Functional Specifications Document - E-commerce Site for Wine Barrels

Version: 1.2
Date: 08/27/2025

1. Project Objective

To create a showcase and e-commerce website, entirely in English, for the company "Millésime Sans Frontières" (Timeless Vintages Without Borders). The site's purpose is to present a catalog of wine barrels (and potentially other spirits) from around the world and to enable their international sale and shipment. The business will primarily focus on professionals (B2B).

2. Primary Targets

- **Professional Clients (B2B - Core Target):** Winemakers, brewers, distillers, wine merchants, restaurateurs, and decorators seeking barrels for aging beverages or for decorative use.
- **Individual Customers (B2C):** Hobbyists and enthusiasts for decoration or micro-vinification projects.

3. Site Map (Page Structure)

- **Homepage:**
    - High-impact visuals: photos of cellars, barrels from different origins.
    - Tagline: "Quality Wine Barrels from Around the World".
    - Highlighting selections: "New Arrivals," "French Origin," "Ex-Bourbon Casks."
    - Clear call to action (e.g., "Explore Our Barrels," "Professional Inquiries").
- **Our Barrels / Catalog:**
    - Listing page for all products.
    - **Essential Filters:**
        - By **Country of Origin** (France, USA, Spain, Italy, etc.).
        - By **Previous Contents** (Red Wine, White Wine, Bourbon, Sherry, Port, etc.).
        - By **Volume** (in liters: 225L, 300L, 500L...).
        - By **Wood Type** (French Oak, American Oak, Acacia...).
        - By Condition (New, Refurbished, Number of uses).
    - Sorting option (by price, by origin, by volume).
- **Product Page:**
    - Clear title: e.g., "French Oak Barrel - Ex-Red Wine - 225L".
    - Multiple high-quality photos (barrel from several angles, interior, details).

    - Detailed description: history of the barrel, potential aromatic influence on contents.
    - Technical Information: Origin, Wood, Volume, Previous Contents, Condition, Dimensions, Weight.
    - Price.
    - "Add to Cart" or "Request a Quote" button for large quantities.

- **About Us / Our Story:**
    - Presentation of expertise in barrel selection and logistics.
    - Highlighting partnerships with wineries and distilleries.

- **B2B / Professional Services:**
    - Page explaining services for professionals: custom sourcing, international logistics, bulk offers.
    - Detailed contact/quote request form.
    - Login access for tiered pricing and a specific catalog.

- **Contact Us:**
    - Contact form.
    - Warehouse address (if relevant), email, phone number.
    - FAQ on logistics, barrel maintenance, etc.
    - Customer support options (email, phone, chat).

- **Legal Pages (in footer):**
    - Terms & Conditions.
    - Privacy Policy.
    - Shipping & Returns.

4. Key Features

- **Catalog and Inventory Management:** Admin interface to manage barrels, their precise stock (unique pieces or lots), and their status.
- **B2B/B2C E-commerce:**
    - Management of differentiated pricing (public / professional).
    - Complex shipping cost calculation based on weight/volume/destination, including management of incoterms.
    - Ability to generate shipping documents (proforma invoices, packing lists).
    - Real-time shipment tracking.
    - Quote request system integrated into the site with quote tracking.
    - Secure online payment (Stripe, PayPal) and bank transfer for professionals.
- **Order Management:** Admin interface to track orders, manage statuses (pending, shipped, delivered), and generate invoices.

- **Customer Management:** Ability to view customer information (B2B and B2C) and their order history.
- **Professional Client Area:** Each professional can create an account to access their pricing, place orders, and track their deliveries.
- **Responsive Design:** The site must be perfectly usable on all devices.
- **Search Engine Optimization (SEO):** Structure optimized to be found via searches like "buy used wine barrels," "french oak casks for sale."

5. Non-Functional Requirements

- **Performance:** The site must load quickly, even with many high-quality images and an extensive catalog.
- **Security:** SSL certificate (HTTPS), protection of customer data and transactions, GDPR compliance.
- **Scalability:** The site must be able to handle an increase in traffic and catalog size without a major redesign, and allow for the addition of new features.
- **Maintenance:** Ease of updating content and features with an intuitive content management system (CMS).
- **Traffic Analysis:** Integration of analytics tools (e.g., Google Analytics) to monitor site performance, user behavior, and traffic sources.

6. Content to be Provided

- **Texts:** All texts must be written in professional English.
- **Photos/Videos:** Very high-quality visuals are essential for a barrel website (photos from multiple angles, details, in-context shots).
- **Product Sheets:** Complete information for each barrel in the catalog (origin, wood, volume, previous contents, condition, dimensions, weight).

