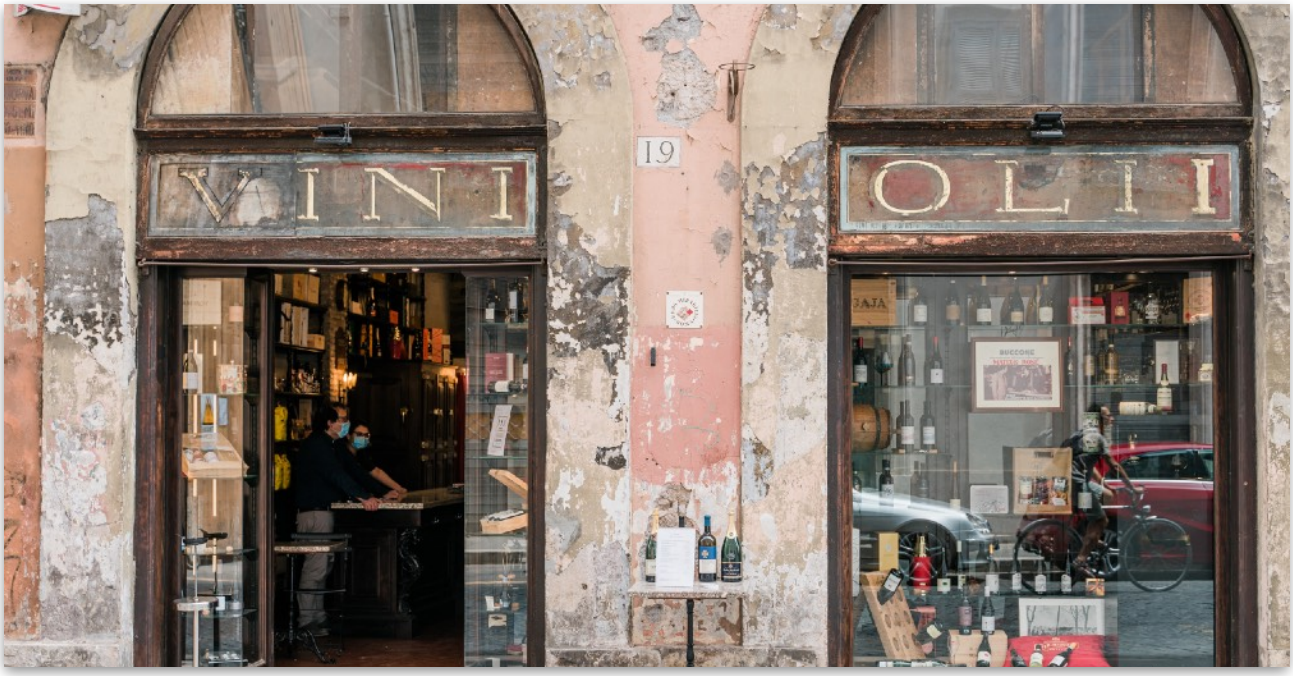
Winer
Object Design
Versione 1.0.1



Antica enoteca di Roma - Unsplash

Giovanni Prisco
0512106534

Indice

Revision History

Data	Versione	Descrizione
18/01/2022	1.0.0	Creazione Object Design Document
31/01/2022	1.0.1	Prima stesura completa del documento

1. Introduzione

Con il seguente documento si intende introdurre tutti i dettagli riguardanti l'implementazione di Winer, a differenza di quanto fatto nei precedenti in cui abbiamo tralasciato i dettagli implementativi concentrandoci più sull'architettura generale del sistema.

Lo scopo finale del documento è quello di fornire un modello applicabile per implementare tutte le funzionalità individuate nei documenti di "System Design" (SDD) e "Requirements Analysis" (RAD). In particolare vogliamo fornire le interfacce e le classi comprensive di parametri e firme delle varie funzioni dei sottosistemi individuati nel SDD.

1.1 Linee Guida per la Documentazione delle interfacce

Di seguito un elenco delle convenzioni da utilizzare durante la scrittura del codice.

