Il sistema contiene una sessione attiva dell'utente. Non esiste una ricetta con lo stesso nome.
Flusso principale	1. L'utente richiede di creare una nuova ricetta con i dati selezinati 2. Il sistema verifica che non esiste una ricetta con lo stesso nome
Postcondizioni	Il sistema contiene la nuova ricetta

GR.EliminaRicetta	
Attore	Utente autenticato
Precondizioni	Il sistema contiene una sessione attiva dell'utente. L'utente può visualizzare e modificare la ricetta.
Flusso principale	1. L'utente richiede di eliminare la ricetta 2. Il sistema mostra la schermata di eliminazione della ricetta
Postcondizioni	Nel sistema viene eliminata la ricetta in questione

GR.ModificaRicetta	
Attore	Utente autenticato
Precondizioni	Il sistema contiene una sessione attiva dell'utente. La ricetta ancora non è stata approvata e l'utente è lo chef proprietario della ricetta o lo chef manager dell'impianto. Oppure l'utente è un utente direzionale.
Flusso principale	1. L'utente accede al pannello di modifica della ricetta in questione 2. L'utente compila i campi modificabili della ricetta 3. L'utente salva le modifiche effettuate 4. Il sistema comunica all'utente che le modifiche sono avvenute con successo
Postcondizioni	Nel sistema vengono effettuate le modifiche alla ricetta in questione

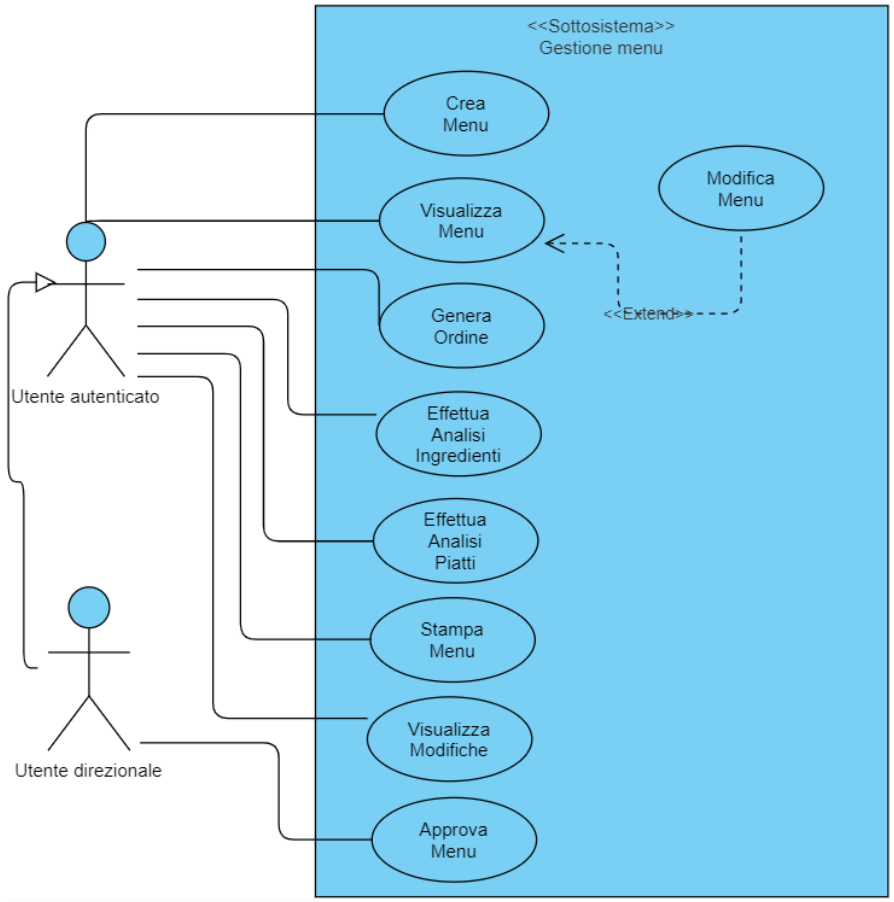
GR.DuplicaRicetta	
Attore	Utente autenticato
Precondizioni	Il sistema contiene una sessione attiva dell'utente. L'utente può visualizzare la ricetta.
Flusso principale	1. L'utente richiede di duplicare la ricetta 2. Il sistema mostra all'utente la schermata di duplicazione della ricetta 3. L'utente deve inserire i dati della nuova ricetta. La ricetta duplicata deve possedere un nome differente. 4. Il sistema comunica all'utente che l'operazione è stata effettuata.
Postcondizioni	Nel sistema viene aggiunta la nuova ricetta

GR.ApprovaRicetta	
Attore	Utente direzionale
Precondizioni	Il sistema contiene una sessione attiva dell'utente. La ricetta risulta ancora non approvata.
Flusso principale	1. L'utente accede alla pagina della ricetta in questione 2. L'utente approva la ricetta
Postcondizioni	La ricetta in questione viene approvata e sarà quindi visibile da tutti nel sistema