7. Next Steps

1. **Validation:** Discuss and validate this finalized specifications document.
2. **Mock-up (Design):** Create a visual mock-up for the site's appearance.
3. **Development:** Choose the technology (e.g., Shopify, WordPress/WooCommerce, or custom development) and begin construction.

Frontend Specifications - Millésime Sans Frontières

Version: 1.0
Date: 08/27/2025

1. Objective

This document describes the technical and functional specifications for the frontend of the "Millésime Sans Frontières" wine barrel e-commerce site. The frontend will be the main user interface, allowing customers to browse the catalog, place orders, and interact with the company's services.

2. Key Technologies

- **JavaScript Framework:** Nuxt.js (recommended)
  - **Justification for choosing Nuxt.js over pure Vue.js:**
    - **SSR (Server-Side Rendering) / Static Site Generation:** Essential for the SEO of an e-commerce site, allowing search engines to easily index content.
    - **Automatic Routing:** Simplifies route management based on the file structure.
    - **State Management (Vuex):** Facilitated integration for centralized data management.
    - **Performance Optimizations:** Fast page loading and a better user experience.
    - **Full-Stack Development:** Ability to add server-side features if necessary.
- **Language:** JavaScript (with the possibility of TypeScript for better robustness if desired later).
- **Package Manager:** npm or Yarn.
- **Styling:** CSS (with a preprocessor like Sass/SCSS) or a CSS framework (e.g., Tailwind CSS, Bootstrap) for rapid development and responsive design.

3. Design and UX (User Experience) Principles

- **Modern and Clean Design:** A visually appealing, intuitive, and easy-to-use interface.
- **Fully Responsive (Mobile-First):** The site must adapt perfectly to all screen sizes (mobiles, tablets, desktops) with a "mobile-first" design approach.
- **Intuitive Navigation:** Clear menus, logical navigation paths, effective search and filters.
- **Performance:** Fast loading times, smooth animations.
- **Accessibility:** Compliance with web accessibility standards (WCAG) to ensure use by everyone.
- **Visual Consistency:** Consistent use of colors, typography, icons, and UI components throughout the entire site.

4. Key Frontend Features

- **Static Pages:**
  - Homepage: Presentation, featured products, calls to action.
  - About Us: History, mission, team.
  - Contact: Contact form, contact details.
  - Legal Pages: Terms of Service, Privacy Policy, Shipping & Returns.
- **Product Catalog (Barrels):**
  - Display of barrels with images, names, prices.
  - Advanced search and filtering features (by country of origin, previous content, volume, wood type, condition).
  - Pagination and sorting of results.
- **Product Detail Page:**
  - Full display of barrel information (description, technical specifications, multiple images).

    - "Add to Cart" or "Request a Quote" button.
- **Shopping Cart:**
    - Display of added items, quantities, prices.
    - Ability to modify quantities or remove items.
    - Subtotal calculation.
- **Checkout Process:**
    - Clear and secure multi-step process (shipping information, billing, payment).
    - Real-time form validation.
    - Integration with the backend's payment gateway.
- **Authentication and User Profile:**
    - Registration and login forms.
    - User profile page: management of personal information, addresses.
    - Order history with details.
    - Order tracking.
- **Professional Area (B2B):**
    - Detailed quote request form.
    - Tracking of submitted quotes.
    - Access to specific information (B2B pricing, documents).

5. Integration with the Backend (RESTful API)

- The frontend will communicate with the FastAPI backend via HTTP requests (GET, POST, PUT, DELETE).
- Use of libraries like Axios or the native `fetch` API for requests.
- Management of JWT tokens for request authentication.
- API error handling and displaying relevant messages to the user.

6. Deployment

- The frontend will be containerized with Docker for easy deployment via Docker Compose, alongside the backend and the database.

7. Next Steps

- Creation of mockups and wireframes for key pages.
- Definition of the graphic charter (colors, typography, logo).
- Breakdown of UI/UX components.
- Setting up the frontend development environment.

Development Environment Setup - Millésime Sans Frontières Backend

Version: 1.0
Date: 08/27/2025

1. Purpose

This document provides step-by-step instructions for setting up the local development environment for the Millésime Sans Frontières backend. It covers installing prerequisites, configuring the database, and launching the FastAPI application.

