Winer
System Design
Versione 1.0.5

Antica enoteca di Roma - Unsplash

Giovanni Prisco
0512106534

Indice

Revision History

Data	Versione	Descrizione
04/01/2022	1.0.0	Creazione del System Design Document
06/01/2022	1.0.1	Stesura dell’introduzione, definizione dei design goals
08/01/2022	1.0.2	Completata introduzione e discussione sull’architettura proposta
09/01/2022	1.0.3	Stesura della sezione sui Dati Persistenti
10/01/2022	1.0.4	Completato capitolo 3 + prima stesura capitolo 4
18/01/2022	1.0.5	Ultima revisione del documento per continuare con i prossimi step

1. Introduzione

Nel Documento “Requirements Analysis” abbiamo messo in evidenza tutti i requisiti e i casi d’uso, sulla base di questi possiamo ora andare ad individuare i sottosistemi che insieme danno vita a Winer.

1.1 Scopo del Sistema

Come abbiamo potuto constatare dal Problem Statement al più recente Requirements Analysis document, il Sistema si propone di essere un e-commerce di vini. Gli utenti possono collegarsi e acquistare il vino che più gli piace per riceverlo a casa.

D’altro canto questo e-commerce dovrà essere gestito da Giuseppe, il quale dovrà mantenere il catalogo aggiornato e gestire gli ordini effettuati sulla piattaforma.

Dobbiamo definire ora gli obiettivi di progettazione che possiamo trarre dall’analisi dei requisiti fatta nell’ultimo documento.

1.2 Obiettivi di Progettazione

La piattaforma in questione, come già detto, è un e-commerce, il che vuol dire che Giuseppe guadagnerà quanto più gli utenti si collegheranno al sito ed effettueranno gli acquisti, passiamo in rassegna quindi gli obiettivi che possiamo fissare:

Reliability

Giuseppe si aspetta ovviamente che la piattaforma sia sempre online pronta a soddisfare le richieste da parte dei clienti. Sarà compito nostro in fase di deployment assicurarci che l’applicativo sia sempre raggiungibile e **fault-tolerant** nel senso che in caso di errori tecnici sia capace di eseguire un riavvio o sostituire un’istanza già pronta a quella andata in errore. Inoltre la piattaforma dovrà prevedere input errati da parte degli utenti e gestire casi in cui i dati ricevuti non siano nel formato atteso e impedire altre operazioni con questi.

Maintainability

Passiamo ora ai design goals strettamente legati alle operazioni di sviluppo e deployment, primo fra tutti è l’obiettivo di manutenibilità.

Per raggiungere quest'obiettivo svilupperemo un'applicazione three tier: il client seguirà un'architettura Model View View Model (MVVM), mentre il server esporrà la logica di business tramite delle REST APIs nascondendo tutti i meccanismi interni al client che si connette per usufruirne. I vantaggi di questa scelta risiedono anche in riutilizzo del sistema e nell'adattabilità di questo a diversi contesti.

Reusability and Adaptability

Per quanto detto prima, abbiamo il massimo in quanto a riutilizzo, prevediamo un caso in cui Giuseppe ci chieda di sviluppare anche un'applicazione nativa per iOS e Android che permetta agli utenti di utilizzare Winer, le REST API sviluppate potranno essere riutilizzate completamente e senza la minima modifica.

Il trade-off di questo nostro obiettivo sarà la **Readability** poichè utilizzeremo costrutti più complessi che richiederanno più esperienza da parte dello sviluppatore per poterne capire subito i meccanismi.

Traceability of requirements

Quanto a tener traccia dei requisiti, ci stiamo già lavorando scrivendo queste documentazioni che dovranno essere in continuo aggiornamento per assicurarci che rispecchino sempre lo "state of the art".

Backward-compatibility

Riguardo questo obiettivo dobbiamo tenere presenti 2 cose:

- Nel caso di un'applicazione web, l'utente al momento dell'apertura avrà sempre la versione più aggiornata

- Per quanto riguarda la logica di business, le REST API dovranno essere versionate, prevedendo quindi metodi accessori basati sulla versione desiderata (ad esempio **v1, v2...**), questi meccanismi ci assicureranno che anche in caso di aggiornamenti, non andremo ad impattare eventuali utenti già connessi che stanno utilizzando una versione precedente dell'applicativo

Rapid Development

Il Cliente si aspetta una dimostrazione della prima versione dell'applicativo molto presto, per questo motivo dobbiamo cercare di sviluppare l'applicazione il più velocemente possibile sacrificando (**trade-off**) alcune funzionalità, in questo caso la gestione degli ordini, daremo a Giuseppe il minimo richiesto affinché i clienti possano cominciare ad effettuare acquisti e lui possa gestire il catalogo offerto. Nelle settimane successive offriremo a Giuseppe la possibilità di approvare lui stesso gli ordini effettuati.