GR.ConsultaRicette	
Attore	Utente autenticato
Precondizioni	Il sistema contiene una sessione attiva dell'utente.
Flusso principale	1. L'utente accede alla pagina per consultare le ricette 2. L'utente può ricercare una ricetta solo se questa è approvata o se lo chef proprietario appartiene ad un suo stesso impianto 3. L'utente può opzionalmente inserire i valori del tag, portata e impianto dello chef proprietario per filtrare le ricerche
Postcondizioni	Il sistema mostra l'utente le ricette desiderate

GR.ImportaRicette	
Attore	Utente direzionale
Precondizioni	Il sistema contiene una sessione attiva dell'utente.
Flusso principale	1. L'utente accede alla pagina per importare le ricette 2. L'utente carica il file Excel che ha compilato in precedenza. Il file deve contenere colonne con il nome, il tag, la portata, e gli ingredienti 3. L'utente può opzionalmente definire una colonna per assegnare la ricetta ad uno chef 4. Il sistema comunica all'utente se le ricette sono state caricate con successo
Postcondizioni	Nel sistema vengono create le ricette che l'utente ha caricato

3. **Gestione menu (GM)**



GM.CreaMenu	
Attore	Utente autenticato
Precondizioni	Il sistema contiene una sessione attiva dell'utente.
Flusso principale	1. L'utente accede nella pagina di un impianto a cui ha accesso 2. Il sistema mostra all'utente le 52 settimane dell'anno selezionato 3. L'utente seleziona una settimana tra quelle segnalate disponibili 4. L'utente seleziona per ogni giorno all'interno della settimana selezionata, i tag delle ricette che vuole inserire 5. L'utente seleziona una ricetta, tra quelle che può visualizzare, per ogni tag che ha selezionato 6. L'utente conferma l'inserimento e salva il nuovo menu
Postcondizioni	Nel sistema vengono creati i menu giornalieri della settimana selezionata

GM.VisualizzaMenu	
Attore	Utente autenticato
Precondizioni	Il sistema contiene una sessione attiva dell'utente.
Flusso principale	1. L'utente accede alla pagina di un impianto a cui ha accesso 2. Il sistema mostra all'utente le 52 settimane dell'anno selezionato 3. L'utente seleziona una settimana in cui esiste già il menu, che può essere approvato oppure in attesa di approvazione 4. Il sistema mostra all'utente i menu giornalieri all'interno della settimana selezionata
Postcondizioni	Il sistema sta mostrando all'utente i menu dei giorni che si trovano all'interno della settimana selezionata

GM.ModificaMenu	
Attore	Utente direzionale
Precondizioni	Il sistema contiene una sessione attiva dell'utente. Il menu ancora non è stato approvato e l'utente è lo chef manager dell'impianto o lo chef creatore del menu. Oppure l'utente è un utente direzionale.
Flusso principale	1. L'utente chiede di accedere al pannello di modifica del menu 2. L'utente può: 2.1. Eliminare le ricette nei menu all'interno della settimana 2.2. Aggiungere nuove ricette nei menu all'interno della settimana 3. Il sistema comunica all'utente che le modifiche sono state effettuate con successo
Postcondizioni	Nel sistema vengono modificati i menu giornalieri che sono stati modificati, e se ne tiene traccia

GM.GeneraOrdine	
Attore	Utente autenticato
Precondizioni	Il sistema contiene una sessione attiva dell'utente. L'utente ha accesso all'impianto. Nella settimana selezionata esiste almeno un menu.
Flusso principale	1. L'utente seleziona i giorni per i quali vuole generare un ordine, almeno uno 2. L'utente seleziona le porzioni per ogni singola ricetta che trova all'interno dei giorni selezionati 3. L'utente può eventualmente scegliere se generare l'ordine degli ingredienti per uno specifico fornitore 4.Il sistema crea un PDF dell'ordine e lo visualizza all'utente
Postcondizioni	Il sistema contiene un PDF dell'ordine generato

GM.EffettuaAnalisiIngredienti	
Attore	Utente autenticato
Precondizioni	Il sistema contiene una sessione attiva dell'utente. L'utente ha accesso all'impianto. La settimana desiderata possiede almeno un giorno con un menu.
Flusso principale	1. L'utente seleziona le portate ed i tag delle ricette che vuole filtrare e la natura degli ingredienti che vuole visualizzare 2. Il sistema ricava i menu delle 4 settimane precedenti e da ognuno di questi estrae le ricette con la portata ed il tag tra quelli selezionati. 3. Il sistema mostra all'utente una tabella con gli ingredienti previsti all'interno delle ricette selezionate, associando la sua frequenza per ognuna delle nature che l'utente aveva selezionato
Postcondizioni	Il sistema sta mostrando una tabella all'utente

GM.EffettuaAnalisiPiatti	
Attore	Utente autenticato
Precondizioni	Il sistema contiene una sessione attiva dell'utente. L'utente ha accesso all'impianto. La settimana desiderata possiede almeno un giorno con un menu.
Flusso principale	1. L'utente seleziona le portate, i tag delle ricette che vuole filtrare 2. Il sistema ricava i menu delle 4 settimane precedenti e da ognuno estrae le ricette con la portata ed il tag tra quelli selezionati. 3. Il sistema mostra all'utente una tabella con all'interno ogni ricetta, tra quelle estratte, associandoci la sua frequenza
Postcondizioni	Il sistema sta mostrando una tabella all'utente

