

# **Typology Redesign**

## Technical Document

# Introduction

This document discusses the redesign of the Typology website. Typology is a skincare brand that appeals to people who like their skincare to be straightforward and environmentally friendly. The company was started by Ning Li and is known for its simple, natural products and its commitment to not testing on animals. The products are made in France and come in eco-friendly packaging.

The redesign of the Typology website aims to make shopping for skincare products online a better experience. The site will be easy to use and will help customers find out more details about the products. The idea is to create a place where people can buy skincare items and learn about how to take care of their skin using products that are simple and honest.

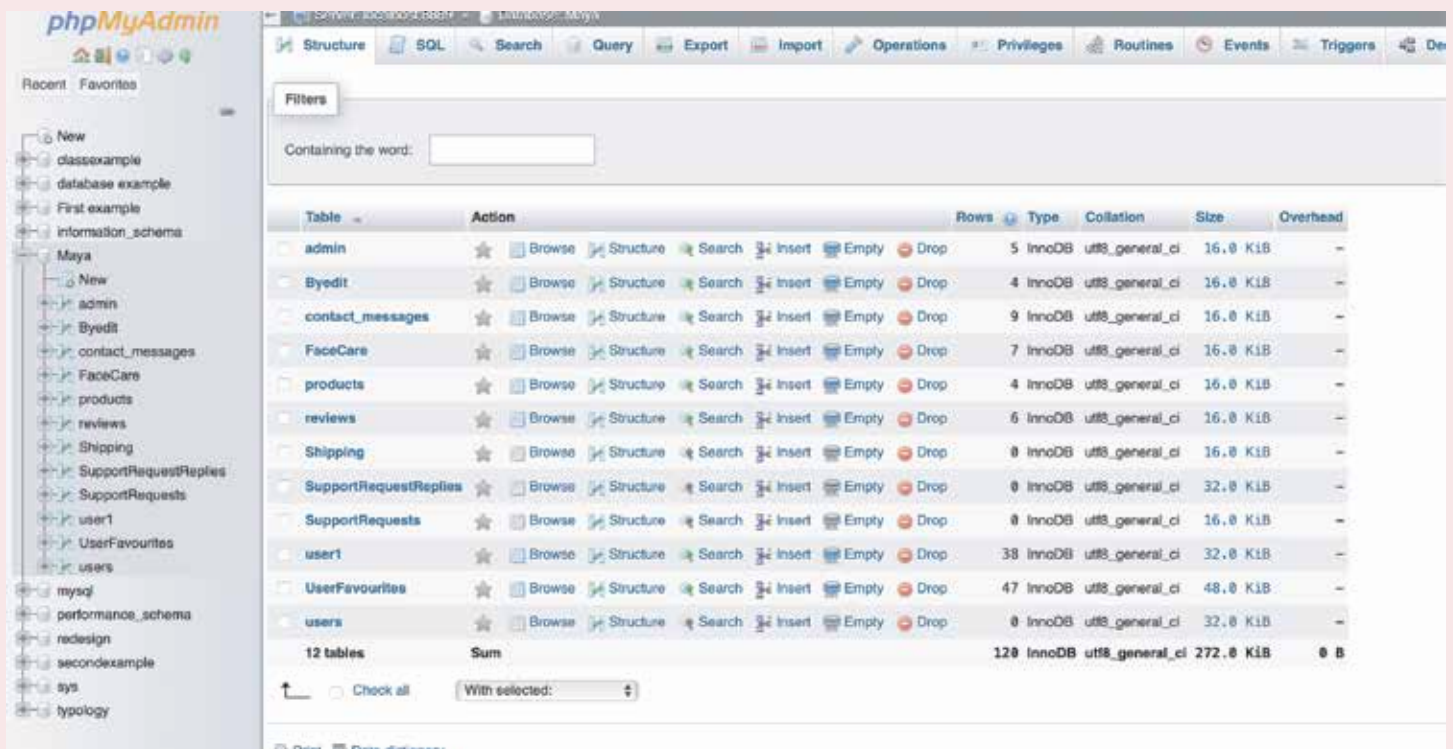
People of all ages who care about using clean and clear skincare items will enjoy the website. The brand is all about giving people what they need without any extra fuss, especially for those who want to understand what's in their skincare products.