1.3 Definizioni, acronimi e abbreviazioni

MVC: Model View Controller

MVVM: Model View View Model

API: Application Programming Interface

1.4 Riferimenti

Questo documento ha riferimenti per i seguenti documenti:

- Requirements Analysis Document: qui sono descritte le funzionalità individuate in fase di analisi

1.5 Overview

Il seguente documento mostra la progettazione del sistema e una sua prima divisione in sottosistemi.

Nella prima sezione abbiamo ricapitolato i punti svolti finora.

Nella seconda sezione cominceremo una breve discussione sull'architettura di un e-commerce e di come possiamo "copiare dai migliori".

Nella terza sezione, fatti nostri i principi presentati nella seconda, introdurremo l'architettura pensata per Winer e le varie scelte riguardanti la decomposizione, la persistenza, l'access control, il mapping software/hardware e altro ancora.

Nella quarta e ultima sezione invece passeremo in rassegna tutti i servizi esposti da ognuno sei sottosistemi individuati nelle sezioni precedenti.

2. Discussione sull'Architettura

Per cominciare il discorso sull'architettura, prendiamo come esempio l'e-commerce per antonomasia, Amazon, le informazioni che seguono vengono da un Case Study presentato in occasione dell'evento AWS re:Invent 2017 (È possibile vedere le slide qui).

I principi guida che hanno portato Amazon a definire la loro architettura a microservizi sono:

- Security
- Availability
- Latency
- Cost

Amazon conta migliaia di servizi utilizzati dagli utenti, questi utenti accedono ai servizi da ogni parte del mondo, è molto importante ridurre la latenza affinché ogni utente possa avere la stessa esperienza fluida indipendentemente dalla sua locazione in quel momento.

Eviteremo di parlare in dettaglio dei meccanismi di Route 53 di Amazon e di come si possa ottenere la latenza minima tra richiesta del servizio e la risposta, ci limitiamo però a sottolineare l'importanza di avere questi servizi loose-coupled che sono distribuiti in diverse locazioni per garantire **high-availability**.

Durante lo sviluppo di Winer ci impegneremo a seguire questi principi considerando alcuni importanti punti:

- Il nostro target (e quello di Giuseppe) si limita per ora alla sola Italia
- I costi e i tempi di sviluppo non ci permettono di pensare un'architettura a microservizi dal giorno 0, svilupperemo quindi le REST API in un'applicazione monolitica per esporre i servizi principali necessari all'MVP, monteremo però un Load Balancer per servire il client e le API e predisporre all'aggiunta di servizi separati accessibili tramite un API Gateway

3. Architettura Proposta

3.1 Overview

Come abbiamo già accennato nei paragrafi precedenti, l'architettura di Winer sarà three-tier, per quanto riguarda il server andremo a sviluppare delle REST API a cui il Client potrà accedere, le quali si connetteranno al server del database per la persistenza dei dati.

Il Client implementerà l'architettura MVVM.

3.2 Decomposizione in Sottosistemi

Partiamo con la decomposizione del sistema server-side: le REST APIs.

Individuiamo ovviamente il sottosistema **Model** il quale si occuperà (come nel MVC) di interagire con gli schemi del database e quindi la persistenza.

Oltre al Model, ci sarà sicuramente un **Controller** il quale si occuperà di intercettare la chiamata HTTP, elaborare gli input e fornire il giusto servizio.

Infine ci saranno i **Services**, i veri e propri servizi che implementano la logica di business.

Il Client invece sarà composto (ovviamente) da una **View** e da un sottosistema che si occupa di comunicare con le REST APIs che chiameremo **Model**.