Naming Conventions

Le nomenclature dovranno essere il più descrittive possibili e di lunghezza medio-corta. Inoltre tutti i nomi dovranno essere camelCase, fatta eccezione per l'access_token fornito in fase di autenticazione per rispettare lo standard dei JSON Web Token (JWT, RFC 7519).

Variabili

I nomi delle variabili cominciano tutti per lettera minuscola e come detto prima seguono una nomenclatura camelCase. Per quanto riguarda le costanti utilizzeremo parole maiuscole separate da caratteri *underscore* "_".

Metodi

Anche i nomi dei metodi saranno camelCase. I nomi dei metodi dell'applicativo **backend** identificano il tipo di accesso alla risorsa in questione la quale è identificata dalla classe di appartenenza del metodo, seguendo le convenzioni riguardanti le REST APIs: alcuni esempi della classe **Wine** (risorsa) che contiene i metodi CRUD: **create**, **findAll**, **findOne**, **update**, **delete**.

Classi e Pagine

I nomi delle classi e delle pagine seguiranno invece una nomenclatura Pascal-Case.

Solitamente un file sorgente contiene una e una sola classe e dovrà essere inserito nel modulo corretto in base alle sue funzionalità.

1.2 Riferimenti

Il documento corrente contiene riferimenti a tutti i documenti forniti in precedenza e dà per scontato la presa visione dei casi d'uso e dei vari requisiti esposti in: SDD, RAD e Problem Statement.

2. Packages

2.1 Backend

L'implementazione del progetto **backend** (cartella */Project/backend/src*) si basa su NestJS, un framework basato su Node.js ed Express che fa uso del Dependency Injection Pattern e si presenta con i seguenti pacchetti, ogni pacchetto rappresenta una risorsa, in ogni risorsa troviamo i data transfer objects (DTO) e le entità che ci permettono di accedere al database utilizzando TypeORM e il suo repository pattern:

- **auth**
- **cart**
- **config**
- **hasher**
- **helpers**
- **middlewares**
- **order**
- **payment**
- **user**
- **wine**
- **wine-winegrape**
- **winefamily**
- **winegrape**
- **winery**

Fuori dalla directory *src* troviamo file di configurazione e il package.json che gestisce gli script del progetto e le dipendenze.

Inoltre nella directory *tests* andremo ad implementare eventuali test e2e ed integration tests.

Alcune convenzioni utilizzate nella stesura delle classi:

- Le classi che terminano per -Module sono riservate a NestJS e si occupano di definire gli import e gli export delle classi utili alla Dependency Injection;
- Le classi che terminano per -Controller sono utilizzate per definire gli endpoint attraverso cui è possibile accedere ai servizi del backend;
- Le classi che terminano per -Service incapsulano la logica di business dei vari packages;
- Le classi che terminano con -Entity servono ad associare l'oggetto in questione all'entità del database, fornendo quindi informazioni quali colonne, tipo, chiavi, indici e relazioni;
- Le classi contenute nelle directory d*to* rappresentano semplici interfacce utili a definire il tipo di richiesta e il tipo di risposta di ogni controller, implementate tramite classi poiché Typescript non mantiene a livello di applicazione i metadati contenuti nelle interfacce, i quali vengono persi nel momento in cui si richiama l'interfaccia da un altro modulo.

La documentazione relativa agli endpoint segue lo standard OpenAPI e può essere visualizzata tramite Swagger UI avviando il progetto backend e navigando al path **/api/swagger**.

/auth

Classe	Descrizione
AuthModule	Classe Wrapper per Dependency Injection —> Modulo NestJS
AuthController	Classe Controller che definisce gli endpoint per accedere ai servizi di Auth con le relative informazioni sul tipo di richiesta e sulla struttura del payload o dei parametri da fornire
AuthService	Incapsula la logica di business dell'auth
JwtStrategy	Gestisce la logica dietro la firma e la validazione dei json web tokens
LocalStrategy	Si occupa della strategia di autenticazione
JwtGuard	Decoratore che richiama JwtStrategy per autorizzare le richieste che arrivano ai vari controller
Roles	Decoratore che imposta metadati riguardo i ruoli richiesti per accedere a classi o metodi