GM.StampaMenu	
Attore	Utente autenticato
Precondizioni	Il sistema contiene una sessione attiva dell'utente. L'utente ha accesso all'impianto. La settimana desiderata possiede almeno un giorno con un menu.
Flusso principale	1. L'utente chiede di stampare il menu della settimana 2. Il sistema crea un PDF del menu della settimana
Postcondizioni	Il sistema contiene un PDF del menu scelto

GM.VisualizzaModifiche	
Attore	Utente autenticato
Precondizioni	Il sistema contiene una sessione attiva dell'utente. L'utente ha accesso all'impianto. La settimana desiderata possiede almeno un giorno con un menu.
Flusso principale	1. L'utente chiede di visualizzare lo storico delle modifiche per un certo menu giornaliero all'interno della settimana 2. Il sistema mostra all'utente lo storico delle modifiche, ordinate, del giorno selezionato
Postcondizioni	Il sistema sta mostrando all'utente lo storico delle modifiche del giorno selezionato all'utente

GM.ApprovaMenu	
Attore	Utente direzionale
Precondizioni	Il sistema contiene una sessione attiva dell'utente. La settimana desiderata possiede almeno un giorno con un menu. Il menu non è già approvato.
Flusso principale	1. L'utente chiede al sistema di approvare il menu della settimana selezionata 2. Il sistema comunica all'utente che l'azione è stata effettuata
Postcondizioni	Nel sistema vengono approvati tutti i menu giornalieri che fanno parte della settimana in questione