Overall, the goal is to make the Typology website inviting and easy to use, keeping in mind the company's values of being clear about what goes into their products and caring for the environment. This document will cover the technical steps taken to build the website, such as setting up the database, getting the server ready, adding dynamic features, and testing everything to make sure it all works well for the user.

# Database setup

The Typology website uses MySQL for organizing it's data because it works well with PHP, the language used to create the website. MySQL is good for websites because it's easy to use, safe, can handle a lot of data, and is fast.

To set up the database, I used MAMP. MAMP is a program that makes it easy to install and manage a database on a computer. It has a simple control panel and comes with phpMyAdmin, which is a tool for looking after the database. They set up MAMP to meet the website's needs, like how much computer memory it can use and how many people can connect to it at the same time. This helps the website run smoothly.



The screenshot shows the phpMyAdmin interface. On the left is a sidebar with a tree view of databases and tables. The main area displays a table list for the 'typology' database. The table list includes columns for Table, Action, Rows, Type, Collation, Size, and Overhead. The tables listed are: admin, Byedit, contact\_messages, FaceCars, products, reviews, Shipping, SupportRequestReplies, SupportRequests, user1, UserFavourites, and users. A summary row at the bottom indicates 12 tables with a total size of 272.0 KIB.

Table	Action	Rows	Type	Collation	Size	Overhead
admin	★ Browse Structure Search Insert Empty Drop	5	InnoDB	utf8_general_ci	16.0 KIB	-
Byedit	★ Browse Structure Search Insert Empty Drop	4	InnoDB	utf8_general_ci	16.0 KIB	-
contact_messages	★ Browse Structure Search Insert Empty Drop	9	InnoDB	utf8_general_ci	16.0 KIB	-
FaceCars	★ Browse Structure Search Insert Empty Drop	7	InnoDB	utf8_general_ci	16.0 KIB	-
products	★ Browse Structure Search Insert Empty Drop	4	InnoDB	utf8_general_ci	16.0 KIB	-
reviews	★ Browse Structure Search Insert Empty Drop	6	InnoDB	utf8_general_ci	16.0 KIB	-
Shipping	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8_general_ci	16.0 KIB	-
SupportRequestReplies	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8_general_ci	32.0 KIB	-
SupportRequests	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8_general_ci	16.0 KIB	-
user1	★ Browse Structure Search Insert Empty Drop	38	InnoDB	utf8_general_ci	32.0 KIB	-
UserFavourites	★ Browse Structure Search Insert Empty Drop	47	InnoDB	utf8_general_ci	48.0 KIB	-
users	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8_general_ci	32.0 KIB	-
12 tables	Sum	120	InnoDB	utf8_general_ci	272.0 KIB	0 B

## Schema Design:

The database schema was designed with a focus on the various aspects of the e-commerce operations.

The tables include:

**admin:** to store administrator credentials and manage access control,  
**contact\_messages:** for storing messages from the website's contact form,  
**products:** which contains information about the skincare products,  
**reviews:** to hold customer feedback and product ratings,  
**users:** for managing customer accounts and their data,  
**SupportRequests and SupportRequestReplies:** to handle customer service inquiries and responses.

These tables are related to each other through primary and foreign keys, ensuring data integrity and supporting efficient data retrieval. For example, reviews are linked to products to associate customer feedback with the correct item, and SupportRequestReplies are related to SupportRequests to track conversations in customer support.

## Initial Setup:

At the beginning, I made the database using phpMyAdmin. I set up different user roles so that the website bosses and the website itself could use the database in different ways. I also put in starting information for things like products, so the website would have details like what the products are, how much they cost, and how many are available.

# Data Manipulation Techniques:

In the Typology website, CRUD operations helped me work with the data in the database. CRUD stands for Create, Read, Update, Delete, and it's how the website manages all the information.

## Create:

When you add something new to the website, like signing up as a new user, adding a new product, or sending a message through the contact form, the website saves this new information in the database.

## Read:

When you want to see information from the website, like looking at products to buy or reading reviews, the website finds and shows you this information from the database.

## Update:

If you need to change something that's already on the website, like when you update your profile or when an admin changes the price of a product, the website makes those changes in the database.

## Delete:

When something needs to be taken off the website, like a product that's not being sold anymore or a user account that should be closed, the website can remove that information from the database. Usually, only someone like an admin can do this to keep things safe.