2. Prerequisites

Before you begin, ensure you have the following tools installed on your machine:

- **Python 3.9+:** Download from python.org.
- **pip:** Usually included with Python.
- **Docker Desktop:** For managing the PostgreSQL database and other services (e.g., Redis for Celery).
    - Download from docker.com.
- **Git:** For cloning the project repository.
    - Download from git-scm.com.
- **A code editor:** Visual Studio Code (VS Code) is recommended.
    - Download from code.visualstudio.com.

3. Project Setup

1. **Clone the Git repository:**
   Open your terminal or command prompt and run:

   ```
   git clone [GIT_REPOSITORY_URL]
   cd millesime-sans-frontieres-backend # or the project folder name
   ```

2. **Create and activate a virtual environment:**
   It is highly recommended to use a virtual environment to isolate project dependencies.

   ```
   python -m venv venv
   # On Windows
   .\venv\Scripts\activate
   # On macOS/Linux
   source venv/bin/activate
   ```

3. **Install Python dependencies:**
   Once the virtual environment is activated, install all required libraries:

   ```
   pip install -r requirements.txt
   ```

   (Note: The `requirements.txt` file will be created during the initial project development.)

4. Database Configuration (PostgreSQL with Docker)

1. **Start Docker Desktop:** Ensure Docker Desktop is running on your machine.

2. **Start the PostgreSQL database:**
   The project will include a `docker-compose.yml` file to facilitate starting the necessary services (database, Redis).

   ```
   docker-compose up -d postgres redis # or simply 'docker-compose up -c
   ```

   (Note: The exact services will depend on the final `docker-compose.yml`.)

3. **Run database migrations:**
   Once the database is started, apply the migrations to create the database schema.

   ```
   alembic upgrade head
   ```

   (Note: `alembic` will be installed via `requirements.txt`.)

5. Launching the Backend Application

1. **Set environment variables:**
   Create a `.env` file at the project root and define the necessary environment variables (e.g., `DATABASE_URL`, `SECRET_KEY`). An `.env.example` file will be provided as a guide.

   ```
   # Example content for .env
   DATABASE_URL="postgresql://user:password@localhost:5432/dbname"
   SECRET_KEY="your_very_long_and_complex_jwt_secret_key"
   # ... other variables
   ```

2. **Start the FastAPI application:**

   ```
   uvicorn app.main:app --reload --host 0.0.0.0 --port 8000
   ```

   (Note: The `app.main:app` path may vary depending on the final project structure.)

   The API will be accessible at `http://localhost:8000`.

3. **Access the API documentation:**
   - **Swagger UI:** `http://localhost:8000/docs`
   - **ReDoc:** `http://localhost:8000/redoc`

6. Running Tests

- To run unit and integration tests:

```
pytest
```

7. Common Development Commands

- **Linter (e.g., Ruff):**

```
ruff check .
```

- **Code formatter (e.g., Black):**

```
black .
```

- **Stop Docker services:**

```
docker-compose down
```

Detailed Database Schema Design - Millésime Sans Frontières

Version: 1.1
Date: 08/27/2025

```
erDiagram
    Users ||--o{ Addresses : has
    Users ||--o{ Orders : places
    Users ||--o{ Quotes : requests
    Orders ||--o{ OrderItems : contains
    Barrels ||--o{ OrderItems : includes
    Orders ||--|{ Addresses : "shipping_address_id"
    Orders ||--o{ Addresses : "billing_address_id"
```

1. Objective

This document details the database schema for the backend of the Millésime Sans Frontières application. It defines the structure of each table, the data types of the columns, the constraints (primary keys, foreign keys, uniqueness, non-nullability), and the relationships between the tables.

2. Chosen Database

- **Type:** PostgreSQL (confirmed)

3. Naming Conventions

- **Tables:** Plural nouns, in PascalCase (e.g., `Users`, `Barrels`).
- **Columns:** Names in snake_case (e.g., `first_name`, `created_at`).
- **Primary Keys:** `id` (UUID).
- **Foreign Keys:** `related_table_id` (e.g., `user_id`).

4. Table Definitions

Table: `Users`

Description: Stores information about users (B2C clients, B2B clients, administrators).

Column Name	Data Type	Constraints	Description
`id`	`UUID`	`PRIMARY KEY`, `DEFAULT gen_random_uuid()`	Unique identifier for the user
`email`	`VARCHAR(255)`	`NOT NULL`, `UNIQUE`	User's email address (login)
`password_hash`	`VARCHAR(255)`	`NOT NULL`	Hash of the user's password
`role`	`VARCHAR(50)`	`NOT NULL`, `DEFAULT 'b2c'`	User's role (`admin`, `b2b`, `b2c`)
`first_name`	`VARCHAR(100)`	`NULLABLE`	User's first name
`last_name`	`VARCHAR(100)`	`NULLABLE`	User's last name
`company_name`	`VARCHAR(255)`	`NULLABLE`	Company name (for B2B clients)
`phone_number`	`VARCHAR(50)`	`NULLABLE`	Phone number
`is_active`	`BOOLEAN`	`NOT NULL`, `DEFAULT TRUE`	Indicates if the account is active
`created_at`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT NOW()`	Date and time of user creation
`updated_at`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT NOW()`	Date and time of the last update

Table: `Addresses`

Description: Stores shipping and billing addresses.