Classe	Descrizione
RoleGuard	Decoratore che utilizza i metadati creati da Roles per autorizzare le richieste

/cart

Classe	Descrizione
CartModule	Modulo NestJS
CartController	Classe Controller che definisce gli endpoint per accedere ai servizi di Carrello con le relative informazioni sul tipo di richiesta e sulla struttura del payload o dei parametri da fornire
CartService	Incapsula la logica di business del carrello
CartEntity	Classe Wrapper per accedere alla persistenza del carrello, è necessario inserirla in un repository di TypeORM affinché si possa accedere al database

/config

Classe	Descrizione
Env	Si occupa di parlare dati da diversi file di configurazione (o variabili d'ambiente) e fornirli all'applicazione attraverso un servizio globale di configurazione che implementa un'interfaccia comune richiamabile ovunque

/hasher

Classe	Descrizione
HasherModule	Modulo NestJS
HasherService	Incapsula funzioni relative ad operazioni di cash

/helpers

Classe	Descrizione
HelpersModule	Modulo NestJS
PaginationService	Classe base che permette di impaginare le entità che vengono dal database grazie all'utilizzo dei generici

/middlewares

Classe	Descrizione
LoggerMiddleware	Middleware utilizzato per loggare informazioni sulle richieste che arrivano ad ogni controller

/order

Classe	Descrizione
OrderModule	Modulo NestJS
OrderController	Classe Controller che definisce gli endpoint per accedere ai servizi di Ordini con le relative informazioni sul tipo di richiesta e sulla struttura del payload o dei parametri da fornire
OrderService	Incapsula la logica di business dei servizi relativi agli ordini
OrderEntity	Classe Wrapper per accedere alla persistenza degli ordini, è necessario inserirla in un repository di TypeORM affinché si possa accedere al database
OrderWineEntity	Entità della tabella di relazione M:M tra ordini e vini

/payment

Classe	Descrizione
PaymentModule	Modulo NestJS
PaymentController	Classe Controller che definisce gli endpoint per accedere ai servizi di Pagamento con le relative informazioni sul tipo di richiesta e sulla struttura del payload o dei parametri da fornire
PaymentService	Incapsula la logica di business dei servizi relativi ai pagamenti

/user

Classe	Descrizione
UserModule	Modulo NestJS
UserController	Classe Controller che definisce gli endpoint per accedere ai servizi sull'utente con le relative informazioni sul tipo di richiesta e sulla struttura del payload o dei parametri da fornire

Classe	Descrizione
UserService	Incapsula la logica di business dei servizi relativi agli utenti
UserEntity	Entità Utente
RoleEntity	Entità Ruolo

/wine

Classe	Descrizione
WineModule	Modulo NestJS
WineController	Classe Controller che definisce gli endpoint per accedere ai servizi di Vini con le relative informazioni sul tipo di richiesta e sulla struttura del payload o dei parametri da fornire
WineService	Incapsula la logica di business dei servizi relativi ai Vini
FilterWinesHelper	Classe Helper che partendo da un'interfaccia utile all'invio di una richiesta al Controller, trasforma il payload in una query utile a TypeORM
CountryEntity	Entità utile a TypeORM
RegionEntity	Entità utile a TypeORM
WineEntity	Entità utile a TypeORM
WinecolorEntity	Entità utile a TypeORM
WinedenomEntity	Entità utile a TypeORM
WinegrapeEntity	Entità utile a TypeORM
WinetypeEntity	Entità utile a TypeORM

/wine-winegrape

Classe	Descrizione
WineWinegrapeModule	Modulo NestJS
WineWinegrapeService	Operazioni CRUD sull'associazione Vino-Uva (Wine-Winegrape)
WineWinegrapeEntity	Entità per TypeORM

/winefamily

Classe	Descrizione
WinefamilyModule	Modulo NestJS
WinefamilyController	Classe Controller che definisce gli endpoint per accedere ai servizi di Winefamily con le relative informazioni sul tipo di richiesta e sulla struttura del payload o dei parametri da fornire
WinefamilyService	Operazioni CRUD sull'associazione Vino-Uva (Wine-Winegrape)
WinefamilyEntity	Entità per TypeORM