## SQL/Query Statements:

SQL statements are used to interact with the database. Each CRUD operation is associated with a specific SQL command:

INSERT INTO table\_name (...) VALUES (...) for creating new records, SELECT \* FROM table\_name WHERE condition for reading data, UPDATE table\_name SET column\_name = value WHERE condition for updating records, and DELETE FROM table\_name WHERE condition for deleting records. Prepared statements are used for executing these SQL commands to prevent SQL injection attacks and improve performance.

Before I change any information from the website in the database, it checks and cleans the data:

### Validation:

The website looks at the data to make sure it's the right kind. For example, it checks if an email address looks like a real email, or if a password is strong enough. This is to make sure all the information is in the format the website expects.

### Sanitization:

The website also cleans the data. This is to stop bad things like SQL injection, which are ways hackers can try to break into the website. It uses special PHP functions to clean up the data from forms, like removing any harmful code or characters, to make sure only safe data is put into the database.

# Virtual Server Setup

I chose MAMP to help me build the Typology website because it's easy to use and set up. MAMP comes with Apache, MySQL, and PHP, which are the tools I need for my website.

I set up MAMP to act like the real server where the website will go live. This means I made sure the settings on MAMP, like how much memory PHP can use and how long scripts can run, are the same as they will be on the real server. I also used the same version of PHP to avoid any surprises when moving from development to the live website.

For testing, I ran all the features of the web application to make sure they work well and fast. This includes checking how data is added, read, changed, and deleted, making sure user logins work right, and that the website responds quickly. With MAMP's PHPMysqlAdmin, I could see the database and make sure everything was correct.



# Building a Dynamic Web Application

## **What I used:**

The Typology web application's is built using HTML5, CSS3, and JavaScript. HTML5 is used to structure the content of web pages, providing a semantic layout. CSS3 is employed to style the interface, ensuring the application is visually appealing and provides a consistent user experience across different devices and browsers. JavaScript, along with jQuery ,a fast, small, and feature. JavaScript library is used to create interactive elements on the web pages, such as form validation, data loading, and animation effects that enhance the user experience without the need for full page reloads.

## **Other:**

On the server side, PHP is the chosen scripting language due to its wide adoption, ease of use.. The PHP code is responsible for handling server side logic, including form submission handling, database interaction, and session management.

## **Dynamic Content:**

Dynamic content allows content to be generated on user interactions and database queries. PHP scripts on the server take requests from the user's browser, interact with the database to fetch, insert, update, or delete data, and then generate HTML content, which is sent back to the browser. This process ensures that users see up to date information without having to manually refresh the page.



## **User Interaction:**

User interactions are what works behind the dynamic of the website. Actions taken by users such as logging in, searching for products, posting reviews, or adding items to the shopping cart. These requests are processed by PHP scripts, which in turn perform the necessary database operations. The results of these operations are then reflected in real-time on the web pages. For example, when a user submits a review, a PHP script processes the input, stores it in the reviews table in the database, and then updates the product page to include the new review without requiring a page reload.

# Test Cases

## Testing Strategy:

The testing approach was structured around the IPO model. Each user action, defined as an input, triggers a specific process on the server, leading to an expected output. The test cases were designed to validate each step of this model to confirm that every user interaction produced the correct outcome.

## Test Scenarios:

### 1. User Login

- Input: User enters their login credentials.
- Process: The system checks if the login and password match the database records.
- Expected Output: User is either logged in and redirected to their account page or asked to re-enter details if there's a mismatch.
- Actual Outcome: As expected.

### 2. Product Selection

- Input: User selects a product.
- Process: The system retrieves product details from the database.
- Expected Output: The product information page is displayed.
- Actual Outcome: As expected.

### 3. Add to Cart

- Input: User adds a product to their shopping cart.
- Process: The product is added to the user's shopping cart and the total price is recalculated.
- Expected Output: The shopping cart page is updated with the new item and total cost.
- Actual Outcome: As expected.