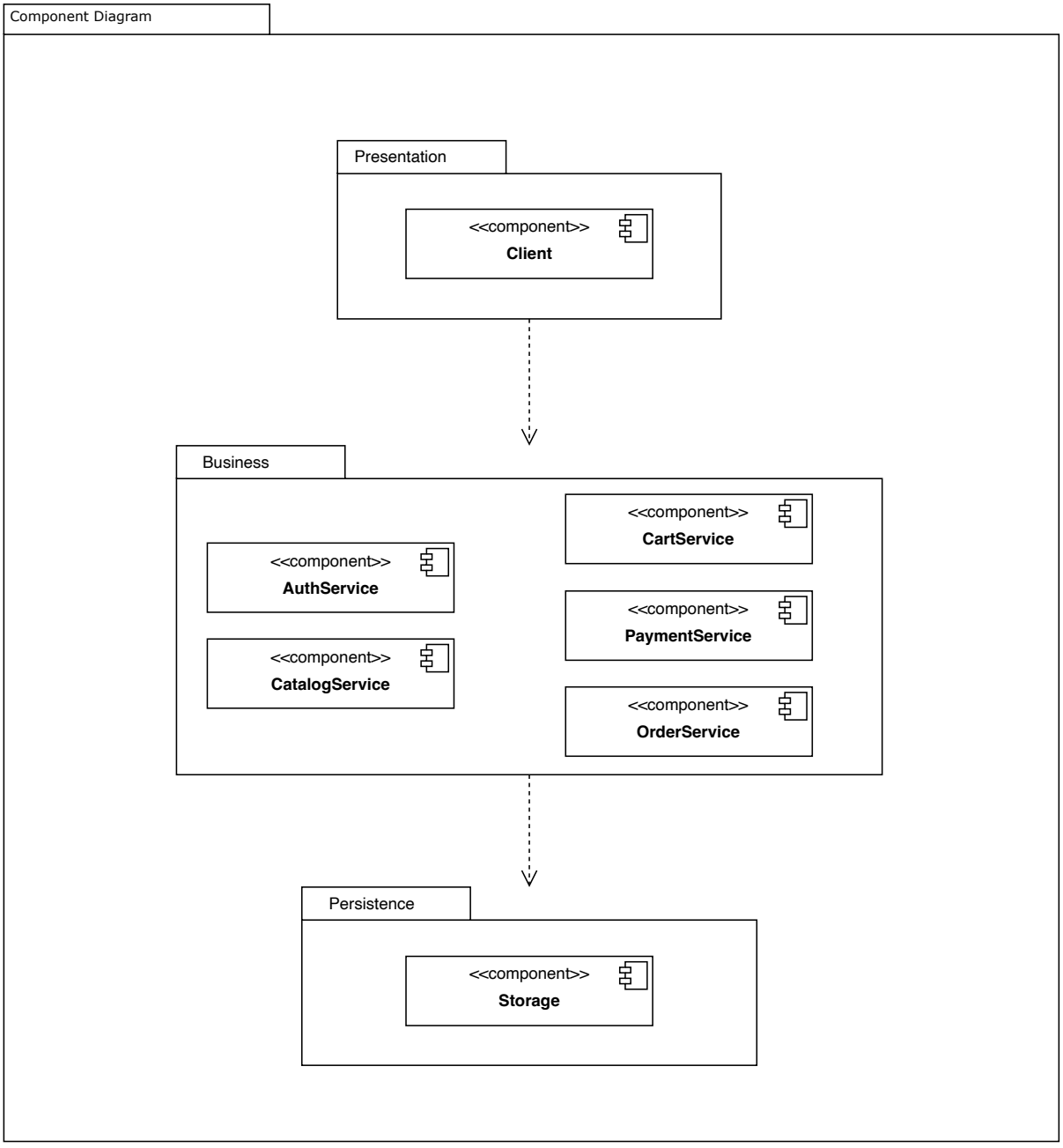
Winer sarà quindi implementato con l'applicazione client che utilizza la logica di business tramite le REST API esposte dall'applicazione server, cercheremo di mantenere l'applicazione client quanto più leggera possibile lasciando tutta la complessità al server.

Di seguito alcuni diagrammi che dimostrano la combinazione di questi componenti.