/winegrape

Classe	Descrizione
WinegrapeModule	Modulo NestJS
WinegrapeController	Classe Controller che definisce gli endpoint per accedere ai servizi di Winegrape con le relative informazioni sul tipo di richiesta e sulla struttura del payload o dei parametri da fornire
WinegrapeService	Operazioni CRUD su Winegrape
WinegrapeEntity	Entità per TypeORM

/winery

Classe	Descrizione
WineryModule	Modulo NestJS
WineryController	Classe Controller che definisce gli endpoint per accedere ai servizi di Winery con le relative informazioni sul tipo di richiesta e sulla struttura del payload o dei parametri da fornire
WineryService	Operazioni CRUD su Winery
WineryEntity	Entità per TypeORM

2.2 Client

L'implementazione del progetto client (directory */Project/client*) è fatta invece grazie a React, una libreria Javascript che permette di utilizzare notazioni jsx

per suddividere la UI in componenti riutilizzabili incapsulando la logica di business. Inoltre fornisce meccanismi di gestione dello stato dei vari componenti nonché meccanismi di gestione dello stato dell'applicazione (a livello globale).

Il progetto si presenta in questo modo nella directory *src*:

- **Components**
- **Consts**
- **Contexts**
- **Helpers**
- **Hooks**
- **Pages**
- **Providers**
- **Services**

Concettualmente abbiamo le **Pages** che richiamano i **Componenti** e i **Providers** per fornire i dati di livello globale e nient'altro, sono poi i componenti ad implementare la logica di business utilizzando gli **Hooks**.

Fuori dalla directory *src* troviamo file di configurazione nonché i provider di Web Vitals per fornire metriche riguardo le performance dell'applicazione come ad esempio il tempo necessario al caricamento della risorsa più grande, il ritardo dal caricamento della pagina alla pronta gestione del primo input…

Nel progetto client utilizzeremo molto i concetti di "Functional Programming" esportando funzioni piuttosto che classi per la logica di business.

Alcune note importanti: nel progetto client compare più volte il termine admin per riferirsi invece all'utente GESTORE individuato nei documenti antecedenti l'Object Design Document.

Di seguito i dettagli sui singoli packages:

/Components/Admin/Wines

Classe	Descrizione
Services	Operazioni CRUD sulle varie entità gestite dall'utente MANAGER (Gestore)
WineForm	Form base per creare un nuovo vino

Classe	Descrizione
AddWinegrapeDialog	Componente che gestisce un Dialog di aggiunta uva al vino che si sta creando

/Components/Auth

Classe	Descrizione
RequireAuth	Componente base che fa da wrapper ai componenti che possono essere utilizzati solo dall'utente GESTORE
SignInForm	Form base di autenticazione
SignUpForm	Form base di registrazione

/Components/Cart

Classe	Descrizione
CartItemQuantity	Componente per visualizzare e modificare la quantità di ogni vino diverso presente nel carrello
CartList	Componente che mostra in una lista i vini aggiunti al carrello dall'utente longato
CartTotal	Componente per visualizzare il prezzo totale del carrello
CheckoutDialog	Componente che visualizza un form di checkout in un dialog presentato all'utente

/Components/Catalog

Classe	Descrizione
AddToCart	Pulsante per aggiungere un vino al carrello
WineList	Mostra i vini del catalogo in una lista
WineListItem	Mostra un singolo vino

/Components/Catalog/Details

Classe	Descrizione
WineDetails	Mostra i dettagli di un singolo vino

/Components/Common

Classe	Descrizione
ConfirmationDialog	Componente Dialog di conferma per un'azione richiesta dall'utente
Copyright	Componente che mostra informazioni e link di copyright
ErrorAlert	Componente per la visualizzazione di un errore
Loader	Componente che mostra un leader quando è in atto una richiesta di rete
PageTitle	Componente che mostra il titolo di una singola pagina

