FACULTY OF TECHNOLOGY
DEPARTMENT OF ENGINEERING AND COMPUTER SCIENCE

OÑATE WATCH AND JEWELERY

Term paper

Informatics study programs
state code 6531BX004
Informatics study fields

Author Abraham Díaz

Software Development *15/04/2025*

Supervisor: Assoc. Prof. Dr. Aleksas Narščius

Software Development *15/04/2025*

Klaipėda, 2025

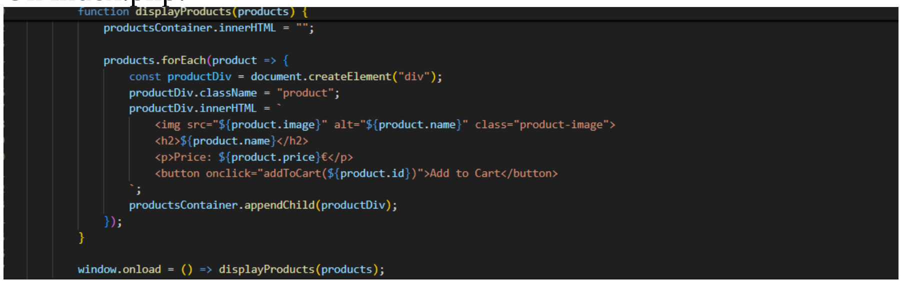
CONTENT

LIST OF TABLES

LIST OF FIGURES

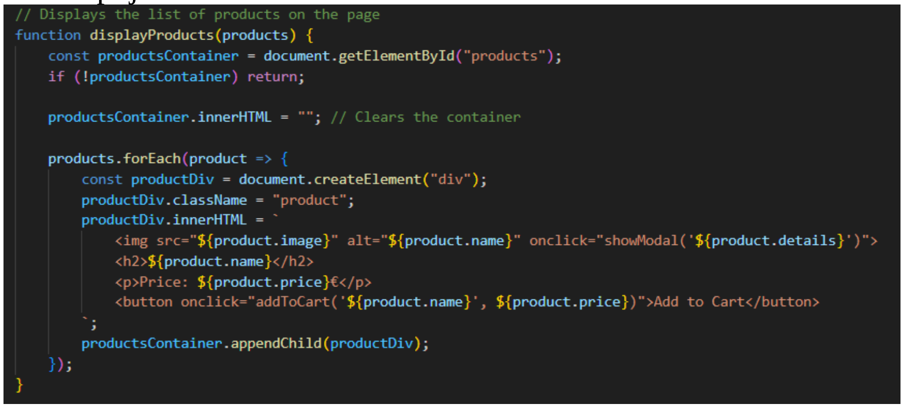
1. Main page displaying dynamically loaded products

On index.php:



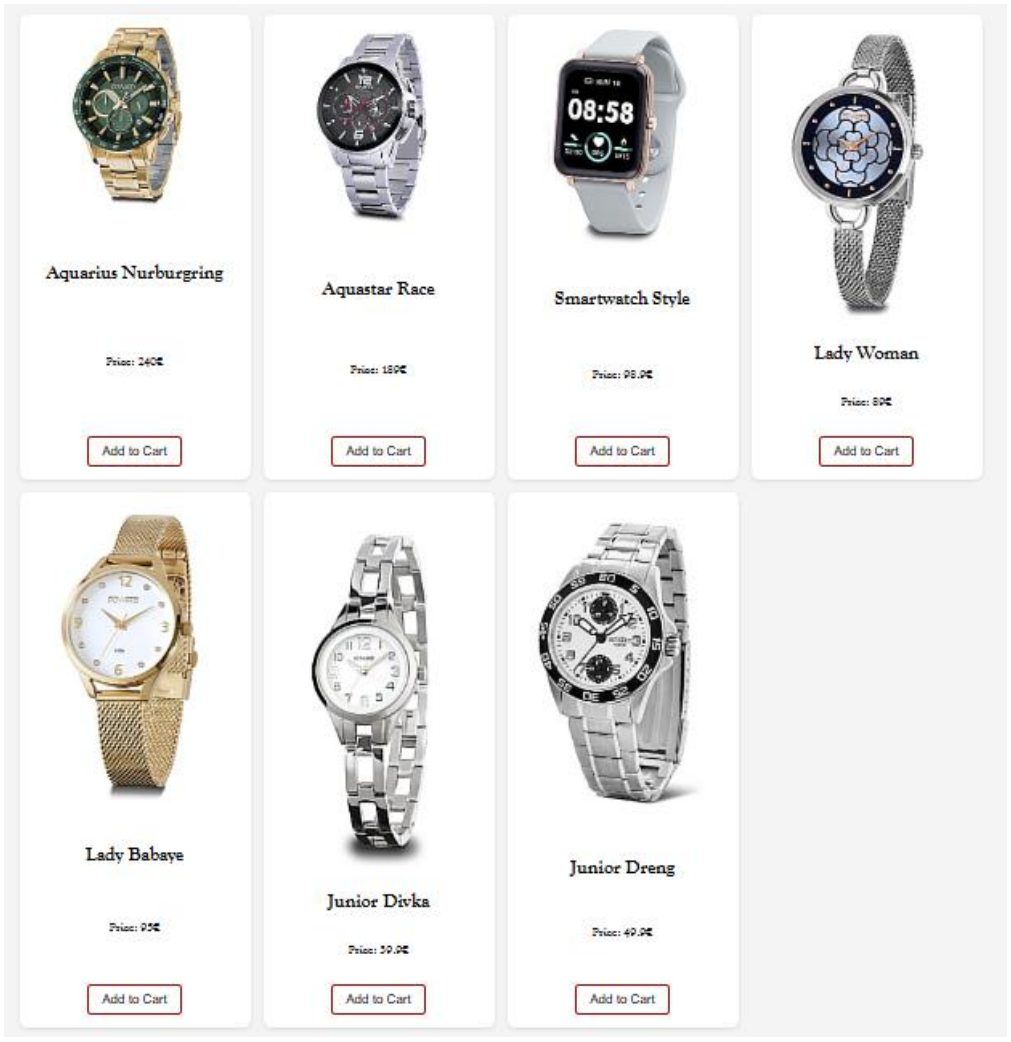
```
function displayProducts(products) {
    productsContainer.innerHTML = "";

    products.forEach(product => {
        const productDiv = document.createElement("div");
        productDiv.className = "product";
        productDiv.innerHTML = `
            <img src="${product.image}" alt="${product.name}" class="product-image">
            <h2>${product.name}</h2>
            <p>Price: ${product.price}€</p>
            <button onclick="addToCart(${product.id})">Add to Cart</button>
        `;
        productsContainer.appendChild(productDiv);
    });
}

window.onload = () => displayProducts(products);
```

On script.js:



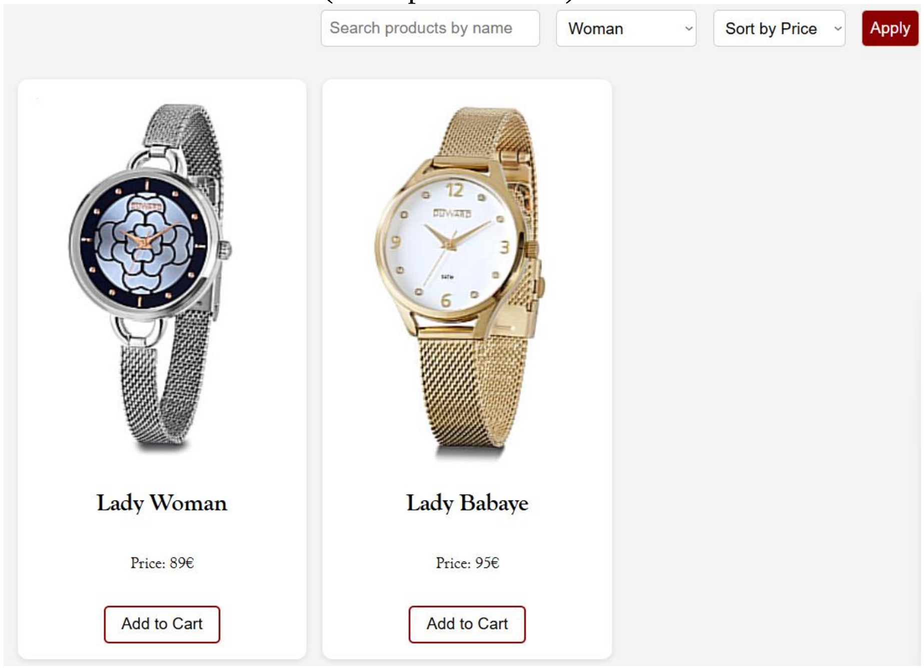
```
// Displays the list of products on the page
function displayProducts(products) {
    const productsContainer = document.getElementById("products");
    if (!productsContainer) return;

    productsContainer.innerHTML = ""; // Clears the container

    products.forEach(product => {
        const productDiv = document.createElement("div");
        productDiv.className = "product";
        productDiv.innerHTML = `
            <img src="${product.image}" alt="${product.name}" onclick="showModal('${product.details}')">
            <h2>${product.name}</h2>
            <p>Price: ${product.price}€</p>
            <button onclick="addToCart('${product.name}', ${product.price})">Add to Cart</button>
        `;
        productsContainer.appendChild(productDiv);
    });
}
```