Column Name	**Data Type**	**Constraints**	**Description**
`id`	`UUID`	`PRIMARY KEY`, `DEFAULT gen_random_uuid()`	Unique identifier for the address
`user_id`	`UUID`	`NOT NULL`, `FOREIGN KEY (Users.id)`	User to whom the address is associated
`address_line1`	`VARCHAR(255)`	`NOT NULL`	First line of the address
`address_line2`	`VARCHAR(255)`	`NULLABLE`	Second line of the address (optional)
`city`	`VARCHAR(100)`	`NOT NULL`	City
`state_province`	`VARCHAR(100)`	`NULLABLE`	State/Province (if applicable)
`zip_code`	`VARCHAR(20)`	`NOT NULL`	Postal code
`country`	`VARCHAR(100)`	`NOT NULL`	Country
`is_default`	`BOOLEAN`	`NOT NULL`, `DEFAULT FALSE`	Indicates if this is the default address
`created_at`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT NOW()`	Date and time of address creation
`updated_at`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT NOW()`	Date and time of the last update

Table: `Barrels`

Description: Stores detailed information about the barrels available for sale.

Column Name	**Data Type**	**Constraints**	**Description**
`id`	`UUID`	`PRIMARY KEY`, `DEFAULT gen_random_uuid()`	Unique identifier for the barrel
`name`	`VARCHAR(255)`	`NOT NULL`	Name of the barrel (e.g., "Ex-Red Wine French Oak Barrel")
`origin_country`	`VARCHAR(100)`	`NOT NULL`	Barrel's country of origin

Column Name	Data Type	Constraints	Description
`previous_content`	`VARCHAR(100)`	`NOT NULL`	Previous content of the barrel (e.g., "Red Wine", "Bourbon")
`volume_liters`	`NUMERIC(10, 2)`	`NOT NULL`	Volume of the barrel in liters
`wood_type`	`VARCHAR(100)`	`NOT NULL`	Type of wood (e.g., "French Oak", "American Oak")
`condition`	`VARCHAR(50)`	`NOT NULL`	Condition of the barrel (e.g., "New", "Refurbished", "Used")
`price`	`NUMERIC(10, 2)`	`NOT NULL`	Unit price of the barrel
`stock_quantity`	`INTEGER`	`NOT NULL, DEFAULT 0`	Quantity in stock
`description`	`TEXT`	`NULLABLE`	Detailed description of the barrel
`image_urls`	`TEXT[]`	`NULLABLE`	Array of URLs for the barrel's images
`dimensions`	`VARCHAR(255)`	`NULLABLE`	Physical dimensions of the barrel
`weight_kg`	`NUMERIC(10, 2)`	`NULLABLE`	Weight of the barrel in kg
`created_at`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL, DEFAULT NOW()`	Date and time the barrel was added
`updated_at`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL, DEFAULT NOW()`	Date and time of the last update

Table: `Orders`

Description: Records orders placed by users.

Column Name	Data Type	Constraints	Description
`id`	`UUID`	`PRIMARY KEY, DEFAULT gen_random_uuid()`	Unique identifier for the order
`user_id`	`UUID`	`NOT NULL, FOREIGN KEY (Users.id)`	User who placed the order

Column Name	Data Type	Constraints	Description
order_date	TIMESTAMP WITH TIME ZONE	NOT NULL, DEFAULT NOW()	Date and time of the order
total_amount	NUMERIC(10, 2)	NOT NULL	Total amount of the order
status	VARCHAR(50)	NOT NULL, DEFAULT 'pending'	Status of the order (e.g., pending, processing, shipped, delivered, cancelled)
shipping_address_id	UUID	NOT NULL, FOREIGN KEY (Addresses.id)	Shipping address for the order
billing_address_id	UUID	NULLABLE, FOREIGN KEY (Addresses.id)	Billing address (if different)
payment_status	VARCHAR(50)	NOT NULL, DEFAULT 'pending'	Payment status (e.g., pending, paid, refunded)
payment_method	VARCHAR(50)	NULLABLE	Payment method used
tracking_number	VARCHAR(255)	NULLABLE	Shipping tracking number
notes	TEXT	NULLABLE	Internal notes about the order
updated_at	TIMESTAMP WITH TIME ZONE	NOT NULL, DEFAULT NOW()	Date and time of the last update

---

Table: `OrderItems`

Description: Details the items included in each order.