/consts

Classe	Descrizione
PlatformRoles	Ruoli disponibili per la piattaforma

/Contexts

Classe	Descrizione
AuthContext	Definisce un contesto (simile ad un singleton) attraverso cui è possibile gestire autenticazione e autorizzazione nell'applicazione
CartContext	Definisce un contesto di interazione con il carrello

Helpers

Classe	Descrizione
AxiosHelper	Funzioni helpers che aiutano a creare richieste http
BearerToken	Funzioni che permettono di interagire con il Bearer Token per l'autorizzazione delle richieste
String	Funzioni helpers di manipolazione stringhe

/Hooks

Classe	Descrizione
useWineFeature	Permette di chiedere al backend le possibili caratteristiche dei vini (uva, azienda, famiglia…)

Classe	Descrizione
useAuth	Permette di interagire con il contesto di auth
useCart	Permette di interagire con il contesto del carrello
useWine	Permette di ottenere dal backend un singolo vino
useWines	Permette di ottenere dal backend una lista di vini opportunamente filtrati

/Pages/Admin

Classe	Descrizione
AdminPage	Pagina base da estendere per implementare i controlli sul ruolo dell'utente
AddWine	Pagina di aggiunta vino
AdminWineList	Pagina di visualizzazione catalogo per il Gestore
UpdateWine	Pagina di modifica vino

/Pages/Auth

Classe	Descrizione
SignIn	Pagina di autenticazione
SignUp	Pagina di registrazione

/Pages/Cart

Classe	Descrizione
Cart	Pagina del carrello

/Pages/Catalog

Classe	Descrizione
Catalog	Pagina di visualizzazione catalogo per l'utente
WineDetailsPage	Pagina di visualizzazione dettagli vino

/Pages

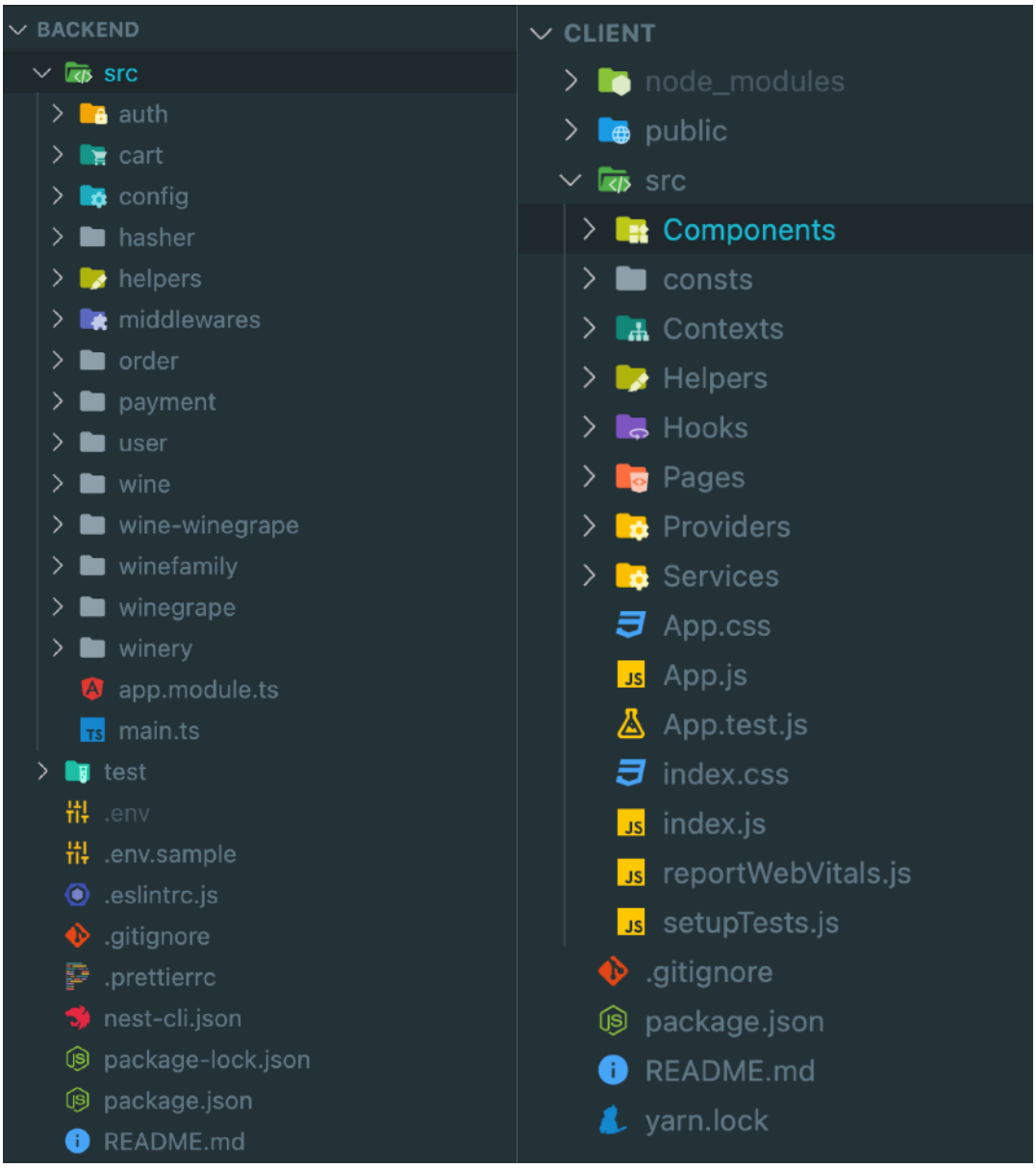
Classe	Descrizione
Main	Pagina Main che gestisce i percorsi di navigazione in base allo stato globale
ResponsiveAppBar	Wrapper di pagina che aggiunge una barra navigazionale