How it looks on website:



2. **Category filtering applied (e.g., Smartwatch, Woman)**

   How it looks on website (Example: Woman):

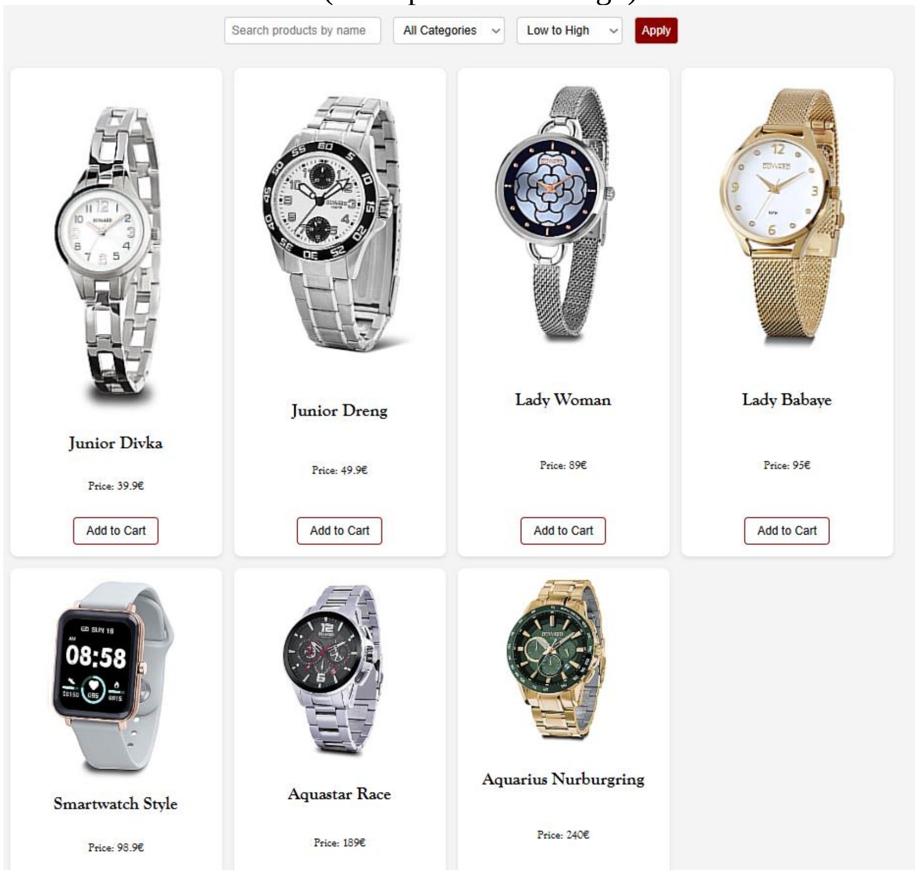


3. **Product sorting by price (ascending or descending)**

   On script.js:

```
// Sorts products by price in ascending or descending order
function sortByPrice(order = "asc") {
    const allProducts = JSON.parse(localStorage.getItem("allProducts")) || [];
    return allProducts.sort((a, b) => order === "asc" ? a.price - b.price : b.price - a.price);
}
```

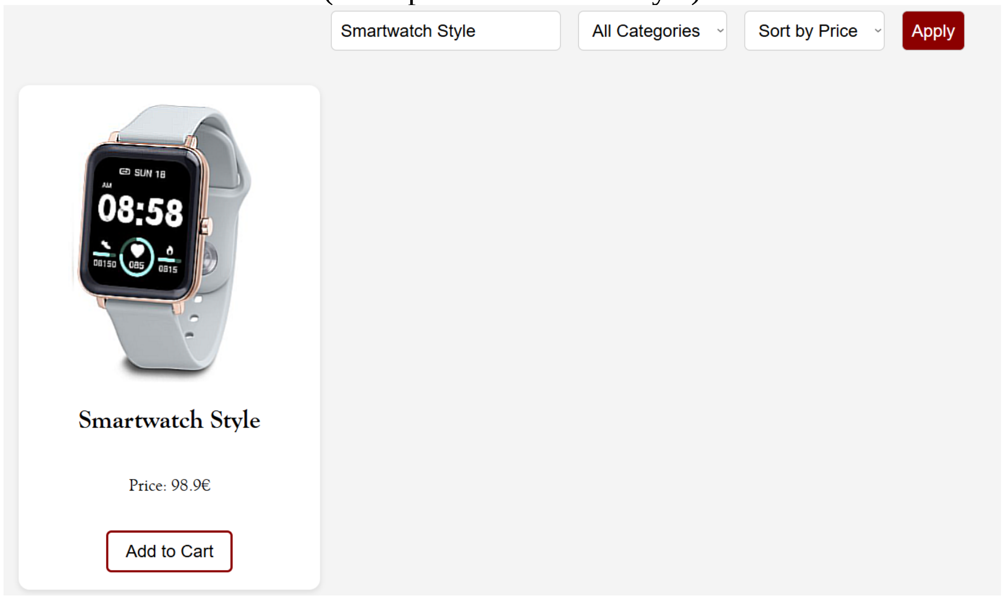
   How it looks on website (Example: Low to High):



4. **Product search with keyword input**
   On script.js:

```
// Searches products by name
function searchByName(term) {
    const allProducts = JSON.parse(localStorage.getItem("allProducts")) || [];
    return allProducts.filter(p => p.name.toLowerCase().includes(term.toLowerCase()));
}
```

   How it looks on website (Example: Smartwatch Style):

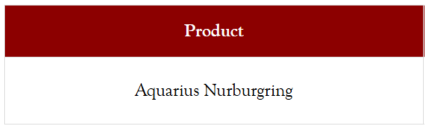


Smartwatch Style | All Categories | Sort by Price | Apply

Smartwatch Style

Price: 98.9€

5. **Shopping cart with added products**
   How it looks on website:

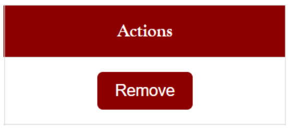


Product
Aquarius Nurburgring

6. **Delete product button in shopping cart**
   On cart.php:

```
<!-- Form to remove an item from the cart -->
<form method="POST" style="display:inline;">
    <input type="hidden" name="action" value="remove_item">
    <input type="hidden" name="product_id" value="<?= $id ?>">
    <button type="submit" class="remove-btn">Remove</button>
</form>
```

   How it looks on website:



Actions
Remove

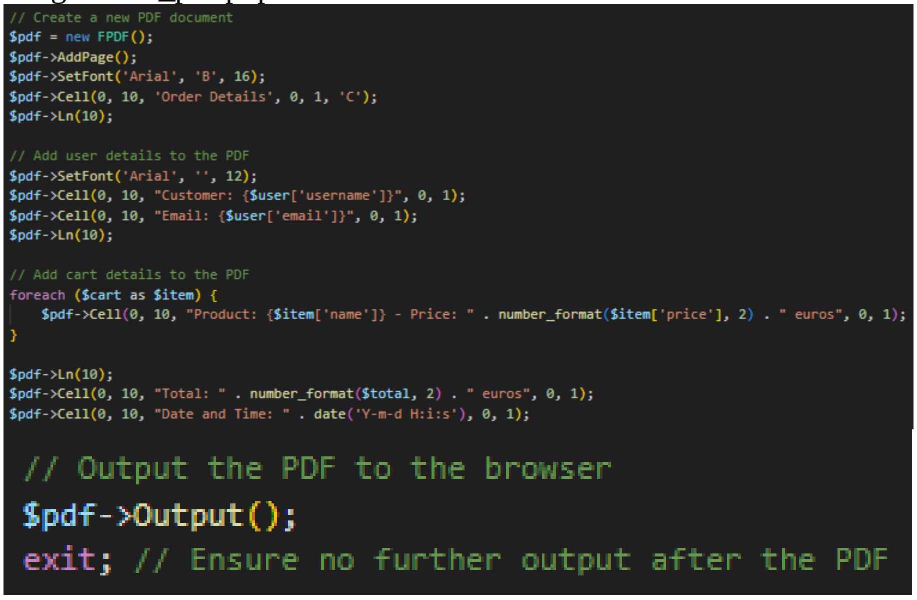
7. **Clear all items from the shopping cart**
   On cart.php:

```
<!-- Form to clear the entire cart -->
<form method="POST">
    <input type="hidden" name="action" value="clear_cart">
    <button type="submit" class="cart-action-btn">Clear Cart</button>
</form>
```