Column Name	Data Type	Constraints	Description
id	UUID	PRIMARY KEY, DEFAULT gen_random_uuid()	Unique identifier for the order item
order_id	UUID	NOT NULL, FOREIGN KEY (Orders.id)	Order to which the item belongs

Column Name	Data Type	Constraints	Description
`barrel_id`	`UUID`	`NOT NULL`, `FOREIGN KEY (Barrels.id)`	Ordered barrel
`quantity`	`INTEGER`	`NOT NULL`, `DEFAULT 1`	Quantity of the ordered barrel
`price_at_purchase`	`NUMERIC(10, 2)`	`NOT NULL`	Price of the barrel at the time of purchase

Table: `Quotes`

Description: Records quote requests from B2B clients.

Column Name	Data Type	Constraints	Description
`id`	`UUID`	`PRIMARY KEY`, `DEFAULT gen_random_uuid()`	Unique identifier for the quote
`user_id`	`UUID`	`NOT NULL`, `FOREIGN KEY (Users.id)`	User who requested the quote
`request_date`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT NOW()`	Date and time of the quote request
`status`	`VARCHAR(50)`	`NOT NULL`, `DEFAULT 'pending'`	Status of the quote (e.g., `pending`, `approved`, `rejected`, `converted_to_order`)
`requested_items`	`JSONB`	`NOT NULL`	Details of requested barrels and quantities (JSON)
`quoted_price`	`NUMERIC(10, 2)`	`NULLABLE`	Price proposed in the quote
`notes`	`TEXT`	`NULLABLE`	Internal notes or comments on the quote
`valid_until`	`TIMESTAMP WITH TIME ZONE`	`NULLABLE`	Expiration date of the quote
`updated_at`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT NOW()`	Date and time of the last update

---

5. Relationships between Tables (Foreign Keys)

- `Users` 1:N `Addresses` (a user can have multiple addresses)
- `Users` 1:N `Orders` (a user can place multiple orders)
- `Users` 1:N `Quotes` (a user can request multiple quotes)
- `Orders` 1:N `OrderItems` (an order contains multiple items)
- `Barrels` 1:N `OrderItems` (a barrel can be in multiple order items)
- `Orders` 1:1 `Addresses` (an order has one shipping address and potentially one billing address)

6. Indexes (for performance optimization)

- `Users` : Index on `email` (for login lookups).
- `Orders` : Index on `user_id` and `order_date` (for order history searches).
- `Barrels` : Index on `origin_country` , `previous_content` , `volume_liters` , `wood_type` (for search filters).
- `Addresses` : Index on `user_id` .
- `Quotes` : Index on `user_id` and `request_date` .

Backend Specifications - Vintage Without Borders

Version: 1.3
Date: 08/27/2025

Note on the Level of Detail: This document provides an overview of the backend specifications. Each section can be further elaborated and detailed in subsequent documents or stages (e.g., detailed API specifications, complete database schema).

1. Objective

This document describes the technical and functional specifications of the backend for the "Vintage Without Borders" e-commerce site for wine barrels. The backend will be responsible for data management, business logic, authentication, and exposing the necessary APIs for the frontend to operate.

2. Key Technologies

- **Programming Language:** Python
- **Web API Framework:** FastAPI
  - *Advantages:* High performance, automatic data validation (Pydantic), interactive documentation (Swagger UI/ReDoc).
- **Database:** PostgreSQL (recommended for its robustness and advanced features)
  - *Simple alternative for initial development:* SQLite (for quick setup)
- **ORM (Object-Relational Mapper):** SQLAlchemy (with Alembic for database migrations)
- **Authentication:** JWT (JSON Web Tokens)

- **Containerization:** Docker (for easier deployment and management)

3. Main Backend Responsibilities

The backend will manage the following functionalities via RESTful APIs:

- **Product Management (Barrels):**
    - Create, Read, Update, Delete (CRUD) for barrels.
    - Inventory management (available quantities, status).
    - Search and filtering of barrels.
- **User Management:**
    - User registration and login (B2C and B2B).
    - User profile management (addresses, contact information).
    - Role and permission management (administrator, B2B client, B2C client).
- **Order Management:**
    - Creation and tracking of orders.
    - Updating order status.
    - Order history per user.
- **Quote Management:**
    - Creation, tracking, and management of B2B quote requests.
    - Generation of custom quotes.
- **Payment Integration:**
    - Interface with secure payment gateways (e.g., Stripe, PayPal).
    - Management of transactions and refunds.
- **Logistics and Shipping:**
    - Potential integration with carrier APIs for fee calculation and tracking.
    - Generation of shipping documents.
- **Content Management (Lightweight CMS):**
    - API for managing static pages (About Us, Contact, FAQ) if not handled by an external CMS.