/Providers

Classe	Descrizione
AuthProvider	Componente che fornisce il contesto AuthContext ai componenti figli
CartProvider	Componente che fornisce il contesto CartContext ai componenti figli

/Services

Classe	Descrizione
Routes	Costanti e funzioni per determinare gli endpoint esposti dal backend per accedere ai suoi servizi



Backend Structure Client Structure

3. Class Interface

3.1 Backend

Auth Module

Nome Classe	AuthController
Descrizione	Gestisce l'interfaccia di accesso ai servizi per il client
Firma dei Metodi	+ login(Request req): JWTDto; + getProfile(Request req): UserPayloadDto; + createUser(CreateUserDto body): void;
Precondizioni	- - **getProfile** si aspetta una richiesta autorizzata da un access_token fornito da **login** -
Postcondizioni	- L'utente ha ottenuto un access_token - L'utente ha ottenuto il suo profilo completo - L'utente è ora registrato

Nome Classe	AuthService
Descrizione	Implementa la logica di business relativa ad autenticazione ed autorizzazione
Firma dei Metodi	+ async validatePassword(email: string, password: string): Promisse<User>; + login(user: User): JWTDto; + async register(user: CreateUserDto): void;
Precondizioni	- L'utente ha un account su Winer - L'utente ha un account su Winer - L'utente non ha un account su Winer
Postcondizioni	- Viene restituito l'utente se la password è valida - Viene restituito un access_token se l'autenticazione ha avuto successo - L'utente viene aggiunto al database

Nome Classe	JwtStrategy
Descrizione	Implementa la validazione di un json web token
Firma dei Metodi	+ constructor(JwtConfig); + async validate(payload: UserPayloadDto): Promise<UserPayloadDto>;

Nome Classe	JwtStrategy
Precondizioni	- - La richiesta ha superato la JwtGuard
Postcondizioni	- - Viene restituito l'utente al prossimo handler della catena oppure viene inviato un'UnauthorizedException

Nome Classe	Strategy
Descrizione	Interfaccia per implementare una strategia di autenticazione
Firma dei Metodi	+ constructor(StrategyConfig<T>); + async validate(…params): Promise<any>;
Precondizioni	- - La richiesta ha superato la relativa Guard
Postcondizioni	-