Componente	Descrizione
Client	Applicazione Client che gestisce l'interazione dell'utente con il sistema
AuthService	Gestisce i meccanismi di registrazione, autenticazione e autorizzazione
CatalogService	Implementa le operazioni crud effettuabili sul catalogo

Componente	Descrizione
CartService	Si occupa di gestire la logica relativa al carrello dell'utente
PaymentService	Gestisce la parte relativa al checkout, pagamento e emissione della ricevuta
OrderService	Gestisce la parte relativa agli ordini effettuati su Winer

Mettendo insieme questi componenti tiriamo fuori un primo Component Diagram.



3.3 Mapping Hardware/Software

Il Sistema, come detto, implementa un'architettura three-tier.

Application Server

Un Load Balancer (NGINX), fa da webserver per servire l'applicazione client e da reverse proxy server per le possibili multiple istanze dell'applicazione server e per il persistence server.

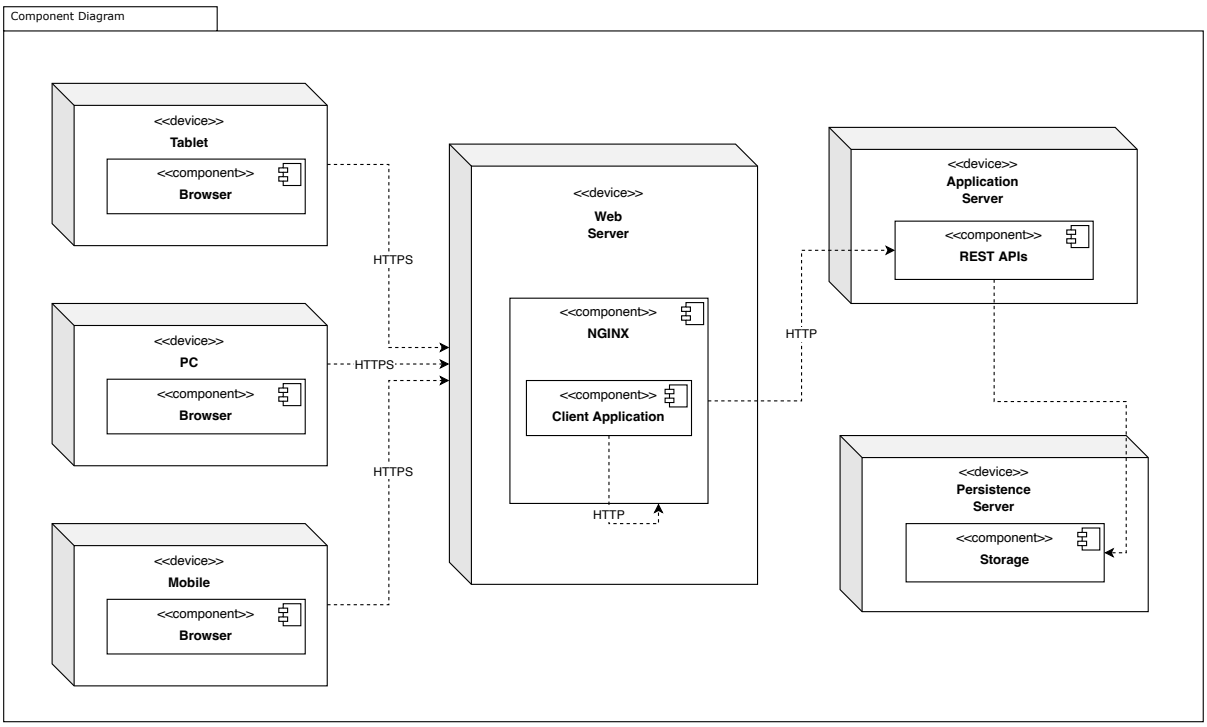
Client

Il Browser si occuperà di mostrare l'applicazione client e di permettere l'interazione tra questa e l'utente.

Il Client comunicherà con il Server attraverso chiamate HTTP.

Persistence Server

L'Application Server è l'unico a dover comunicare con il Persistence Server, dove risiederà il database, grazie alle apposite API.



3.4 Gestione dei Dati Persistenti

La Gestione dei Dati persistenti avverrà tramite un database di tipo SQL, in questo caso abbiamo scelto MySQL, di seguito troviamo le informazioni sulle tabelle da avere per gestire i dati persistenti presentati nel documento di analisi dei requisiti.