4.3 Schema del DB

Nel framework Django esiste il concetto di *Model*: si tratta di oggetti che definiscono la struttura dati che andrà memorizzata nel database con i loro campi. Tutti i modelli devono essere sottoclassi della classe Model, lo schema del DB è definito nelle seguenti righe di **codice python**:

```
class Tag(models.Model):
        nome = models.CharField(max_length=100, primary_key=True)
        posizione = models.PositiveIntegerField(validators= [MinValueValidator(1)])

        class Meta:
                ordering = ('posizione',)

class Allergene(models.Model):
        nome = models.CharField(max_length=100, primary_key=True)
        foto = models.ImageField(null=True, blank=True, upload_to="allergeni/")

        class Meta: ordering = ('nome',)

class Ingrediente(models.Model):
        nome_specifico = models.CharField(max_length=100)
        nome_generico = models.CharField(max_length=100, primary_key=True)
        listino = models.FloatField(validators=[MinValueValidator(0.0)])
        allergeni = models.ManyToManyField(Allergene, blank=True)
        energiaKcal = models.FloatField(default=0, validators=[MinValueValidator(0.0)])
        ...

        class Meta:
            ordering = ('nome_generico','nome_generico')

class Portata(models.Model):
        nome = models.CharField(max_length=100, primary_key=True)
        posizione = models.PositiveIntegerField(validators= [MinValueValidator(1)],
        blank=True, null=True)

        class Meta:
                ordering = ('posizione',)

class CapoArea(models.Model):
        user = models.ForeignKey(User, on_delete=models.CASCADE)

class Area(models.Model):
        nome = models.CharField(max_length=100, primary_key=True)
        capo = models.ForeignKey(CapoArea, on_delete=models.SET_NULL, blank=True, null=True)

        class Meta:
                ordering = ('nome','nome')
```

```
42 class Impianto(models.Model):
43         CLASSIFICAZIONI = (("Classico", "Classico"), ("Plus", "Plus"),
44             ("Premium", "Premium") )
45         nome = models.CharField(max_length=100, primary_key=True)
46         area = models.ForeignKey(Area, on_delete=models.CASCADE)
47         classificazione = models.CharField(max_length=8, choices=CLASSIFICAZIONI)
48 
49         class Meta:
50                 ordering = ('area','nome')
51 
52         def get_absolute_url(self):
53                 my_date = datetime.date.today()
54                 annoOggi = my_date.isocalendar()[0]
55                 return reverse('impianto',args=[annoOggi,self.nome])
56 
57 class Chef(models.Model):
58         impianto = models.ForeignKey(Impianto, on_delete=models.CASCADE)
59         user = models.OneToOneField(User,on_delete=models.CASCADE, related_name="user_id")
60         manager = models.BooleanField(default=False)
61 
62         def clean(self):
63                 try:
64                         manager_attuale = Chef.objects.get(impianto=self.impianto,
65                         manager=True)
66                 except:
67                         #non esiste nessun manager dell'impianto
68                         manager_attuale = False
69                 if self.manager and manager_attuale:
70                         raise ValidationError("L'impianto possiede già un manager')
71 
72 class Ricetta(models.Model):
73         nome = models.CharField(max_length=100, primary_key=True)
74         ingredienti = models.ManyToManyField(Ingrediente, through='Contenuto')
75         tag = models.ForeignKey(Tag, on_delete=models.SET_NULL, blank=True, null=True)
76         portata = models.ForeignKey(Portata, on_delete=models.SET_NULL, blank=True,
77         null=True)
78         chef = models.ForeignKey(Chef, on_delete=models.CASCADE, blank=True, null=True)
79         approvata = models.BooleanField(default=False)
80         foto = models.ImageField(null=True, blank=True, upload_to="ricette/")
81         special = models.BooleanField(default=False)
82 
83         class Meta:
84                 ordering = ('nome',)
85 
86         def get_absolute_url(self): return reverse('app:pivot',args=[self.nome, ])
87 
88         #cancello anche l'eventuale foto dalla cartella media
89         def delete(self, using=None, keep_parents=False):
90                 if self.foto:
91                         self.foto.storage.delete(self.foto.name)
92                 super().delete()
```

```
93 class Contenuto(models.Model):
94         ingrediente = models.ForeignKey(Ingrediente, on_delete=models.CASCADE)
95         ricetta = models.ForeignKey(Ricetta, on_delete=models.CASCADE)
96         quantita = models.FloatField(validators=[MinValueValidator(1.0)])
97 
98         class Meta: unique_together = ('ingrediente', 'ricetta',)
99 
100 def current_year(): return datetime.date.today().year
101 
102 def max_value_current_year(value): return MaxValueValidator(current_year())(value)
103 
104 class MenuSettimana(models.Model):
105         GIORNI = (("Lunedi", "Lunedi"), ("Martedi", "Martedi"), ("Mercoledi", "Mercoledi"),
106                 ("Giovedi", "Giovedi"), ("Venerdi","Venerdi"))
107         impianto = models.ForeignKey(Impianto, on_delete=models.CASCADE)
108         anno = models.PositiveIntegerField(default=current_year(),
109         validators=[MinValueValidator(2021), max_value_current_year])
110         settimana = models.CharField(max_length=12,
111                 choices=((str(x), "Settimana "+str(x)) for x in range(1,53)))
112         giorno = models.CharField(max_length=9, choices=GIORNI)
113         ricette = models.ManyToManyField(Ricetta)
114         creatore = models.ForeignKey(User, on_delete=models.SET_NULL,
115             related_name="creatore", blank=True, null=True)
116         creazione=models.DateTimeField(default=timezone.now, editable=False)
117         approvatore = models.ForeignKey(User, on_delete=models.SET_NULL, blank=True, null=True
118                 limit_choices_to={'is_staff': True},  related_name='approvatore')
119 
120         class Meta: unique_together = ('anno','impianto', 'giorno', 'settimana',)
121 
122 class HistoryMenu(models.Model):
123         version = models.IntegerField(editable=False)
124         menu = models.ForeignKey(MenuSettimana, on_delete=models.CASCADE)
125         istante = models.DateTimeField(default=timezone.now, editable=False)
126         text = models.TextField()
127         user = models.ForeignKey(User, on_delete=models.CASCADE)
128 
129         class Meta: unique_together = ('version','menu',)
130 
131         def save(self, *args, **kwargs):
132                 ''' Ricavo quale sarà il nuovo numero di versione'''
133                 attuale = HistoryMenu.objects.filter(menu=self.menu).order_by('-version')[:1]
134                 self.version = attuale[0].version + 1 if attuale else 1
135                 return super(HistoryMenu, self).save(*args, **kwargs)
136 
137 class Fornitore(models.Model):
138         nome = models.CharField(max_length=250, primary_key=True)
139 
140         class Meta: ordering = ('nome',)
141 
142 class IngredienteImpianto(models.Model):
143         ingrediente = models.ForeignKey(Ingrediente, on_delete=models.CASCADE)
144         impianto = models.ForeignKey(Impianto, on_delete=models.CASCADE)
145         listino = models.FloatField(validators=[MinValueValidator(0.0)])
146         fornitore = models.ForeignKey(Fornitore, on_delete=models.SET_NULL, blank=True, null=
147 
148         class Meta:
149                 unique_together = ('ingrediente','impianto',)
150                 ordering = ('ingrediente__nome_generico',)
```

5. Implementazione, Caso di studio e test

5.1 Panoramica su come è stato implementato

Per quanto riguarda l'implementazione è stato utilizzato il linguaggio di programmazione **Python**.

Definizione degli urls. Inizialmente è stato importante definire gli url del sistema, ovvero tutti i path che servono agli utenti per accedere alle pagine del sito. Django consente di progettare gli URL, nel file *urls.py*, in maniera totalmente personalizzata e in particolare è possibile utilizzare valori generici utilizzando le parentesi angolari, ad esempio, con ⟨str:nome⟩ è possibile acquisire un parametro di tipo stringa.

Questo file *urls.py* consiste in una mappatura tra le espressioni del percorso URL alle funzioni Python (chiamate views, presenti nel file *views.py*).

Quando un utente effettua una richiesta per una pagina, Django prende il controllo della vista corrispondente tramite questo file, quindi restituisce la risposta HTML o un errore 404 se non ha trovato una corrispondenza.