   How it looks on website:

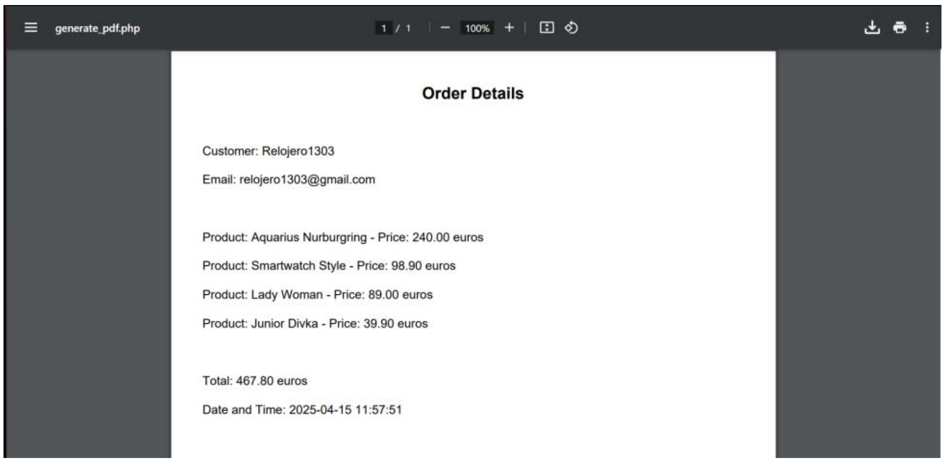
8. Purchase receipt generated as PDF using jsPDF

On generate_pdf.php:



```
// Create a new PDF document
$pdf = new FPDF();
$pdf->AddPage();
$pdf->SetFont('Arial', 'B', 16);
$pdf->Cell(0, 10, 'Order Details', 0, 1, 'C');
$pdf->Ln(10);

// Add user details to the PDF
$pdf->SetFont('Arial', '', 12);
$pdf->Cell(0, 10, "Customer: {$user['username']}", 0, 1);
$pdf->Cell(0, 10, "Email: {$user['email']}", 0, 1);
$pdf->Ln(10);

// Add cart details to the PDF
foreach ($cart as $item) {
    $pdf->Cell(0, 10, "Product: {$item['name']} - Price: " . number_format($item['price'], 2) . " euros", 0, 1);
}

$pdf->Ln(10);
$pdf->Cell(0, 10, "Total: " . number_format($total, 2) . " euros", 0, 1);
$pdf->Cell(0, 10, "Date and Time: " . date('Y-m-d H:i:s'), 0, 1);

// Output the PDF to the browser
$pdf->Output();
exit; // Ensure no further output after the PDF
```

How it looks on website:



Order Details

Customer: Relojero1303

Email: relojero1303@gmail.com

Product: Aquarius Nurburgring - Price: 240.00 euros

Product: Smartwatch Style - Price: 98.90 euros

Product: Lady Woman - Price: 89.00 euros

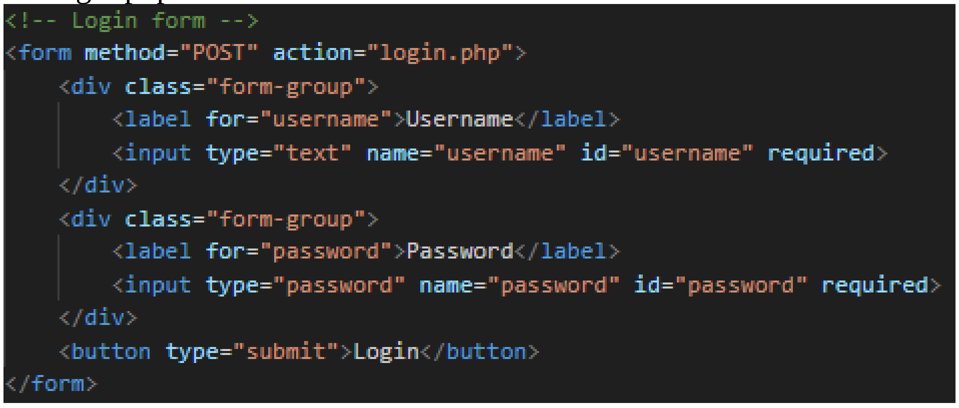
Product: Junior Divka - Price: 39.90 euros

Total: 467.80 euros

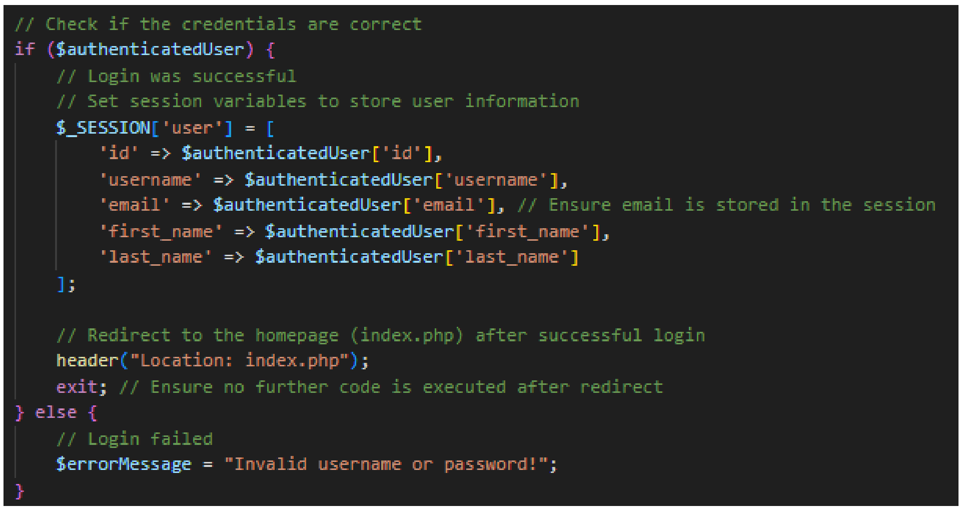
Date and Time: 2025-04-15 11:57:51

9. Login form with validation (email)

On login.php:

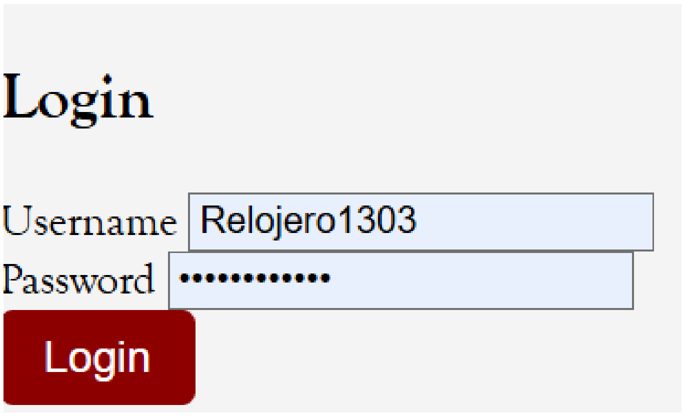


```
<!-- Login form -->
<form method="POST" action="login.php">
    <div class="form-group">
        <label for="username">Username</label>
        <input type="text" name="username" id="username" required>
    </div>
    <div class="form-group">
        <label for="password">Password</label>
        <input type="password" name="password" id="password" required>
    </div>
    <button type="submit">Login</button>
</form>
```



```
// Check if the credentials are correct
if ($authenticatedUser) {
    // Login was successful
    // Set session variables to store user information
    $_SESSION['user'] = [
        'id' => $authenticatedUser['id'],
        'username' => $authenticatedUser['username'],
        'email' => $authenticatedUser['email'], // Ensure email is stored in the session
        'first_name' => $authenticatedUser['first_name'],
        'last_name' => $authenticatedUser['last_name']
    ];

    // Redirect to the homepage (index.php) after successful login
    header("Location: index.php");
    exit; // Ensure no further code is executed after redirect
} else {
    // Login failed
    $errorMessage = "Invalid username or password!";
}
```

How it looks on website:



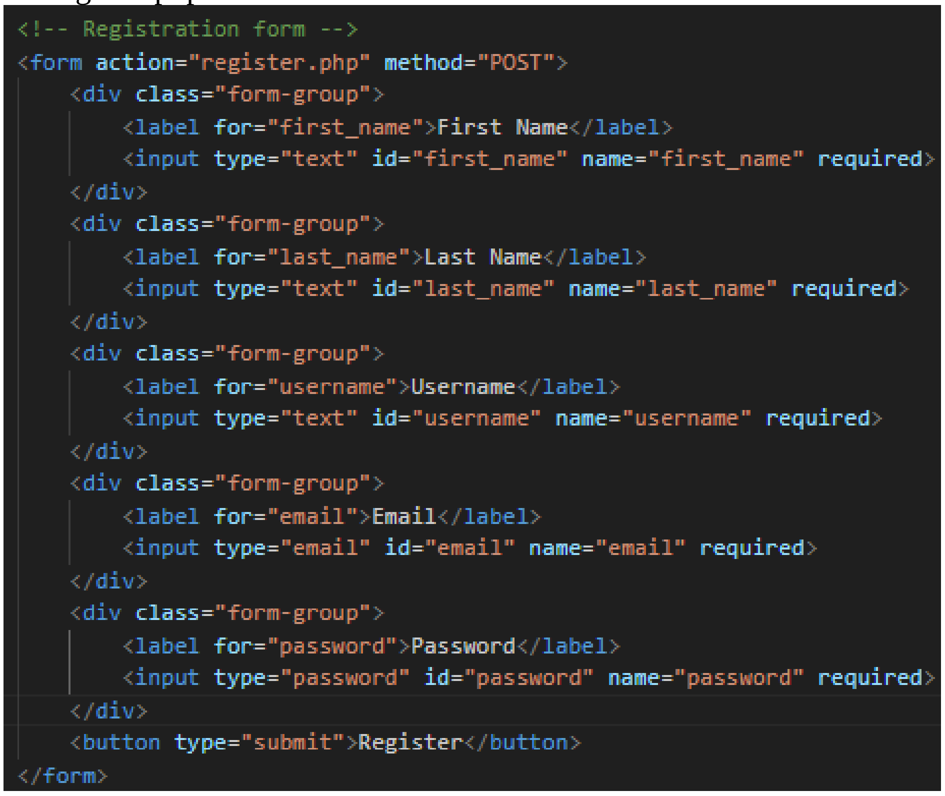
Login

Username Relojero1303

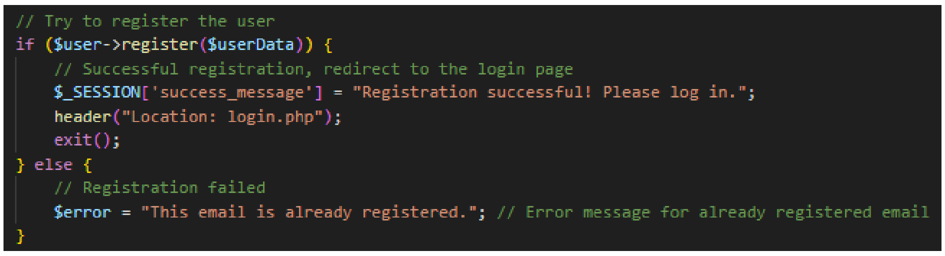
Password •••••••••••••

10. Working registration form with validations

On register.php:

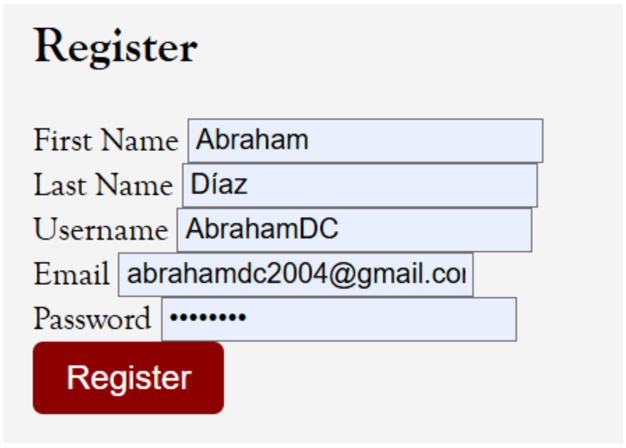


```
<!-- Registration form -->
<form action="register.php" method="POST">
    <div class="form-group">
        <label for="first_name">First Name</label>
        <input type="text" id="first_name" name="first_name" required>
    </div>
    <div class="form-group">
        <label for="last_name">Last Name</label>
        <input type="text" id="last_name" name="last_name" required>
    </div>
    <div class="form-group">
        <label for="username">Username</label>
        <input type="text" id="username" name="username" required>
    </div>
    <div class="form-group">
        <label for="email">Email</label>
        <input type="email" id="email" name="email" required>
    </div>
    <div class="form-group">
        <label for="password">Password</label>
        <input type="password" id="password" name="password" required>
    </div>
    <button type="submit">Register</button>
</form>
```



```
// Try to register the user
if ($user->register($userData)) {
    // Successful registration, redirect to the login page
    $_SESSION['success_message'] = "Registration successful! Please log in.";
    header("Location: login.php");
    exit();
} else {
    // Registration failed
    $error = "This email is already registered."; // Error message for already registered email
}
```

How it looks on website:

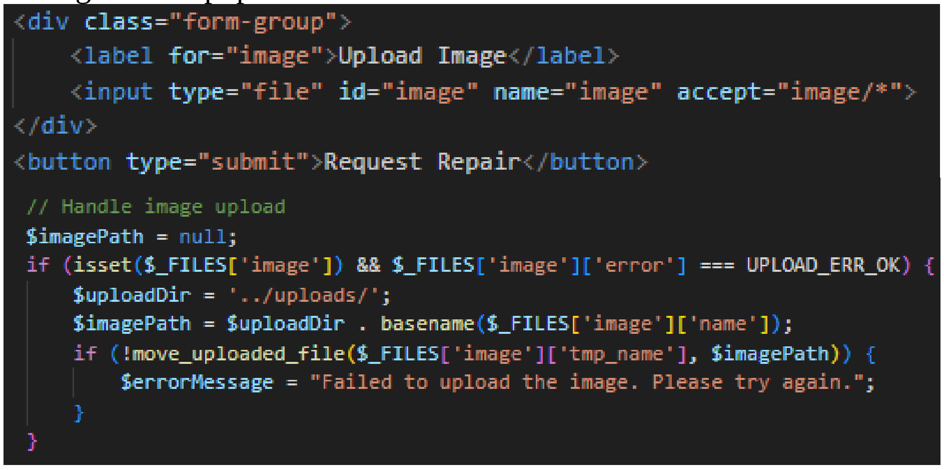


Register

First Name	Abraham
Last Name	Díaz
Username	AbrahamDC
Email	abrahamdc2004@gmail.co
Password	••••••••

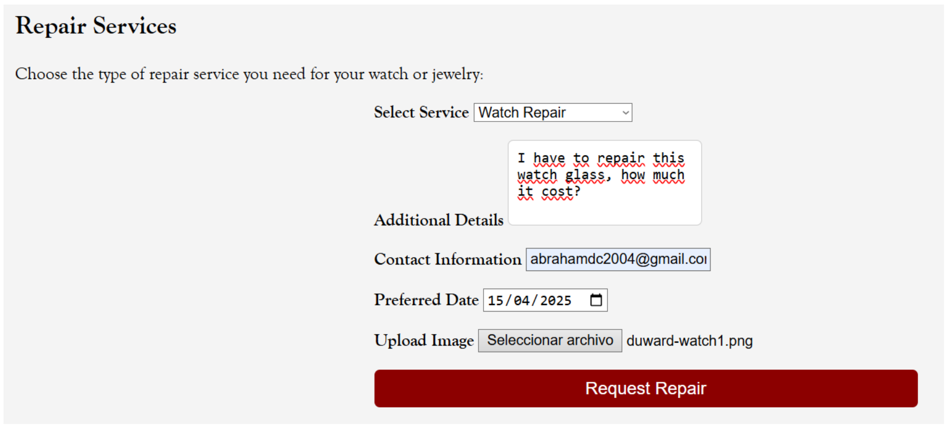
11. Repair form with image upload functionality

On registration.php:



```
<div class="form-group">
    <label for="image">Upload Image</label>
    <input type="file" id="image" name="image" accept="image/*">
</div>
<button type="submit">Request Repair</button>

// Handle image upload
$imagePath = null;
if (isset($_FILES['image']) && $_FILES['image']['error'] === UPLOAD_ERR_OK) {
    $uploadDir = '../uploads/';
    $imagePath = $uploadDir . basename($_FILES['image']['name']);
    if (!move_uploaded_file($_FILES['image']['tmp_name'], $imagePath)) {
        $errorMessage = "Failed to upload the image. Please try again.";
    }
}
```

How it looks on website:

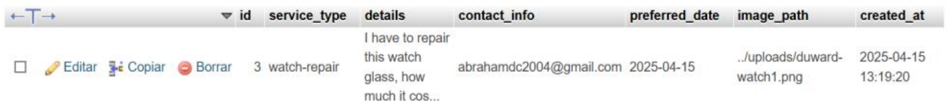


Repair Services

Choose the type of repair service you need for your watch or jewelry:

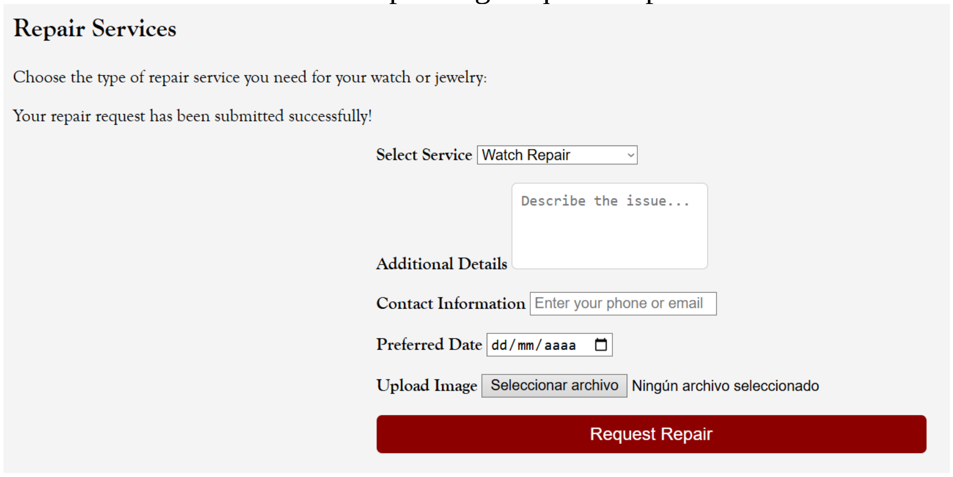
Select Service	Watch Repair
Additional Details	I have to repair this watch glass, how much it cost?
Contact Information	abrahamdc2004@gmail.co
Preferred Date	15/04/2025
Upload Image	Seleccionar archivo duward-watch1.png

On Database after pushing Request Repair button:



	id	service_type	details	contact_info	preferred_date	image_path	created_at
☐ Editar Copiar Borrar	3	watch-repair	I have to repair this watch glass, how much it cos...	abrahamdc2004@gmail.com	2025-04-15	../uploads/duward-watch1.png	2025-04-15 13:19:20

How it looks on website after pushing Request Repair button:



Repair Services

Choose the type of repair service you need for your watch or jewelry:

Your repair request has been submitted successfully!

Select Service: Watch Repair

Additional Details: Describe the issue...

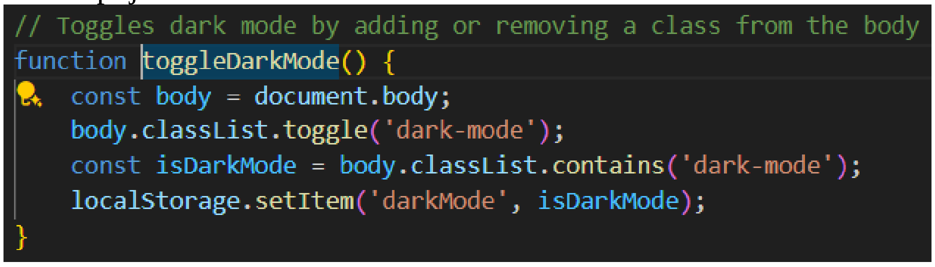
Contact Information: Enter your phone or email

Preferred Date: dd/mm/aaaa

Upload Image: Seleccionar archivo Ningún archivo seleccionado

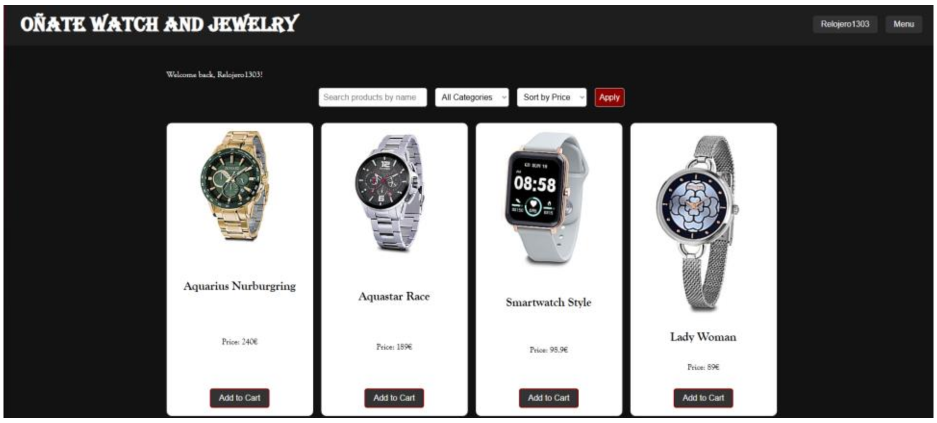
12. Dark mode activated on main page

On script.js:



```
// Toggles dark mode by adding or removing a class from the body
function toggleDarkMode() {
    const body = document.body;
    body.classList.toggle('dark-mode');
    const isDarkMode = body.classList.contains('dark-mode');
    localStorage.setItem('darkMode', isDarkMode);
}
```

How it looks on website:



Welcome back, Relojero1303!

Aquarius Nurburgring

Price: 240€

Aquastar Race

Price: 189€

Smartwatch Style

Price: 98.9€

Lady Woman

Price: 89€

13. Database schema shown in phpMyAdmin (4 tables and relations)

Relations:

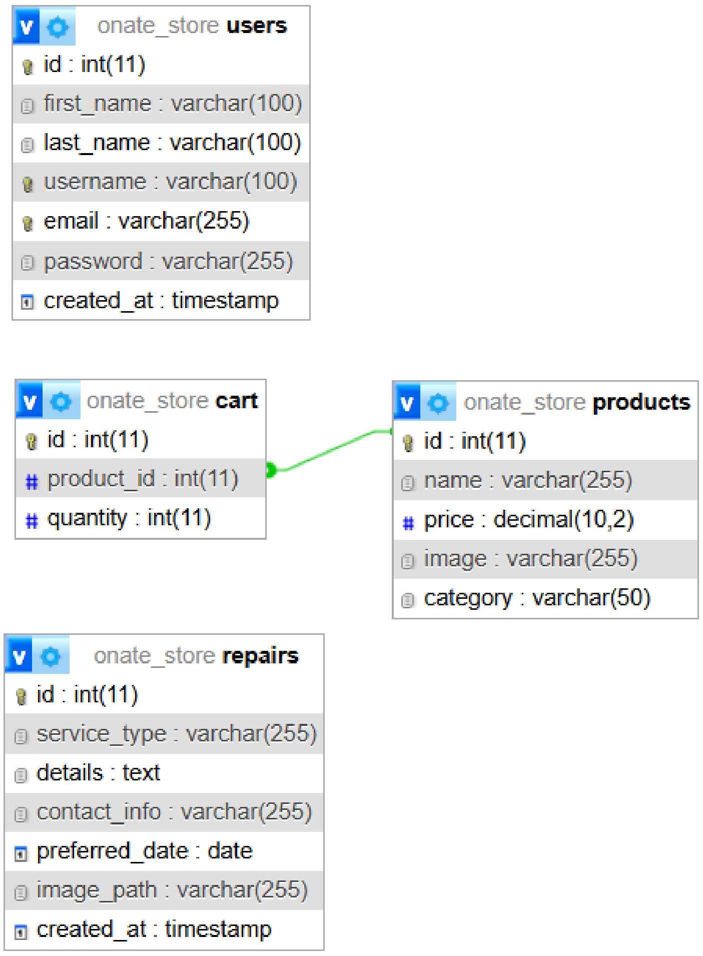
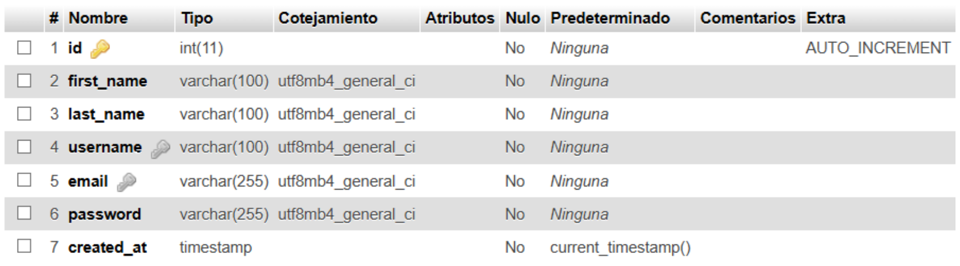