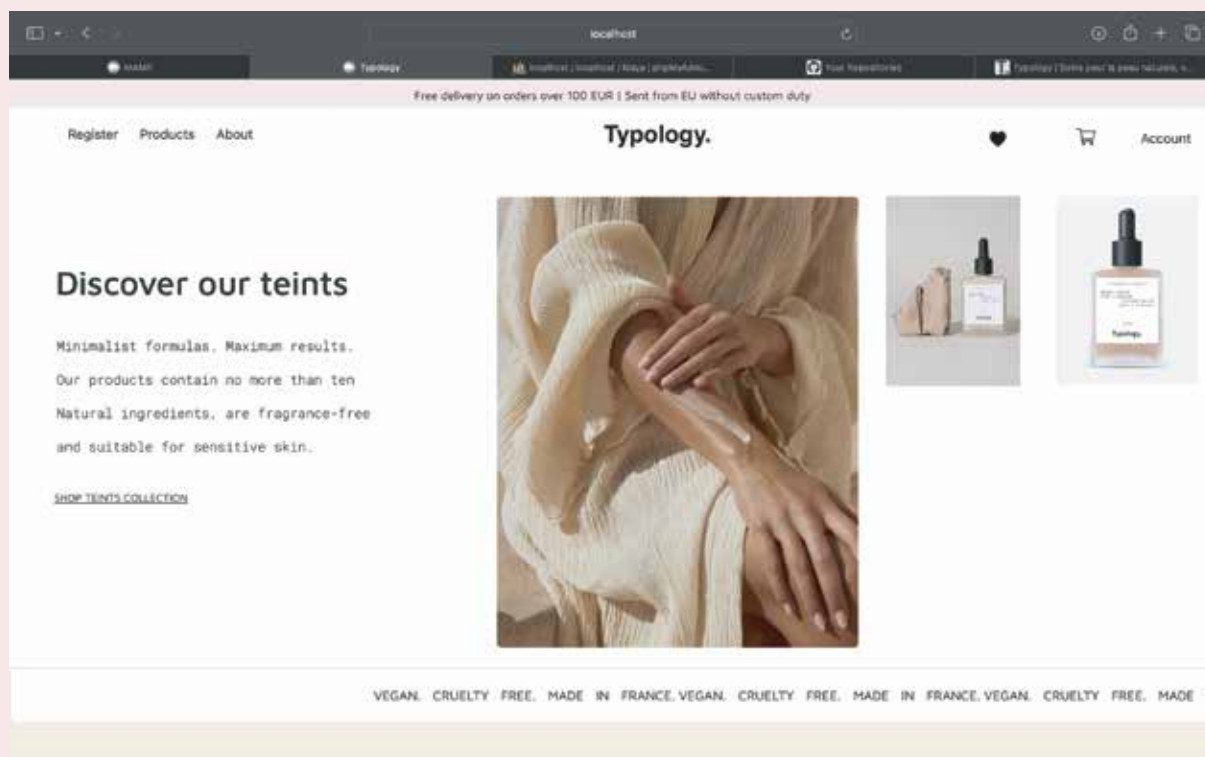
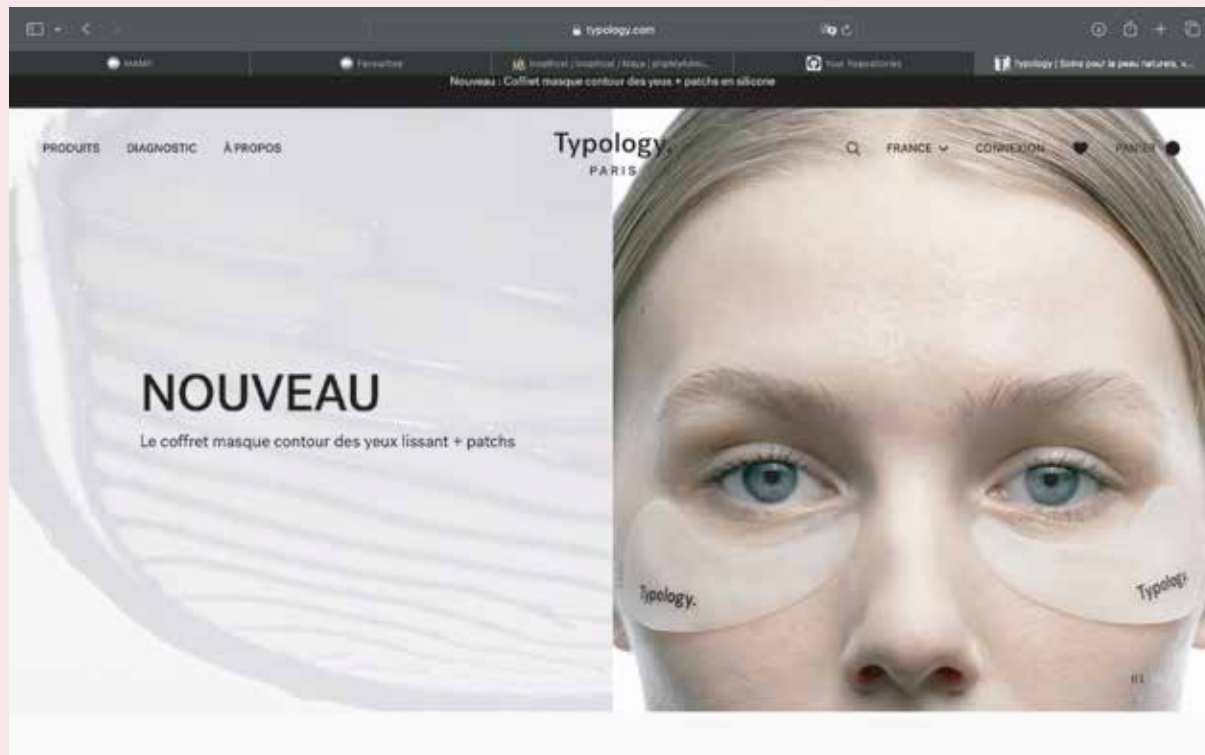
**Interactive Element Testing:**