Seguono due esempi di path, tra le più di 30 complessive, che sono state utilizzate per il sistema:

```
urlpatterns = [ ...
            path('consulta/<str:pk>/', views.pivot, name="pivot"),
            path('menu/<int:year>/<str:pk>/settimana<int:sett>', views.visualizzaSettimana,
            name="settimane"),
                ... ]
```

Il primo path è quello a cui gli utenti si collegano quando vogliono visualizzare una ricetta che ha un certo nome (una stringa pk).

Il secondo path è quello che serve per accedere al menu di un certo anno (un intero year), di un certo impianto (con pk che ne rappresenta il suo nome) e di una certa settimana (un intero sett).

Creazione delle views. Ogni path che è stato definito è collegato ad una particolare view, ed ognuna di queste deve essere implementata.

Una view è una funzione Python che prende in input una *richiesta Web* (se il path prevede dei parametri generici devono essere anch'essi inseriti in input) e restituisce una risposta Web. Questa risposta può essere il contenuto HTML di una pagina Web, un reindirizzamento, un errore 404, un documento XML, un'immagine o un pdf. La view stessa contiene la logica arbitraria che è necessaria per restituire la risposta desiderata, ad esempio manipolando i dati presenti nel database. La convenzione è di mettere le viste in un file chiamato *views.py*, posizionato nella directory del progetto dell'applicazione. All'interno sono state create anche delle funzioni ausiliarie utili per rendere il codice più modulare.

Creazione dei templates. Una volta che è stato collegato un path con una determinata view, quest'ultima è necessario che renderizzi una pagina html. Per tanto è stato di primaria importanza creare i templates html.

Un template è un documento di testo che utilizza un linguaggio che possiede una propria sintassi. In particolare viene eseguito il rendering di un template con un contesto, che è un oggetto simile a un dict che mappa le chiavi ai valori, passatogli tramite la funzione view. Il rendering sostituisce le variabili con i loro valori, che vengono cercati nel contesto, tutto il resto viene emesso così com'è.

La sintassi del linguaggio template Django prevede tre costrutti che sono stati utilizzati nel progetto:

1. **Variabili**, Una variabile assume un valore preso dal contesto, ed ognuna è identificata da una doppia parentesi graffa aperta e chiusa.

2. **Tag**, che possono emettere contenuti, fungere da struttura di controllo, ad es. un'istruzione "if" o un ciclo "for", acquisire contenuti da un database o persino consentire l'accesso ad altri tag modello.

3. **Filtri**, trasformano i valori delle variabili e gli argomenti dei tag. E' come se fossero delle funzioni utilizzabili esclusivamente all'interno dei templates.

```
1  from django import template
2  from app.models import Area, Impianto, CapoArea, User, Chef
3
4  register = template.Library()
5
6  @register.filter
7  def getImpiantiTab(user):
8      impiantiTab = dict()
9      if user.is_staff:
10         aree = Area.objects.all()
11         for area in aree:
12             impiantiTab[area] = Impianto.objects.filter(area=area)
13     else:
14         try:
15             impianto = Chef.objects.get(user = user).impianto
16             impiantiTab[impianto.area] = [impianto]
17         except Chef.DoesNotExist:
18             area = Area.objects.get(capo__user = user)
19             impiantiTab[area] = Impianto.objects.filter(area=area)
20     return impiantiTab.items()
```

Listing 5.1: Porzione di codice del filtro che: dato in input l'utente che effettua una richiesta, mostra nella navigation bar, gli impianti a cui lui ha accesso

5.2 Dettaglio implementativo delle funzioni principali

```
#Data in input una ricetta ritorna un dizionario con chiavi gli ingredienti contenuti,
#associati alla loro quantità.
def ingredientiRicettaConQuantita(ricetta):
    diz = dict()
    contenuti = Contenuto.objects.filter(ricetta=
        ricetta).values_list('ingrediente','quantita').order_by('-quantita')
    for stringa,q in contenuti:
        ing = Ingrediente.objects.get(nome_generico = stringa)
        diz[ing] = int(q)
    return diz

#ritorna un dizionario dei 5 giorni di una settimana di un certo anno
def calcolaGiorniSettimana(year,sett):
    diz = dict()
    first = date(year, 1, 1)
    base = 1 if first.isocalendar()[1] == 1 else 8
    giorni = ['Lunedi','Martedi','Mercoledi','Giovedi','Venerdi']
    for i,giorno in enumerate(giorni, start=0):
        giornoDateTime = first + timedelta(days=base - first.isocalendar()[2]
                + 7 * (sett - 1) + i)
        diz[giorno] = giornoDateTime.strftime('%d/%m/%Y')
    return diz

#Data una ricetta calcola il suo costo totale in un certo impianto
def calcolaCostoRicetta(ricetta,impianto):
    totale = 0.0
    ingRicetta = ingredientiRicettaConQuantita(ricetta)
    for ing,q in ingRicetta.items():
        prezzo = IngredienteImpianto.objects.get(ingrediente=ing,impianto=impianto).listino
        totale+= round(prezzo * q / 1000, 2)
    return round(totale,2)

#Data in input la lista dei 5 menu settimanali, ritorna l'istante piu grande delle modifiche
def getUltimaModifica(menuSettimana):
    ins = set()
    creazione = menuSettimana[0].creazione
    for menu in menuSettimana:
        ha = HistoryMenu.objects.filter(menu=menu).order_by('-istante').first()
        if ha:
            ins.add(ha.istante)
    if ins:
        return max(ins)
    return creazione
```