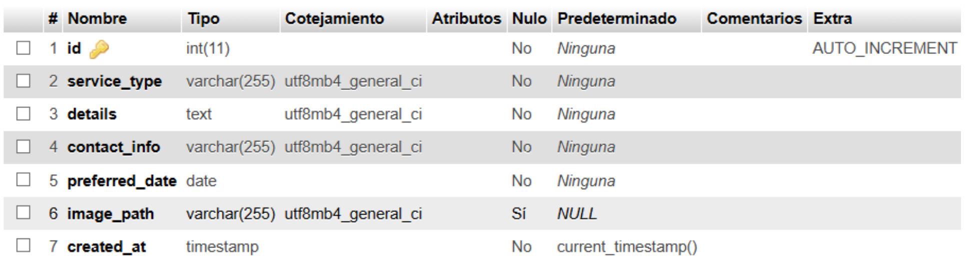
Table Structure:

- user



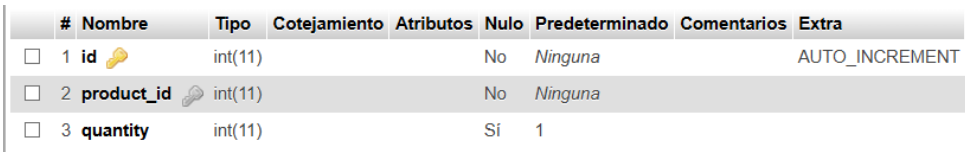
	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
☐	1	**id**	int(11)			No	*Ninguna*		AUTO_INCREMENT
☐	2	**first_name**	varchar(100)	utf8mb4_general_ci		No	*Ninguna*		
☐	3	**last_name**	varchar(100)	utf8mb4_general_ci		No	*Ninguna*		
☐	4	**username**	varchar(100)	utf8mb4_general_ci		No	*Ninguna*		
☐	5	**email**	varchar(255)	utf8mb4_general_ci		No	*Ninguna*		
☐	6	**password**	varchar(255)	utf8mb4_general_ci		No	*Ninguna*		
☐	7	**created_at**	timestamp			No	current_timestamp()		

- repairs



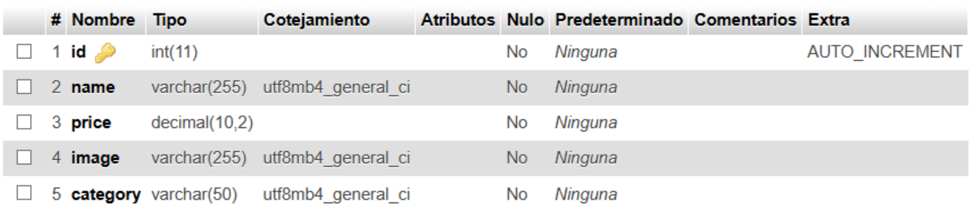
	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
☐	1	**id**	int(11)			No	*Ninguna*		AUTO_INCREMENT
☐	2	**service_type**	varchar(255)	utf8mb4_general_ci		No	*Ninguna*		
☐	3	**details**	text	utf8mb4_general_ci		No	*Ninguna*		
☐	4	**contact_info**	varchar(255)	utf8mb4_general_ci		No	*Ninguna*		
☐	5	**preferred_date**	date			No	*Ninguna*		
☐	6	**image_path**	varchar(255)	utf8mb4_general_ci		Sí	*NULL*		
☐	7	**created_at**	timestamp			No	current_timestamp()		

- cart



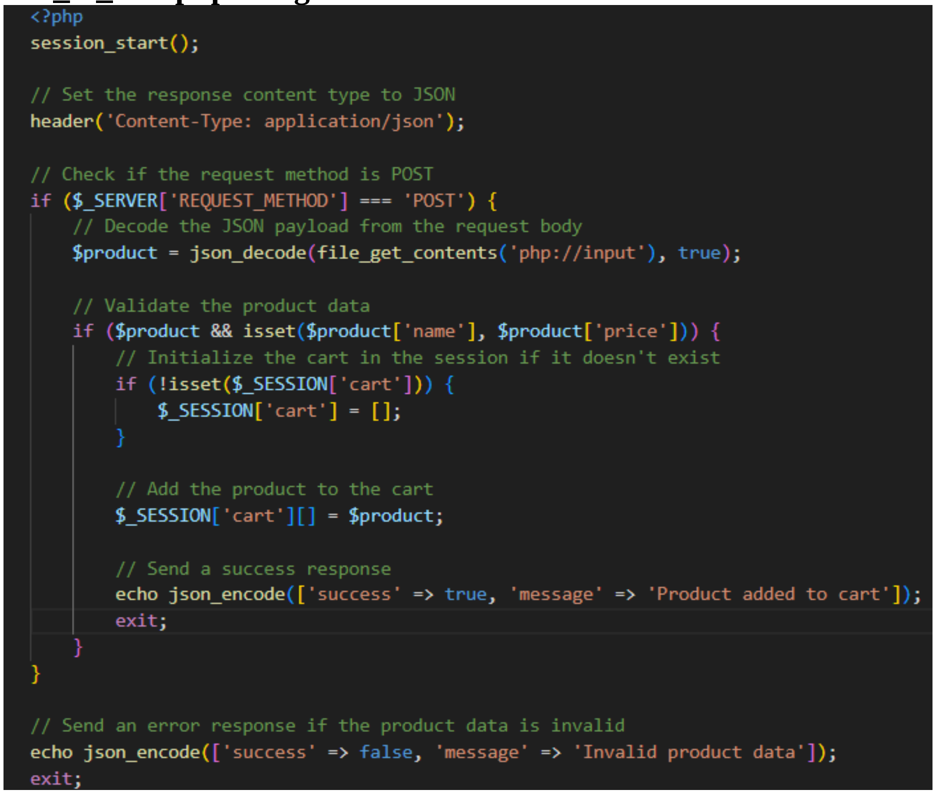
	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
☐	1	**id**	int(11)			No	*Ninguna*		AUTO_INCREMENT
☐	2	**product_id**	int(11)			No	*Ninguna*		
☐	3	**quantity**	int(11)			Sí	1		

- product



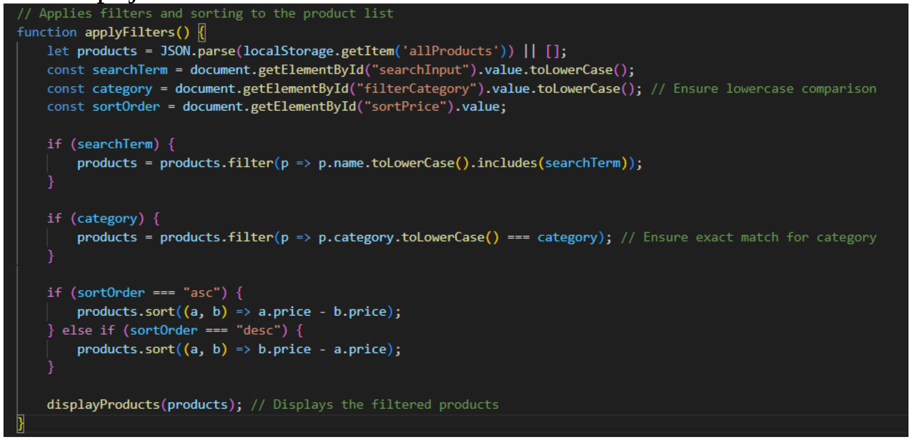
	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
☐	1	**id**	int(11)			No	*Ninguna*		AUTO_INCREMENT
☐	2	**name**	varchar(255)	utf8mb4_general_ci		No	*Ninguna*		
☐	3	**price**	decimal(10,2)			No	*Ninguna*		
☐	4	**image**	varchar(255)	utf8mb4_general_ci		No	*Ninguna*		
☐	5	**category**	varchar(50)	utf8mb4_general_ci		No	*Ninguna*		