Forms, buttons, and links were tested to ensure that they correctly captured user input and navigated to the expected pages. For example, submitting the login form was tested for both successful logins and various error conditions, such as incorrect passwords or empty fields.

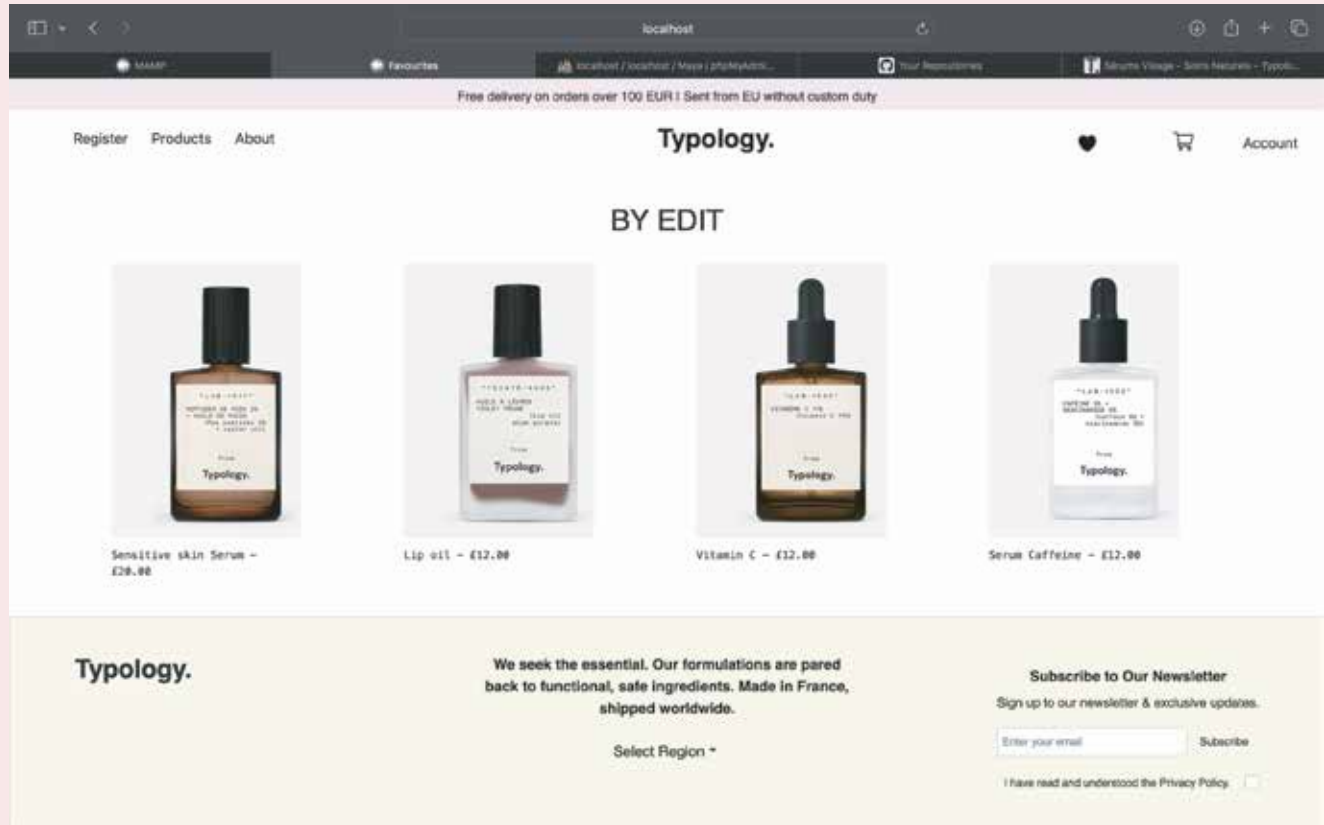