Listing 5.2: Alcune delle principali *funzioni ausiliarie* utilizzate per modularizzare il lavoro

```python
@login_required(login_url='/login/')
def nuovaRicetta(request,filtro=""):
    portate, ricette = Portata.objects.all(), Ricetta.objects.all()
    tags = ricavaTagsDellaPortata(filtro)
    if request.user.is_staff:
        chefs = list(Chef.objects.filter(manager=True))
        chefs.append("")
    else:
        try:
            requestChef = Chef.objects.get(user = request.user)
            if requestChef.manager:
            chefs=Chef.objects.filter(impianto=requestChef.impianto)
            else: chefs = [requestChef]
        except Chef.DoesNotExist: pass
        try:
            area = Area.objects.get(capo__user=request.user)
            chefs = Chef.objects.filter( Q(impianto__area=area) & Q(manager=True) )
        except Area.DoesNotExist: pass
    if request.method == "POST" and 'cerca' in request.POST:
        return redirect("https://www.google.it/search?q="+ str(request.POST['nome']))
    if request.method == "POST" and 'Save' in request.POST:
        nome = str(request.POST['nome']).lower()
        try: #se già esiste nel db la ricetta con quel nome allora non lo permetto
            if Ricetta.objects.get(nome__iexact=nome):
                messages.error(request, ('ricetta ' +nome+' già esistente nel sistema'))
                return redirect(reverse('nuovaricetta'))
        except: pass #la ricetta è nuova: procedo alla creazione
        if len(str(request.POST['chef'])) > 0 : # è stato assegnato uno chef
            chef = Chef.objects.get(user = str(request.POST['chef']))
        else:         #non è stato assegnato nessuno chef (solo gli admin possono)
            chef = None
        t = Tag.objects.get(nome=str(request.POST['tag']))
        p = Portata.objects.get(nome=str(request.POST['portata']))
        nuova = Ricetta(nome=nome, portata=p,tag=t,chef=chef)
        if len(request.FILES) != 0: nuova.foto = request.FILES['image']
        if request.POST.get("special"): nuova.special = True
        nuova.save()
        messages.success(request, ('Ricetta ' +nome + ' creata con successo!'))
        return redirect(reverse('pivot', kwargs = {'pk' : nome}))

    return render(request, 'app/nuova_ricetta.html', {'tags':tags, 'portate':portate,
    'chefs':chefs, 'filtro':filtro})
```

Listing 5.3: Porzione di codice in cui è stata implementata la view per la **creazione di una ricetta**

5.3 Descrizione del piano dei Test

In questo capitolo vengono elencati i tre principali *casi di test* che sono stati effettuati per ogni sottosistema, ognuno ha l'obiettivo di andare a verificare la correttezza del comportamento del corrispondente caso d'uso definito.

Caso d'uso di riferimento	GA.EffettuaLogin
Obiettivo	Testare il corretto funzionamento del login nel sistema
Precondizioni	L'utente non si è ancora autenticato nel sistema
Azioni	1. L'utente si collega ai sito web per effettuare l'autenticazione 2. L'utente inserisce un username errato ed il sistema deve segnalare l'errore chiedendo di nuovo le credenziali. 3. L'utente inserisce il suo vero username ma con la password errata, il sistema deve segnalare l'errore e chiedere di reinserire la password. 4. L'utente inserisce le sue vere credenziali, verificare che il sistema lo reindirizza alla pagina principale del sistema.
Postcondizioni	Viene creata una nuova sessione per l'utente

Caso d'uso di riferimento	GA.RegistraUtente
Obiettivo	Testare il corretto funzionamento della creazione di nuovi utenti
Precondizioni	L'utente che effettua la procedura è un utente direzionale autenticato
Azioni	1. L'utente inizia la procedura di registrazione di un nuovo utente. 2. L'utente seleziona la tipologia di utente che vuole creare, e successivamente inserisce i dati richiesti dal sistema. 3. Il sistema verifica che non ci sia un utente con lo stesso username, che la password sia valida e che siano stati inseriti tutti i campi. 4. Verificare che il sistema comunichi l'avvenuta registrazione e che il nuovo utente disponga delle funzionalità associate al tipo di account.
Postcondizioni	Viene creato un nuovo utente, della tipologia selezionata, con i dati inseriti

Caso d'uso di riferimento	GA.ModificaDatiDiAccesso
Obiettivo	Testare il corretto funzionamento del cambio dei dati di accesso
Precondizioni	L'utente che effettua la procedura è un utente direzionale autenticato
Azioni	1. L'utente inizia la procedura di cambio dei dati di accesso di un certo utente. 2. L'utente inserisce la nuova password. 3. Il sistema verifica che il campo è stato compilato e che la password inserita rispetti i requisiti. 4. Verificare che il sistema comunichi che l'azione è stata effettuata con successo. 5. Il nuovo utente prova ad accedere con la sua nuova password ed il sistema lo reindirizza alla pagina desiderata.
Postcondizioni	Vengono cambiate le credenziali dell'utente