Utilizzeremo la seguente convenzione per i nomi delle tabelle, applicheremo un prefisso per distinguere le tabelle che riguardano classi di dati differenti: **a**_ per i dati che riguardano l'autenticazione, **w**_ per i dati che riguardano i vini, **c**_ per i dati del carrello, **g**_ per i dati geografici e infine **o**_ per i dati riguardanti gli ordini.

Per evitare inoltre di riempire il documento di codice SQL, assumeremo che le colonne che portano lo stesso nome degli ID di altre colonne, sono foreign keys e che sia di default ON DELETE CASCADE, tranne per gli ordini effettuati che ovviamente non vogliamo eliminare nel caso in cui venga a mancare qualcuno dei vini correlati.

a_user

1. *userID*: int(11)
2. *emailAddress*: varchar(320)
3. *password*: varchar(255)

La tabella **a_user** memorizza gli utenti registrati a Winer, il campo password potrà essere ulteriormente ottimizzato dopo aver scelto l'algoritmo di criptazione.

a_role

1. *roleID*: int(11)
2. *roleName*: unique varchar(30)
3. *roleDescription*: varchar(255)

La tabella **a_role** contiene tutti i ruoli possibili di su Winer, in questo caso un ruolo di default *UtenteRegistrato* che si otterrà in fase di registrazione e un ruolo *Gestore* che andremo ad attribuire manualmente agli account che vogliamo (inizialmente Giuseppe).

a_user_role

1. *roleID*: int(11)
2. *userID*: int(11)
3. issuedAt: datetime

La tabella associativa **a_user_role** mantiene le relazioni molti a molti tra gli utenti e i relativi ruoli attribuiti. Abbiamo deciso di mantenere questa relazione molti a molti prevedendo evolutive tra cui quella di aggiungere nuovi ruoli a Winer.

Passiamo ora ai dati relativi al Catalogo (vini e relative caratteristiche).

w_winecolor

1. *winecolorID*: int(11)
2. *winecolor*: varchar(20)

Una tabella **w_winecolor** manterrà i colori possibili dei vini, prevediamo evolutive future nel caso in cui Giuseppe vorrà avere la possibilità di aggiungere birre o altri alcolici al catalogo e basterà aggiungere i relativi colori e tipologie (come vedremo nella prossima tabella) per far avere a Giuseppe tutto ciò che serve.

w_winetype

1. *winetypeID*: int(11)
2. *winetype*: varchar(20)

Per la tabella **w_winetype** va fatto lo stesso discorso fatto per **w_winecolor**, questa mantiene le diverse tipologie di vino come ad esempio fermo o frizzante… prevediamo l'evolutiva esposta sopra.

g_country

1. *countryID*: int(11)

2. *country*: varchar(100)

La tabella **g_country** contiene le nazioni del mondo da cui provengono i vari vini.

g_region

1. *regionID*: int(11)
2. *region*: varchar(100)
3. *countryID*: int(11)

La tabella **g_region** contiene le regioni delle nazioni del mondo da cui provengono i vini.

w_winefamily

1. *winefamilyID*: int(11)
2. *winefamily*: varchar(100)
3. *winecolorID*: int(11)
4. *winetypeID*: int(11)
5. *regionID*: int(11)

La tabella **w_winefamily** conserva le famiglie di vini con le relative caratteristiche (colore, tipologia e provenienza) oltre ad un nome.

w_winery

1. wineryID: int(11)
2. winery: varchar(100)
3. address: varchar(100)
4. telephone: varchar(100)

La tabella **w_winery** contiene le aziende produttrici di vini e alcune informazioni di contatto.

w_wine

1. *wine*: varchar(255)
2. *vintage*: int(11)
3. *wineryID*: int(11)
4. *winefamilyID*: int(11)
5. *availability*: int(11)
6. *price*: float
7. PRIMARY KEY(*wine*, *vintage*)

Arriviamo alla tabella **w_wines**, questa conserva ovviamente i dati dei vari vini, collegandoli opportunamente alla famiglia, nonché all'azienda che li ha prodotti, contiene inoltre un nome e un'annata (che insieme identificano univocamente un vino) ed un prezzo e una disponibilità ai fini della vendita su Winer.

w_winegrape

1. winegrapeID: int(11)
2. winegrape: varchar(100)

La tabella **w_winegrape**, come si evince dal nome, contiene le informazioni sulle varie uve esistenti.