14. add_to_cart.php using PHP sessions



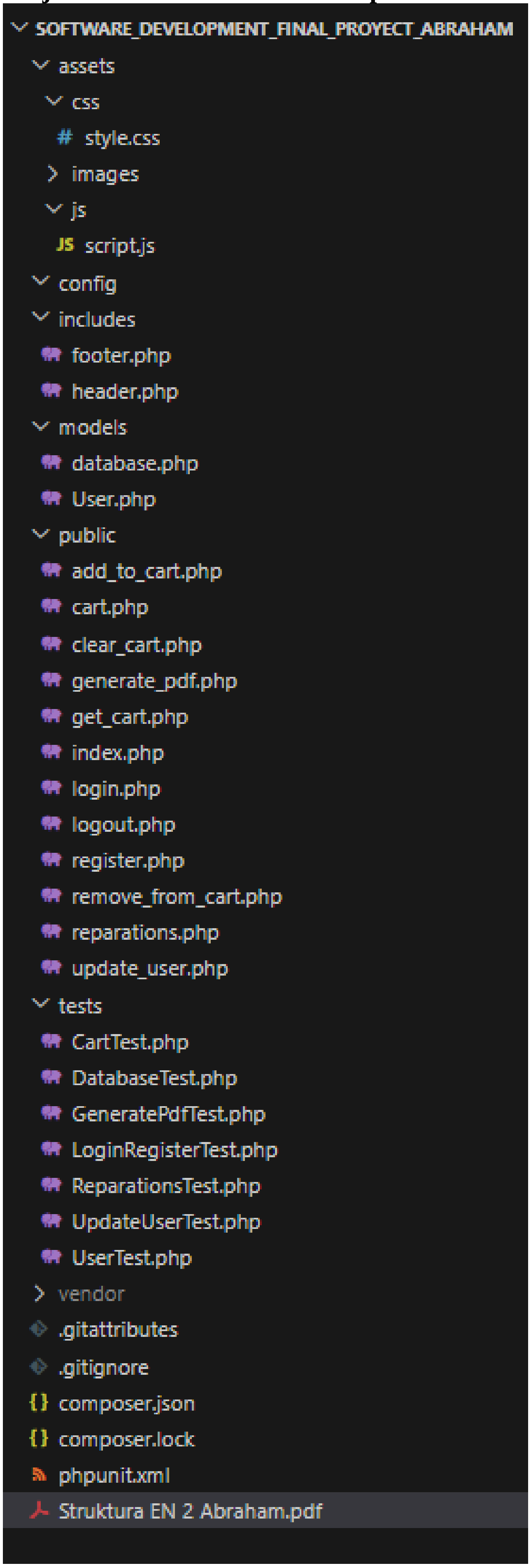
```php
<?php
session_start();

// Set the response content type to JSON
header('Content-Type: application/json');

// Check if the request method is POST
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Decode the JSON payload from the request body
    $product = json_decode(file_get_contents('php://input'), true);

    // Validate the product data
    if ($product && isset($product['name'], $product['price'])) {
        // Initialize the cart in the session if it doesn't exist
        if (!isset($_SESSION['cart'])) {
            $_SESSION['cart'] = [];
        }

        // Add the product to the cart
        $_SESSION['cart'][] = $product;

        // Send a success response
        echo json_encode(['success' => true, 'message' => 'Product added to cart']);
        exit;
    }
}

// Send an error response if the product data is invalid
echo json_encode(['success' => false, 'message' => 'Invalid product data']);
exit;
```

15. JavaScript filtering and sorting logic
On script.js:



```javascript
// Applies filters and sorting to the product list
function applyFilters() {
    let products = JSON.parse(localStorage.getItem('allProducts')) || [];
    const searchTerm = document.getElementById("searchInput").value.toLowerCase();
    const category = document.getElementById("filterCategory").value.toLowerCase(); // Ensure lowercase comparison
    const sortOrder = document.getElementById("sortPrice").value;

    if (searchTerm) {
        products = products.filter(p => p.name.toLowerCase().includes(searchTerm));
    }

    if (category) {
        products = products.filter(p => p.category.toLowerCase() === category); // Ensure exact match for category
    }

    if (sortOrder === "asc") {
        products.sort((a, b) => a.price - b.price);
    } else if (sortOrder === "desc") {
        products.sort((a, b) => b.price - a.price);
    }

    displayProducts(products); // Displays the filtered products
}
```

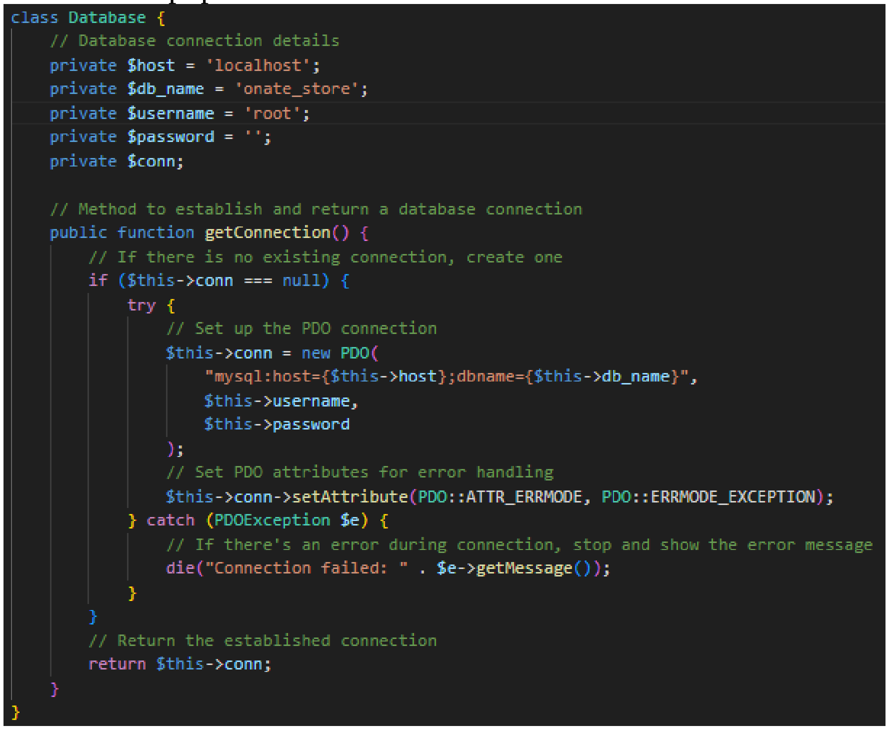
16. Project folder structure in file explorer



```
SOFTWARE_DEVELOPMENT_FINAL_PROYECT_ABRAHAM
  assets
    css
      style.css
    images
    js
      script.js
  config
  includes
    footer.php
    header.php
  models
    database.php
    User.php
  public
    add_to_cart.php
    cart.php
    clear_cart.php
    generate_pdf.php
    get_cart.php
    index.php
    login.php
    logout.php
    register.php
    remove_from_cart.php
    reparations.php
    update_user.php
  tests
    CartTest.php
    DatabaseTest.php
    GeneratePdfTest.php
    LoginRegisterTest.php
    ReparationsTest.php
    UpdateUserTest.php
    UserTest.php
  vendor
  .gitattributes
  .gitignore
  composer.json
  composer.lock
  phpunit.xml
  Struktura EN 2 Abraham.pdf
```

17. Example of Singleton pattern (config or DB connection)

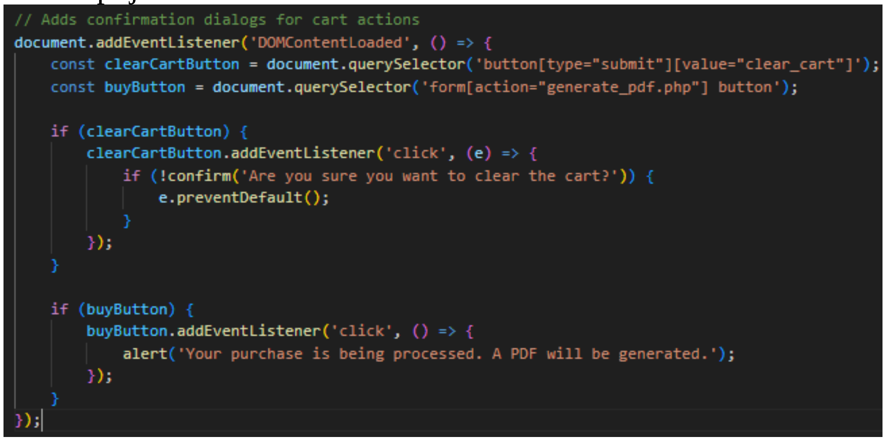
On database.php:



```php
class Database {
    // Database connection details
    private $host = 'localhost';
    private $db_name = 'onate_store';
    private $username = 'root';
    private $password = '';
    private $conn;

    // Method to establish and return a database connection
    public function getConnection() {
        // If there is no existing connection, create one
        if ($this->conn === null) {
            try {
                // Set up the PDO connection
                $this->conn = new PDO(
                    "mysql:host={$this->host};dbname={$this->db_name}",
                    $this->username,
                    $this->password
                );
                // Set PDO attributes for error handling
                $this->conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
            } catch (PDOException $e) {
                // If there's an error during connection, stop and show the error message
                die("Connection failed: " . $e->getMessage());
            }
        }
        // Return the established connection
        return $this->conn;
    }
}
```