Caso d'uso di riferimento	GR.CreaRicetta
Obiettivo	Verificare che il sistema permetta la corretta creazione di una ricetta
Precondizioni	L'utente che effettua la procedura è un utente autenticato
Azioni	1. L'utente inizia la procedura di creazione di una nuova ricetta. 2. L'utente inserisce i dati della ricetta. 3. Il sistema verifica che siano stati compilati tutti i campi obbligatori. 4. Verificare che il sistema comunichi che l'azione è stata effettuata con successo e che il database contenga la nuova ricetta con i relativi ingredienti. 5. Verificare che il sistema reindirizzi l'utente alla pagina della ricetta.
Postcondizioni	Il sistema contiene la nuova ricetta

Caso d'uso di riferimento	GR.VisualizzaRicetta
Obiettivo	Verificare che il sistema permetta la corretta visualizzazione delle ricette a cui l'utente ha accesso
Precondizioni	L'utente che effettua la procedura è un utente autenticato
Azioni	1. L'utente inizia la procedura di consultazione delle ricette. 2. L'utente prova a ricercare una ricetta a cui può accedere, ed il sistema verifica che la possa visualizzare correttamente. 3. L'utente prova a ricercare una ricetta a cui non ha accesso, ed il sistema deve segnalare all'utente che non ha i diritti per accedervi.
Postcondizioni	L'utente visualizza la ricetta a cui può accedere correttamente

Caso d'uso di riferimento	GR.ModificaRicetta
Obiettivo	Verificare che il sistema permetta di modificare correttamente una ricetta
Precondizioni	L'utente che effettua la procedura è un utente autenticato. L'utente ha accesso alla ricetta che vuole modificare.
Azioni	1. L'utente inizia la procedura di modifica di una ricetta esistente. 2. L'utente modifica alcuni dei campi tra quelli disponibili. 3. Verificare che il sistema comunichi che le modifiche sono avvenute correttamente, e che si possano da subito visualizzare nella pagina della ricetta.
Postcondizioni	La ricetta è stata modificata correttamente

Caso d'uso di riferimento	GM.CreaMenu
Obiettivo	Verificare che il sistema permetta di creare correttamente un menu settimanale per un certo impianto
Precondizioni	L'utente che effettua la procedura è un utente autenticato. L'utente ha accesso all'impianto.
Azioni	1. L'utente inizia la procedura di creazione di un menu in un certo impianto. 2. L'utente decide se crearlo oppure importarlo, il sistema deve mostrare, in entrambi i casi, le ricette che l'utente può visualizzare per ogni tag selezionato. 3. Verificare che il sistema comunichi un errore se l'utente non inserisce nessuna ricetta, nel caso opposto invece deve comunicare che il menu è stato creato correttamente. 4. Verificare che, dopo la sua creazione, il menu venga mostrato con tutte le ricette inserite e che la data di creazione corrisponda alla data attuale.
Postcondizioni	Il menu è stato creato con successo.

Caso d'uso di riferimento	GM.ModificaMenu
Obiettivo	Verificare che il sistema permetta di modificare correttamente un menu settimanale per un certo impianto
Precondizioni	L'utente che effettua la procedura è un utente autenticato. L'utente ha accesso all'impianto e può modificare il menu.
Azioni	1. L'utente inizia la procedura di modifica di un menu in un certo impianto. 2. L'utente decide se aggiungere o eliminare alcune ricette. In caso di eliminazione il sistema deve comunicare che l'azione è avvenuta, e verificare che non siano più presenti all'interno del menu. In caso di aggiunta di nuove ricette, verificare che il sistema mostri, per i tag selezionati, solo le ricette a cui l'utente ha accesso. 3. Verificare che il menu sia stato aggiornato correttamente e che l'istante di ultima modifica sia uguale all'istante corrente. Il menu deve risultare in fase di approvazione.
Postcondizioni	Il menu è stato modificato con successo.

Caso d'uso di riferimento	GM.GeneraOrdine
Obiettivo	Verificare che il sistema permetta di generare un ordine correttamente per un menu settimanale di un certo impianto
Precondizioni	L'utente che effettua la procedura è un utente autenticato. L'utente ha accesso all'impianto.
Azioni	1. L'utente seleziona i giorni della settimana per i quali vuole generare l'ordine. 2. Verificare che, in caso l'utente non selezioni nessun giorno, il sistema comunichi l'errore all'utente. In caso contrario il sistema deve far visualizzare correttamente tutte le ricette dei giorni selezionati con la relativa porzione da inserire. 3. Verificare che il sistema faccia inserire correttamente le porzioni associate ad ogni ricetta.
Postcondizioni	Il pdf dell'ordine settimanale è stato generato correttamente