Cart Module

Nome Classe	CartController
Descrizione	Controller per gli endpoint di accesso ai servizi relativi al carrello
Firma dei Metodi	+ create(createCartDto: CreateCartDto, req: Request): void; + findAll(req: Request): Promise<CartItem[]>; + update(params: UpdateCartParams, updateCartDto: UpdateCartDto, req: Request): void; + remove(updateCartItemParams: UpdateCartItemParams, req: Request): void;
Precondizioni	Class level: Il consumatore del servizio deve essere autenticato
Postcondizioni	- Viene aggiunto un elemento al carrello - Viene restituita una lista di elementi presenti nel carrello - Viene aggiornato l'elemento del carrello scelto - Viene rimosso l'elemento dal carrello

Nome Classe	CartService
Descrizione	Implementa la logica di business relativa al carrello

Nome Classe	CartService
Firma dei Metodi	+ async create(createCartDto: CreateCartDto, userID: number): Promise<void>; + async findAll(userID: number): Promise<CartItem[]>; + async update(wine: string, vintage: number, updateCartDto: UpdateCartDto): Promise<void>; + async remove(wine: string, vintage: number, userID: number): Promise<void>; + async empty(userID: number): Promise<void>;
Precondizioni	-
Postcondizioni	- Viene aggiunto un elemento al carrello - Viene restituita una lista di elementi presenti nel carrello - Viene aggiornato l'elemento del carrello scelto - Viene rimosso un elemento dal carrello - Viene svuotato il carrello dell'utente

Hasher Module

Nome Classe	HasherService
Descrizione	Implementa la logica di business riguardo l'hash delle password
Firma dei Metodi	+ async compare(plainText: string, hash: string): Promise<boolean>; + async hash(plainText: string): Promise<string>;
Precondizioni	-
Postcondizioni	- Viene restituito true se gli hash corrispondono, altrimenti false

Helpers Module

Nome Classe	PaginationService
Descrizione	Implementa la paginazione dell'entità scelta
Firma dei Metodi	+ async paginate<T>(page: number, repository: Repository<T>, options: FindManyOptions<T>): Promise<PaginationResponse<T>>;
Precondizioni	- T è un'entità TypeORM
Postcondizioni	- Vengono restituiti gli elementi della pagina *page* dell'entità *T*. Il numero massimo di elementi per pagina è costante e definito nella classe stessa.

Middlewares

Nome Classe	LoggerMiddleware
Descrizione	Implementa l'interfaccia middleware ed intercetta le richieste per salvare dati relativi alla loro elaborazione e inviarli ad un file (default —> standard output)
Firma dei Metodi	+ use(req: Request, res: Response, next: NextFunction): void;
Precondizioni	- La classe viene registrata globalmente come middleware per gli endpoint scelti
Postcondizioni	- Vengono stampati i dati riguardo il ciclo di vita della richiesta

Di seguito troviamo l'interfaccia utilizzata per creare le risorse con le relative operazioni CRUD necessarie ai servizi, durante lo sviluppo si incoraggia l'utilizzo della Command Line Interface fornita dal framework NestJS per creare facilmente risorse REST con i metodi da implementare.

Nome Classe	RestResource
Descrizione	Definisce i metodi che controller e service devono implementare per fornire le funzionalità base di una risorsa con *Persistence capabilities*
Firma dei Metodi	+ create<T>(createBody: CreateBody<T>): void; + findAll<T>(filterOptions?: FilterOptions<T>, page?: number): PaginationResponse | T[]; + findOne<T>(id: number | PrimaryKey<T>): T; + update<T>(id: number | PrimaryKey<T>, updateBody: UpdateBody<T>): void; + remove<T>(id: number | PrimaryKey<T>): void;
Precondizioni	
Postcondizioni	- Viene creato un nuovo record con i parametri contenuti in *createBody* - Vengono restituiti i record paginati o grezzi - Viene restituito il record che porta l'ID passato come parametro - Viene aggiornato il record con l'id scelto modificando solo i parametri passati in *updateBody* - Viene rimosso il record che ha id specificato dal parametro passato in input

Payment Module

Nome Classe	PaymentController
Descrizione	Controller per l'accesso ai servizi di pagamento (mock)
Firma dei Metodi	+ async completeOrder(req: Request, body: PaymentDto): Promise<Order>;
Precondizioni	- L'utente è autenticato