4. Data Model (High Level)

- **`Users`**: `id`, `email`, `password_hash`, `role` (`admin`, `b2b`, `b2c`), `first_name`, `last_name`, `company_name` (for B2B), `address_id`.
- **`Addresses`**: `id`, `street`, `city`, `zip_code`, `country`.
- **`Barrels`**: `id`, `name`, `origin_country`, `previous_content`, `volume_liters`, `wood_type`, `condition`, `price`, `stock_quantity`, `description`, `image_urls`.
- **`Orders`**: `id`, `user_id`, `order_date`, `total_amount`, `status` (`pending`, `processing`, `shipped`, `delivered`, `cancelled`), `shipping_address_id`.
- **`OrderItems`**: `id`, `order_id`, `barrel_id`, `quantity`, `price_at_purchase`.
- **`Quotes`**: `id`, `user_id`, `request_date`, `status` (`pending`, `approved`, `rejected`), `requested_items` (JSON), `quoted_price`, `notes`.

5. Security and Authentication

- **Authentication:** Based on JWT (JSON Web Tokens) to secure API access.
- **Authorization:** Implementation of roles to control access to different resources (e.g., only administrators can create/modify barrels).
- **Data Validation:** Use of Pydantic for automatic validation of incoming requests.
- **Protection:** Against common attacks (SQL injection, XSS, CSRF) via FastAPI's built-in features and best practices.
- **Data Encryption:** Sensitive data (passwords, payment information) will be encrypted at rest and in transit.
- **Rate Limiting:** Implementation of mechanisms to prevent abuse and Denial of Service (DoS) attacks on API endpoints.

6. API Design Principles

- **RESTful:** Use of standard HTTP methods (GET, POST, PUT, DELETE) and appropriate status codes.
- **Clarity and Consistency:** Clear and consistent resource names.
- **Documentation:** Automatic generation of API documentation (Swagger UI/ReDoc) to facilitate frontend integration.
- **Error Handling:** Clear and informative error responses.
- **API Versioning:** Use of a versioning scheme (e.g., `/v1/barrels`) to allow the API to evolve without breaking compatibility with existing clients.

7. Deployment

- The backend will be containerized with Docker to ensure portability and reproducibility of the environment.
- Deployment is planned on a cloud service (e.g., AWS, Google Cloud, Azure) or a VPS.

8. Compliance and Regulation

- **GDPR (General Data Protection Regulation):** The backend will be designed in compliance with GDPR principles, including:
  - **Data Minimization:** Collecting only necessary data.
  - **Rights of the Data Subject:** Implementing mechanisms to allow users to exercise their rights (access, rectification, erasure, data portability).
  - **Data Security:** Appropriate technical and organizational measures to protect personal data.
  - **Consent:** Management of consent for data collection and processing.
- **Other Regulations:** Consideration of specific regulations related to international trade and the sale of alcohol-related products (if applicable to the sale of barrels).

9. Detailed Technical Aspects

- **Logging and Monitoring:**

    - Implementation of a structured logging system (e.g., JSON logs) to facilitate analysis.
    - Definition of log levels (DEBUG, INFO, WARNING, ERROR, CRITICAL).
    - Integration with monitoring tools (e.g., Prometheus, Grafana) for performance and error monitoring in production.
- **Error Management:**
    - Definition of custom exceptions for specific business errors.
    - Implementation of a global error handler to standardize API error responses.
    - Logging of critical errors.
- **Testing Strategy:**
    - **Unit Tests:** To validate the correct functioning of individual functions and classes.
    - **Integration Tests:** To verify the interaction between different components (e.g., API and database).
    - **End-to-End (E2E) Tests:** To simulate complete user scenarios (potentially with a testing framework like Playwright or Selenium if the frontend is integrated).
- **Configuration Management:**
    - Use of environment variables for sensitive configurations (API keys, database credentials).
    - Loading configurations via libraries like `python-dotenv` or `Dynaconf` .
- **Background Tasks:**
    - Use of a task queue system (e.g., Celery with Redis or RabbitMQ) for long-running or asynchronous operations (sending confirmation emails, processing complex orders, generating reports).
- **Caching:**
    - Implementation of a caching mechanism (e.g., Redis) for frequently accessed data to improve API performance and reduce the load on the database.
- **Reporting and Data Analysis:**
    - **Web Analytics:** Integration with web analytics tools (e.g., Google Analytics) to track user behavior on the site (page views, clicks, user journeys, conversion rates).
    - **Business Intelligence (BI) Reports:**
        - Generation of reports from backend data (sales, inventory, customers, quotes, product performance).
        - Ability to create custom dashboards for administrators.
        - Exporting data for further analysis (e.g., CSV, Excel).