w_wine_winegrape

1. wine: varchar(255)
2. vintage: int(11)
3. winegrapeID: int(11)
4. percentage: int(11)

La tabella **w_wine_winegrape** è una tabella associativa per la relazione tra vini e uve, questa tabella ci permette di mostrare l'uvaggio dei vari vini con le relative percentuali per ogni uva presente.

c_cart_item

1. *wine*: varchar(255)
2. *vintage*: int(11)
3. *userID*: int(11)
4. *quantity*: int(11)

La tabella **c_cart_item** conserva i vini presenti nel carrello di un utente con la relativa quantità impostata da quest'ultimo. Si tratta di una tabella associativa per la relazione molti a molti tra utenti e vini.

p_order

1. *orderID*: int(11)
2. *createdAt*: datetime
3. *userID*: int(11)
4. *confirmed*: boolean

La tabella **p_order** contiene gli ordini effettuati su Winer da un determinato utente.

p_order_wine

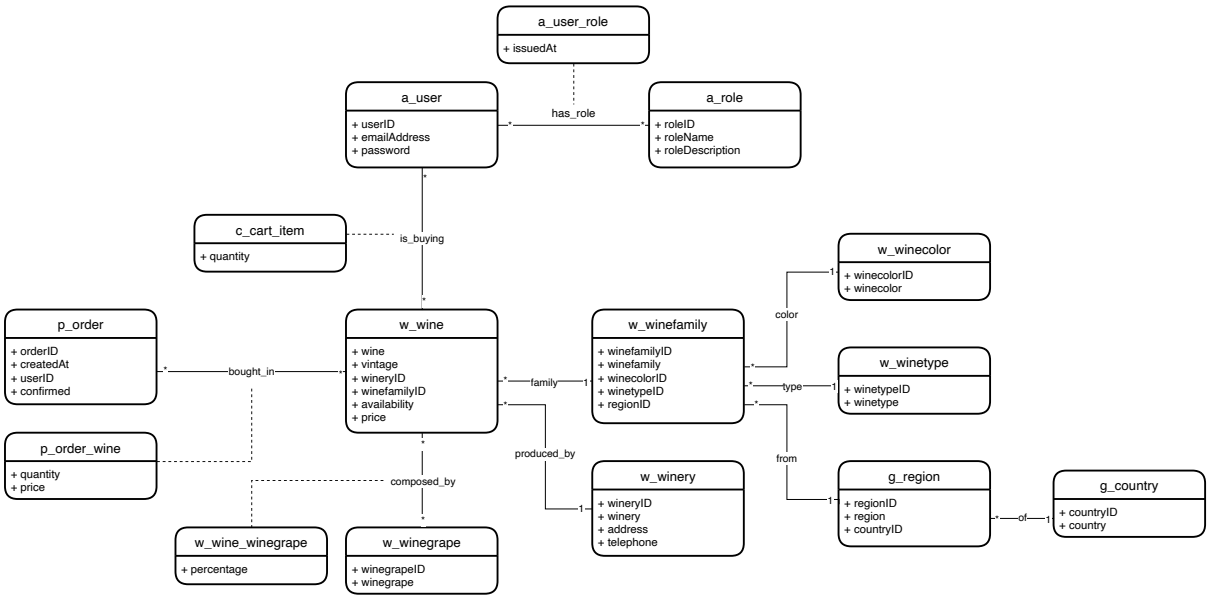
1. *wine*: varchar(255)
2. *vintage*: int(11)
3. *orderID*: int(11)
4. *quantity*: int(11)
5. *price*: float

La tabella **p_order_wine** è una tabella associativa per la relazione tra ordini e vini.