Nome Classe	PaymentController
Postcondizioni	- Il carrello dell'utente è vuoto e l'ordine da lui effettuato è creato correttamente con i vini del carrello

Nome Classe	PaymentService
Descrizione	Logica di business dei pagamenti
Firma dei Metodi	+ async completeOrder(userID: number, address: string): Promise<Order>;
Precondizioni	-
Postcondizioni	- L'ordine viene creato nel database - Il carrello dell'utente viene svuotato al completamento dell'ordine

Per la documentazione sui metodi delle entità si fa riferimento alla documentazione di TypeORM.

3.2 Client

Per quanto riguarda il progetto client ci limiteremo a documentare l'interfaccia degli Hooks e dei Providers, in quanto sono gli unici ad avere logica, mentre i componenti li utilizzano esclusivamente per generare contenuti da inserire nella UI.

Providers

Nome Classe	AuthProvider
Descrizione	Fornisce accesso globale all'autenticazione nell'app
Firma dei Metodi	+ async signIn(email: string, password: string, callback: (error?: Error, user?: UserPayloadDto) => void): Promise<string>; + async signUp(email: string, password: string, callback: (error?: Error) => void) : Promise<string>; + signOut(callback: (error?: Error) => void): void;
Precondizioni	1. L'utente è un GUEST 2. L'utente è un GUEST 3. L'utente ha effettuato precedentemente l'operazione di signIn

Nome Classe	AuthProvider
Postcondizioni	1. L'utente ottiene un access_token che gli permette di accedere alle pagine protette (Carrello e Checkout) 2. L'utente ottiene un account Winer 3. Viene invalidato l'access_token dell'utente

Nome Classe	CartProvider
Descrizione	Fornisce accesso globale al carrello nell'app
Firma dei Metodi	+ async add(wine: string, vintage: number, callback: (error?: Error, cartItem: CartItem) => void): Promise<void>; + async update(wine: string, vintage: number, callback: (error?: Error) => void): Promise<void>; + async remove(wine: string, vintage: number, callback: (error?: Error) => void): Promise<void>; + async getCart(): Promise<CartItem[]>; + async buy(creditCardNumber, cvc, callback: (error?: Error, order?: Order) => void): Promise<void>; + getTotalPrice(): number;
Precondizioni	1. - 2. Il vino in questione esiste nel carrello 3. Il vino in questione esiste nel carrello 4. - 5. Il carrello non è vuoto 6. -
Postcondizioni	1. Il carrello contiene ora il vino richiesto 2. Il carrello è stato aggiornato 3. Il carrello non contiene più il vino richiesto 4. Si ottiene una lista di vini aggiunti precedentemente al carrello 5. Viene completata una transazione (mock) e viene creato l'ordine 6. Viene restituito il prezzo totale degli elementi presenti nel carrello

Hooks

Gli Hooks sono stati introdotti nella versione 17 di React e introducono un nuovo modo di gestire lo stato dei componenti nella nostra applicazione utilizzando i paradigmi della programmazione funzionale.

Nome Classe	Hooks
Descrizione	Forniscono dati e meccanismi di interazione con lo stato del componente che utilizza un look

Nome Classe	**Hooks**
Firma dei Metodi	+ useWineFeature(endpoint: string): void; + useAuth(): void; + useCart(): void; + useWine(wine: string, vintage: number, onError: () => void): void; + useWines(onError: () => void): void;
Precondizioni	1. - 2. - 3. - 4. - 5. - 6. -
Postcondizioni	1. Si ottiene una lista di entità in base all'endpoint passato come parametro, le entità supportate sono tutte le risorse rest di cui si è parlato nel **paragrafo 3.1** 2. Viene restituito un riferimento al Singleton AuthProvider 3. Viene restituito un riferimento al Singleton CartProvider 4. Si ottiene un oggetto WineEntity che ha come ID la combinazione (wine, vintage) passata come parametro 5. Si ottiene una lista di vini e un array di filtri che è possibile applicare, la lista di vini viene aggiornata ad ogni cambiamento sui filtri