5.4 Risultato del testing

Prima di mandare l'applicazione web in produzione sono stati effettuati in locale tutti i test funzionali. Successivamente è stata distribuita ad un gruppo di utenti (in particolare agli chef) che hanno provato ad utilizzarla fornendo i loro feedback e ciò ha permesso di migliorare l'applicazione. Durante questa fase non ci sono stati problemi nello svolgere le funzionalità implementate. Oltre alla correzione di piccoli errori, sono state ottimizzate alcune funzioni e ne sono state aggiunte di nuove. Vengono elencate le principali modifiche che sono state richieste dalla direzione durante questa fase:

- Sono stati richiesti solamente i colori bianco e verde all'interno dell'interfaccia.
- Sono state richieste alcune modifiche grafiche, come per esempio il cambiamento dei titoli delle pagine, grandezza dei bottoni ecc. Anche la presenza di un bottone "torna indietro" sempre presente nella navigazione tra le schermate del sito.
- Nella schermata del menu "visualizza con allergeni" è stato richiesto di far visualizzare anche il numero totale di calorie di ogni piatto presente nel menu.
- Nel pdf che viene generato quando si effettua un ordine è stato richiesto di esprimere le quantità in kg e non in grammi.
- quando si modifica una ricetta è stata aggiunta la possibilità di cambiare le singole quantità dei vari ingredienti presenti.
- E' stata richiesta la creazione di una nuova figura che si chiama "Direttore impianto" che si associa ad un singolo impianto proprio come lo chef.
- Oltre all'area menu è stata richiesta anche un'area inventario su ogni impianto. L'area inventario permetterebbe all"impianto di compilare l'inventario a fine mese partendo dalla lista degli ingredienti. L'impianto inserisce le quantità ed il sistema moltiplica il prezzo per la quantità per stabilire il valore. L'impianto deve selezionare il mese ed il sistema visualizzerà l'intera lista dei prodotti a cascata, magari filtrabile per fornitore e ricercabili manualmente, ai quali il direttore e lo chef possono aggiungere la quantità. Scritta la quantità possono aggiungere un nuovo prodotto in inventario . Questo consentirebbe loro di valutare le giacenze disponibili in magazzino prima di ordinare.
- Quando si crea un menu è stata data la possibilità di inserire un massimo di 20 piatti in ogni giorno, fornendo quindi una struttura flessibile.
- Quando si crea un menu si può scegliere, opzionalmente, la possibilità di aggiungere esclusivamente i piatti creati dall'impianto stesso.

6. Conclusioni

6.1 Sommario del lavoro fatto ed eventuali sviluppi futuri

Durante la scrittura di questa tesina e durante la fase di testing sono emersi molti spunti interessanti riguardanti il sistema, che hanno consentito di modificare alcune funzionalità presenti e di programmarne di nuove, e al momento per questione di tempi ancora non tutte sono state realizzate.

Abbiamo visto come gli utenti del sistema sono stati in grado di creare i menu all'interno del loro impianto senza problemi, questo anche grazie all'usabilità dell'interfaccia che è stata appositamente costruita con l'obiettivo di essere semplice ed efficace. Anche il processo di creazione di una ricetta è stato perfezionato man mano, ora per esempio, selezionando la portata della ricetta che si vuole creare vengono mostrati solamente i tag che si riferiscono ad essa, evitando di dare la possibilità di creare ricette invalide.

Un passaggio di fondamentale importanza è stato l'ottimizzazione del codice per garantire di effettuare le query al database in tempi efficienti, fornendo un riscontro rapido all'utente che effettua una ricerca.

Si è rivelato molto efficiente fornire un sistema di *"paginazione"* all'interno delle pagine in cui vengono mostrati molti risultati, questo è fondamentale per garantire un caricamento rapido di queste, evitando quindi tempi di attesa troppo lunghi dopo che un utente abbia effettuato la richiesta.

La scelta di dividere ogni anno in 52 settimane si è rivelata molto efficace, infatti è stato possibile realizzare una funzione, utilizzando le librerie di Python, che ha permesso di estrarne da ognuna di essa le date corrispondenti.

6.2 Ringraziamenti

Questo tirocinio per me è stata un'esperienza interessante e formativa, ringrazio Emiliano Casalicchio per avermi dato questa possibilità. In questo contesto ho potuto affrontare toccandoli con mano molti argomenti che sono stati oggetto dei miei studi nei vari corsi di questi 3 anni di università, permettendomi quindi di passare dall'aspetto teorico a quello pratico, potendoli applicare e far prendere forma ad un sistema software reale. Tra questi c'è la progettazione software con in particolare la progettazione del database e delle funzionalità, la creazione dei templates utilizzando html,css e javascript, la creazione di interfaccie che rispettino il material design, l'approfondimento delle conoscenze linux per la fase di deployment in un server remoto, aver programmato con il linguaggio di programmazione Python. Un ringrazimento anche a Manuel Covuccia per la sua disponibilità e per avermi seguito nella progettazione di questa applicazione web fornendomi continui riscontri sulle migliorie da effettuare.

Il progetto è reperibile al seguente link: `https://github.com/AlbertoCotu/tirocinio_cotumaccio`