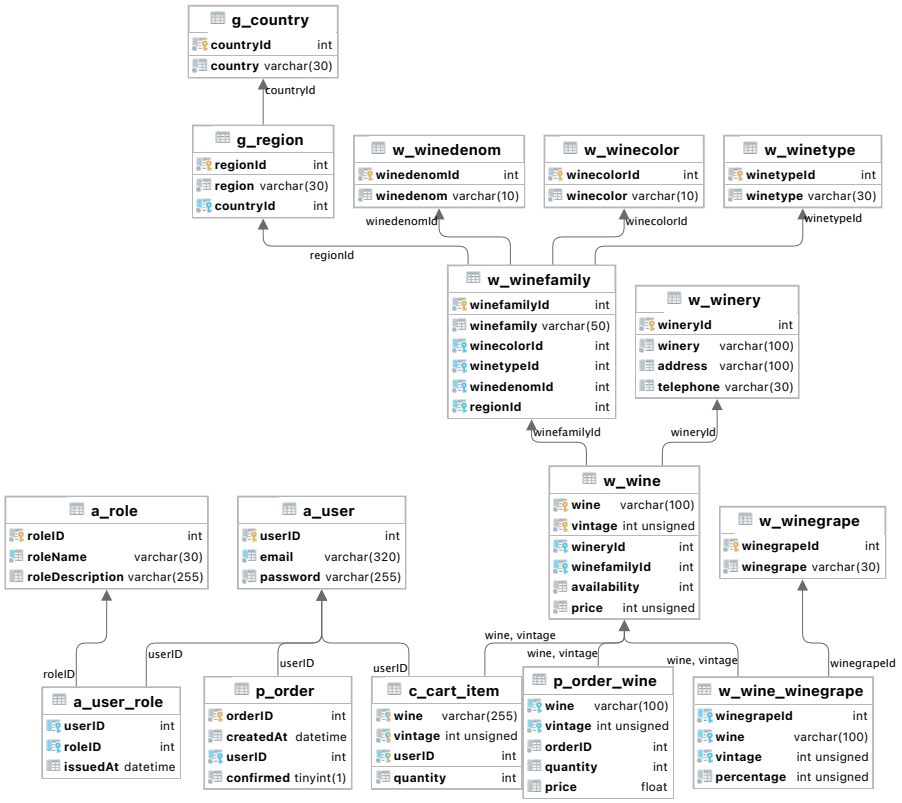
Abbiamo deciso di ridonare il prezzo dei vini in questa tabella per ovvi motivi, ovvero nel caso in cui Giuseppe decida di aumentare o diminuire il prezzo dei

vini questo verrebbe aggiornato anche per i vari ordini, noi vogliamo fissare il prezzo al momento di finalizzazione dell'ordine.

Di seguito il diagramma di persistenza del database come descritto finora.



Una volta implementato questo schema in MySQL, il risultato sarà il seguente:



3.5 Controllo degli Accessi

Winer permette agli utenti di registrarsi e loggarsi per effettuare determinate operazioni protette come la finalizzazione di un ordine.

Attualmente consentiamo a 3 tipologie di utenti di navigare in Winer: Guest, Registrato e Gestore. Ognuno di questi utenti avrà dei permessi per poter effettuare operazioni sui vari oggetti coinvolti, visualizziamoli un una matrice per il controllo degli accessi.

N.B. La dicitura CRUD si riferisce all'insieme delle operazioni Create, Read, Update e Delete.

Attore / Oggetto	Wine	Winefamily	Winery	Winegrape	Order	CartItem
Guest	Read	Read	Read	Read	-	CRUD
Registrato	Read	Read	Read	Read	Create, Read	CRUD
Gestore	CRUD	Read	Read	Read	CRUD	CRUD

Gli oggetti che non sono inseriti nella tabella sopra sono utilizzati semplicemente per fornire dati per il corretto funzionamento dell'ecommerce.

3.6 Global Software Control

Il sistema in questione è di tipo event-based, tutte le operazioni vengono svolte in risposta a delle azioni dell'utente.

Trattandosi di una Single Page Application, l'unico momento in cui l'utente chiama direttamente il Web Server è all'apertura di Winer, successivamente tutte le operazioni vengono eseguite in risposta ai suoi input.

La View del Client prende le richieste dell'utente ed interagisce con il ViewModel, il quale chiede al Model di effettuare le chiamate REST al backend, che a sua volta prende in carico la richiesta e tramite un dispatcher la passa al giusto Controller, il quale una volta validata la richiesta, ed eventualmente autenticata, utilizza i Servizi per eseguire l'operazione richiesta ed interagire con il Model (e quindi col Database se ce n'è bisogno), ritorna la risposta al Model del Client, il quale può ora fornire i dati al ViewModel affinché aggiorni la View.

“Il ***view model*** fa da intermediario tra la vista e il modello, ed è responsabile per la gestione della logica della vista. In genere, il *view model* interagisce con il modello invocando metodi nelle classi del modello. Il *view model* fornisce quindi i dati dal modello in una forma che la vista può usare facilmente.” (Wikipedia)

3.7 Boundary Conditions

Le Boundary Conditions riguardano l’inizializzazione, la terminazione ed eventuali failures del sistema.