10. Next Steps

- Detailed design of the database schema.
- Precise definition of each API endpoint (URL, HTTP methods, parameters, responses).
- Setup of the development environment.

API Endpoint Definitions - Millésime Sans Frontières Backend

Version: 1.0

Date: 08/27/2025

1. Purpose

This document details the RESTful API endpoints exposed by the Millésime Sans Frontières backend. It specifies the HTTP methods, paths, query parameters, request bodies, and response structures for each feature.

2. General Principles

- **API Versioning:** All endpoints will be prefixed with `/v1/` (e.g., `/v1/barrels`).
- **Authentication:** Endpoints requiring authentication need a valid JWT in the `Authorization: Bearer <token>` header.
- **Error Handling:** Errors will be returned with an appropriate HTTP status code and a standardized JSON response body, generally in the format `{"detail": "Error message"}` .
- **Data Validation:** All incoming data will be validated by Pydantic. Validation errors will return a `422 Unprocessable Entity` status.

3. Endpoints by Resource

Resource: Authentication and Users (`/v1/auth` , `/v1/users`)

Endpoint: Register a new user

- **Method:** `POST`
- **Path:** `/v1/auth/register`
- **Description:** Creates a new user account (B2C or B2B).
- **Authentication Required:** No
- **Request Body:**

```
{
  "email": "user@example.com",
  "password": "StrongPassword123",
  "first_name": "John",
  "last_name": "Doe",
  "role": "b2c" // or "b2b"
}
```

- **Response (Success - 201 Created):**

```
{
  "id": "user-uuid",
  "email": "user@example.com",
  "role": "b2c"
}
```

- **Response (Error - 400 Bad Request, 422 Unprocessable Entity):**

```
{"detail": "Email already registered"}
```

Endpoint: User login

- **Method:** `POST`
- **Path:** `/v1/auth/login`
- **Description:** Authenticates the user and returns a JWT.
- **Authentication Required:** No
- **Request Body:**

```
{
  "email": "user@example.com",
  "password": "StrongPassword123"
}
```

- **Response (Success - 200 OK):**

```
{
  "access_token": "your_jwt_token",
  "token_type": "bearer"
}
```

- **Response (Error - 401 Unauthorized):**

```
{"detail": "Invalid credentials"}
```

Endpoint: Get the connected user's profile

- **Method:** `GET`
- **Path:** `/v1/users/me`
- **Description:** Returns the profile information of the authenticated user.
- **Authentication Required:** Yes (Any role)
- **Response (Success - 200 OK):**

```
{
  "id": "user-uuid",
  "email": "user@example.com",
  "first_name": "John",
  "last_name": "Doe",
  "role": "b2c",
  "company_name": null
}
```

Resource: Barrels (`/v1/barrels`)

Endpoint: List all barrels

- **Method:** `GET`
- **Path:** `/v1/barrels`
- **Description:** Returns a paginated and filterable list of all available barrels.
- **Authentication Required:** No
- **Query Parameters (Query Params):**
    - `skip` : `integer` (Optional, number of items to skip, default 0)
    - `limit` : `integer` (Optional, max number of items to return, default 100)
    - `origin_country` : `string` (Optional, filter by country of origin)
    - `previous_content` : `string` (Optional, filter by previous content)
    - `min_volume` : `numeric` (Optional, minimum volume)
    - `max_volume` : `numeric` (Optional, maximum volume)
- **Response (Success - 200 OK):**

```
[
  {
    "id": "barrel-uuid-1",
    "name": "French Oak Barrel Ex-Red Wine",
    "origin_country": "France",
    "volume_liters": 225,
    "price": 350.00,
    "stock_quantity": 5
  },
  // ... other barrels
]
```