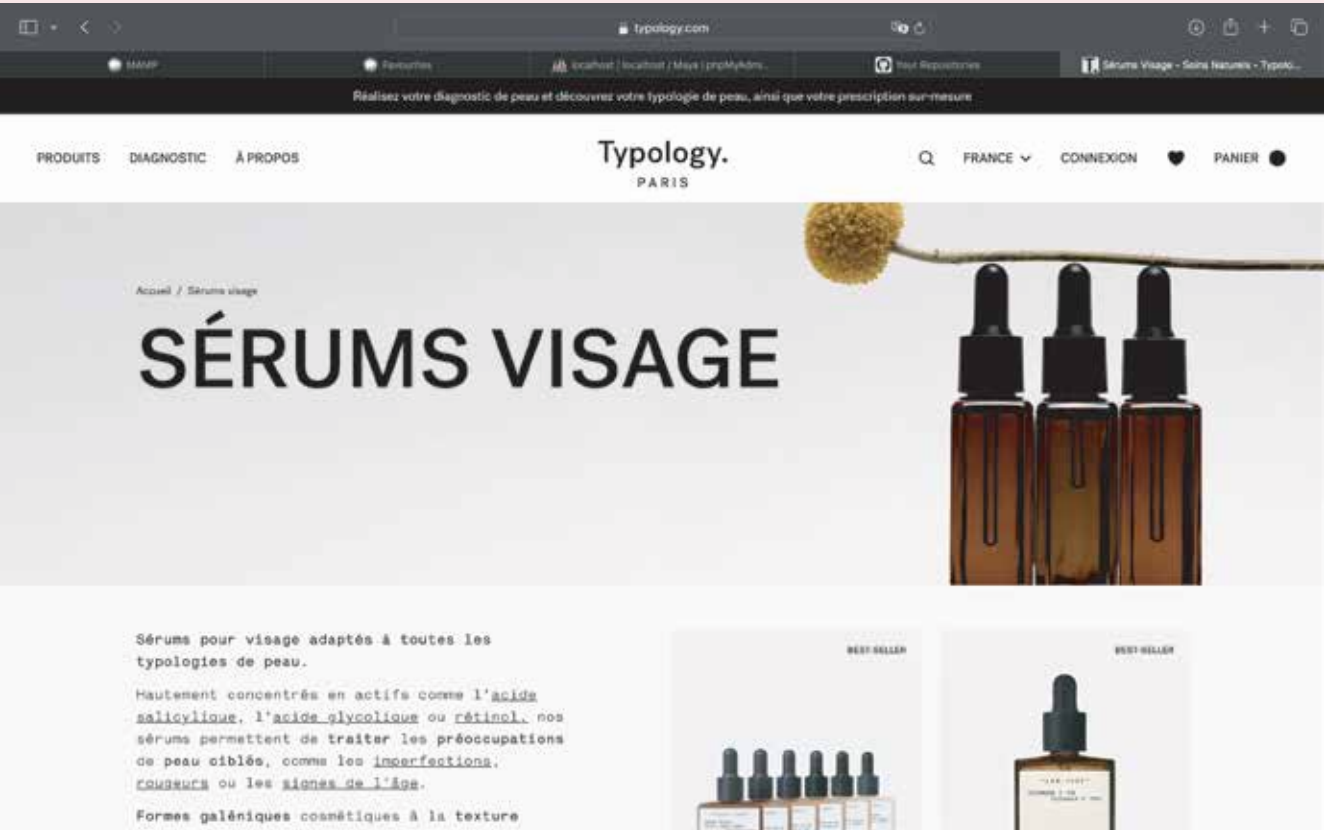
**Error Handling and Debugging:**

Errors were logged and reviewed. Error messages were displayed to guide users to correct mistakes. During debugging, tools such as browser developer consoles and log files helped identify issues, which were then resolved.

# Home page



# Byedit page



# Product page