Inizializzazione del Sistema

Identificativo	**BC01 - StartServer**
Attori	Amministratore
Flusso di Eventi	- L’amministratore accede al Server e invia un comando di Start - Il Sistema prende in carico il comando di Start e avvia tutti i servizi necessari al funzionamento di Winer

Terminazione del Sistema

Identificativo	**BC02 - StopServer**
Attori	Amministratore
Flusso di Eventi	- L’amministratore accede al Server e invia un comando di Stop - Il Sistema prende in carico il comando di Stop e smette di accettare nuove richieste, una volta finito di servire le richieste già prese in carico comincia a terminare tutti i servizi componenti di Winer

Fallimento del Sistema

Identificativo	**BC03 - RestartApplication**
Attori	Amministratore
Flusso di Eventi	- L’applicazione di Winer termina per un errore non gestito correttamente, il Sistema sostituisce l’istanza “rotta” con una nuova già pronta a ricevere richieste

Backup del Sistema

Identificativo	**BC04 - BackupServer**
Attori	Sistema

Flusso di Eventi	- Il Sistema genera periodicamente snapshot dello stato corrente e li invia ad un server separato al fine di conservarlo per una futura necessità di restore

4. Servizi dei Sottosistemi

Nel paragrafo **3.2** abbiamo parlato dei vari sottosistemi che compongono Winer, ora passiamo in rassegna tutti i servizi offerti da questi sottosistemi.

4.1 Auth Service

Il primo sottosistema di cui parleremo è quello che si occupa di tutte le operazioni necessarie all'autenticazione e all'autorizzazione degli utenti.

Di seguito forniamo un tabella che racchiude i servizi offerti:

Servizio	Descrizione
Register	Crea una nuova utenza nel database occupandosi anche dei meccanismi di criptazione della password fornita
Login	Prende in input un username e una password e controlla che l'utenza con quell'username nel database abbia il corretto hash della password fornita. Fornisce un JSON Web Token (JWT) che il Client dovrà fornire nelle successive richieste per accedere ai servizi protetti
Authorize	Prende in input un JWT e lo valida, si occupa di controllare l'accesso ai servizi protetti

4.2 Catalog Service

Il Catalog Service si occupa di tutti i servizi relativi al catalogo, quindi le operazioni sui vini.

Servizio	Descrizione
Get Wines	Permette di leggere i vini presenti su Winer opportunamente paginati e con eventuali filtri applicati

Servizio	Descrizione
Create Wine	Permette di creare un nuovo vino e aggiungerlo al catalogo
Update Wine	Modifica un vino già presente su Winer
Delete Wine	Elimina un vino presente su Winer

4.3 Cart Service

Servizio	Descrizione
Add to Cart	Crea un nuovo CartItem relazionandolo al vino selezionato ed eventualmente all'utente collegato
Update Quantity	Permette di aggiornare la quantità di un vino presente nel carrello
Remove from Cart	Elimina un vino dal carrello, indipendentemente dalla quantità

4.4 Payment Service

Alcune note su questo servizio, in un caso reale andrebbe utilizzato un servizio come Stripe o andrebbe stabilita un'infrastruttura adeguata che sia a norma per inviare e conservare dati sensibili riguardanti carte di credito.

Per ora andiamo a stabilire un servizio mock che imita le funzionalità di base che vogliamo avere senza ovviamente effettuare transazioni reali.

Servizio	Descrizione
Start Session	Avvia una sessione per ottenere il form dedicato all'inserimento dei dati di pagamento
Pay	Metodo per finalizzare il pagamento, prende l'esito e se positivo utilizza OrderService per registrare l'ordine effettuato
Refund	Metodo utilizzato per emettere un rimborso

4.5 Order Service

Servizio	Descrizione
Create Order	Crea un nuovo ordine relazionandolo all'utente collegato e ai vini acquistati
Update Order	Permette di aggiornare lo stato di un ordine, in particolare la colonna "confirmed"
Delete Order	Utilizzato per respingere un ordine, invia una mail all'utente che ha effettuato l'ordine ed utilizza PaymentService.refund per emettere un rimborso