Endpoint: Get a barrel by ID

- **Method:** `GET`
- **Path:** `/v1/barrels/{barrel_id}`

- **Description:** Returns the details of a specific barrel.
- **Authentication Required:** No
- **Response (Success - 200 OK):**

```
{
  "id": "barrel-uuid-1",
  "name": "French Oak Barrel Ex-Red Wine",
  "origin_country": "France",
  "previous_content": "Red Wine",
  "volume_liters": 225,
  "wood_type": "French Oak",
  "condition": "Used",
  "price": 350.00,
  "stock_quantity": 5,
  "description": "225L barrel having contained red wine from Bordeau
  "image_urls": ["image_url_1", "image_url_2"]
}
```

- **Response (Error - 404 Not Found):**

```
{"detail": "Barrel not found"}
```

Endpoint: Create a new barrel (Admin)

- **Method:** `POST`
- **Path:** `/v1/barrels`
- **Description:** Adds a new barrel to the catalog.
- **Authentication Required:** Yes (Role: `admin`)
- **Request Body:** (similar to the GET response, without the ID and dates)
- **Response (Success - 201 Created):** (similar to the GET response)

Endpoint: Update an existing barrel (Admin)

- **Method:** `PUT`
- **Path:** `/v1/barrels/{barrel_id}`
- **Description:** Updates the information of a specific barrel.
- **Authentication Required:** Yes (Role: `admin`)
- **Request Body:** (similar to the GET response, with the fields to be modified)
- **Response (Success - 200 OK):** (similar to the GET response)

Endpoint: Delete a barrel (Admin)

- **Method:** `DELETE`
- **Path:** `/v1/barrels/{barrel_id}`
- **Description:** Deletes a barrel from the catalog.
- **Authentication Required:** Yes (Role: `admin`)
- **Response (Success - 204 No Content):** (No response body)

Resource: Orders (`/v1/orders`)

Endpoint: Create a new order

- **Method:** `POST`
- **Path:** `/v1/orders`
- **Description:** Creates a new order for the authenticated user.
- **Authentication Required:** Yes (Any role)
- **Request Body:**

```
{
  "shipping_address_id": "shipping-address-uuid",
  "billing_address_id": "billing-address-uuid", // Optional
  "items": [
    {"barrel_id": "barrel-uuid-1", "quantity": 1},
    {"barrel_id": "barrel-uuid-2", "quantity": 2}
  ]
}
```

- **Response (Success - 201 Created):**

```
{
  "id": "order-uuid",
  "user_id": "user-uuid",
  "total_amount": 1050.00,
  "status": "pending",
  "order_date": "2025-08-27T10:00:00Z"
}
```

Endpoint: List user's orders

- **Method:** `GET`
- **Path:** `/v1/orders`
- **Description:** Returns the list of orders for the authenticated user.
- **Authentication Required:** Yes (Any role)

- **Response (Success - 200 OK):** (List of order objects)

Endpoint: Get an order by ID

- **Method:** `GET`
- **Path:** `/v1/orders/{order_id}`
- **Description:** Returns the details of a specific order for the authenticated user.
- **Authentication Required:** Yes (Any role)
- **Response (Success - 200 OK):** (Order details with `OrderItems`)

Resource: Quotes (`/v1/quotes`)

Endpoint: Request a quote

- **Method:** `POST`
- **Path:** `/v1/quotes`
- **Description:** Submits a quote request for barrels.
- **Authentication Required:** Yes (Role: `b2b`)
- **Request Body:**

```
{
  "requested_items": [
    {"barrel_id": "barrel-uuid-3", "quantity": 10},
    {"barrel_id": "barrel-uuid-4", "quantity": 5}
  ],
  "notes": "Needed for a new distillery project."
}
```

- **Response (Success - 201 Created):**

```
{
  "id": "quote-uuid",
  "user_id": "b2b-user-uuid",
  "status": "pending",
  "request_date": "2025-08-27T10:30:00Z"
}
```

Endpoint: List user's quotes

- **Method:** `GET`
- **Path:** `/v1/quotes`

- **Description:** Returns the list of quotes for the authenticated user.
- **Authentication Required:** Yes (Role: `b2b`)
- **Response (Success - 200 OK):** (List of quote objects)

Endpoint: Get a quote by ID

- **Method:** `GET`
- **Path:** `/v1/quotes/{quote_id}`
- **Description:** Returns the details of a specific quote for the authenticated user.
- **Authentication Required:** Yes (Role: `b2b`)
- **Response (Success - 200 OK):** (Quote details)

---

4. Next Steps

- Definition of administration endpoints (user management, order management, quote management).
- Detail of Pydantic schemas for each request and response body.