18. Observer pattern behavior using JavaScript events

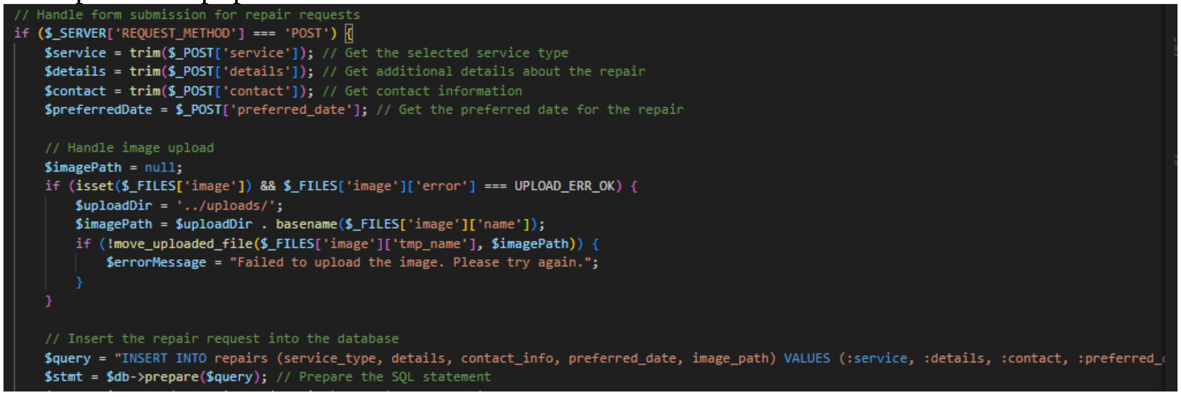
On script.js:



```javascript
// Adds confirmation dialogs for cart actions
document.addEventListener('DOMContentLoaded', () => {
    const clearCartButton = document.querySelector('button[type="submit"][value="clear_cart"]');
    const buyButton = document.querySelector('form[action="generate_pdf.php"] button');

    if (clearCartButton) {
        clearCartButton.addEventListener('click', (e) => {
            if (!confirm('Are you sure you want to clear the cart?')) {
                e.preventDefault();
            }
        });
    }

    if (buyButton) {
        buyButton.addEventListener('click', () => {
            alert('Your purchase is being processed. A PDF will be generated.');
        });
    }
});
```

19. Repair form with image file selected for upload

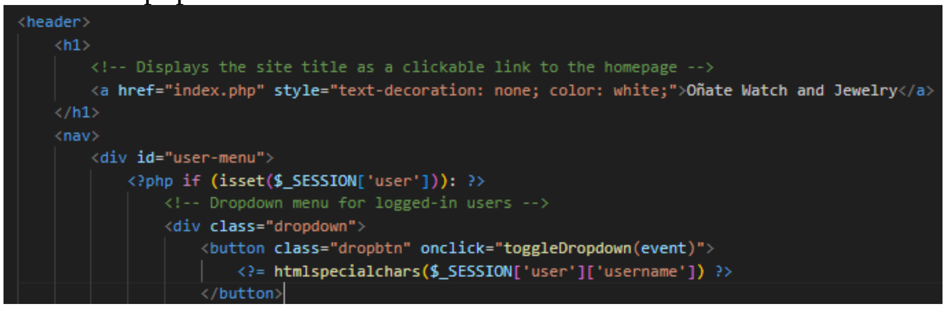
On reparations.php:



```php
// Handle form submission for repair requests
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $service = trim($_POST['service']); // Get the selected service type
    $details = trim($_POST['details']); // Get additional details about the repair
    $contact = trim($_POST['contact']); // Get contact information
    $preferredDate = $_POST['preferred_date']; // Get the preferred date for the repair

    // Handle image upload
    $imagePath = null;
    if (isset($_FILES['image']) && $_FILES['image']['error'] === UPLOAD_ERR_OK) {
        $uploadDir = '../uploads/';
        $imagePath = $uploadDir . basename($_FILES['image']['name']);
        if (!move_uploaded_file($_FILES['image']['tmp_name'], $imagePath)) {
            $errorMessage = "Failed to upload the image. Please try again.";
        }
    }

    // Insert the repair request into the database
    $query = "INSERT INTO repairs (service_type, details, contact_info, preferred_date, image_path) VALUES (:service, :details, :contact, :preferred_
    $stmt = $db->prepare($query); // Prepare the SQL statement
```

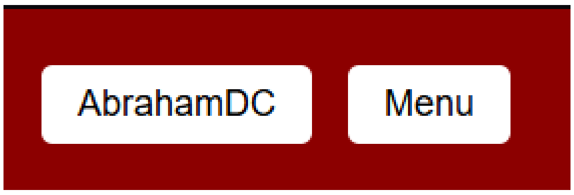
20. Session data usage in the header (showing username)

    On header.php:



```
<header>
    <h1>
        <!-- Displays the site title as a clickable link to the homepage -->
        <a href="index.php" style="text-decoration: none; color: white;">Oñate Watch and Jewelry</a>
    </h1>
    <nav>
        <div id="user-menu">
            <?php if (isset($_SESSION['user'])): ?>
                <!-- Dropdown menu for logged-in users -->
                <div class="dropdown">
                    <button class="dropbtn" onclick="toggleDropdown(event)">
                        <?= htmlspecialchars($_SESSION['user']['username']) ?>
                    </button>
```

How it looks on website:



INTRODUCTION

Purpose. To master data structures, databases, graphical user interface programming, the application of design patterns, version control, documentation, and testing tools while developing a cohesive domain-specific application.

To achieve the intended goal, the following practical tasks are set:

1. Design a software system:
   1.1 Design the initial system data and output;
   1.2 Define the data structures used in the program;
   1.3 Describe the structure of the software project;
   1.4 Select and apply design patterns when designing the architecture.
2. Develop the software system:
   2.1 Implement data input/output flows;
   2.2 Implement the program's calculation algorithms;
   2.3 Implement the graphical user interface (GUI);
3. Ensure the management of the software system development process and quality assurance:
   3.1 Create automated tests for code validation;
   3.2 Use version control tools for the code.

The development of the programming course project was based on the provided minimum requirements table (see Table 1).

1 Table: Minimum Requirements Table

Minimum requirements:	**Filled in by the teacher**
Adherence to code naming conventions	
The report is free of grammatical or formatting errors	
The code and report are provided on GitLab	
The report contains all the sections of the given template filled out	
Each section of the report clearly indicates where in the code the result is implemented	

It was also based on the evaluation criteria table (see Table2).

2 Table. Evaluation criteria table

Evaluated section (chapter in the report)	**Value**	**5-6**	**7-8**	**9-10**
Database Design (1.1)	5 %	A database with at least 3 tables (each with a minimum of 20 records and at least 3 fields).	A database with at least 4 tables (each with a minimum of 20 records and at least 3 fields). Use multiple types of relationships.	There is reading from the database with at least 4 tables (each with a minimum of 20 records and at least 3 fields). Multiple types of relationships are used. The Lithuanian language character encoding has been properly handled.

Data Objects (1.2.1)	5 %	At least one data object is used, consisting of a minimum of 3 properties.	Multiple data objects are used, or a single composite data object is worked with.	Multiple data objects are used, and at least one of them is a composite data object.
Data Structures (1.2.2)	10 %	One data structure is selected, and its suitability is justified.	Multiple data structures are used, with their own combination defined. Alternatively, there is the possibility to extend them with new objects.	Multiple data structures are used, with their own combination defined, and there is the possibility to extend them with new objects.
Software Project (1.3)	5 %	The software project and the technologies used are described.	The software project and the technologies used are described, and the complete architectural model of the project is specified.	The software project and the technologies used are described, the complete architectural model of the project is specified, and the operational algorithm models are described.
Design Patterns (1.4)	10 %	In the program code, one creation, structural, and behavioral design pattern is applied.	In the program code, 5 design patterns are applied: one each from the creation, structural, and behavioral categories.	In the program code, 7 design patterns are applied: at least 2 from each category: creation, structural, and behavioral patterns.
JPA Implementation (Java Persistence API) (2.1)	5 %	JPA is configured using Hibernate or an alternative framework. At least one entity is created, and CRUD operations are performed.	JPA is configured using Hibernate or an alternative framework. At least one more complex entity structure is created (one entity consists of several others, and one entity must obligatorily reference another, etc.).	JPA is configured using Hibernate or an alternative framework. Several more complex entity structures are created (one entity consists of several others, and one entity must obligatorily reference another, etc.).
DB Queries (2.1)	7 %	All CRUD operations are performed.	Compound queries involving multiple parameters or multiple DB tables are performed.	Compound queries involving multiple parameters and multiple DB tables are performed.
Algorithms (2.2)	8 %	One operation is performed from: Searching for items in a collection, Selection (filtering) of elements in a collection, Sorting items in a collection.	Two operations are performed from: Searching for items in a collection, Selection (filtering) of elements in a collection, Sorting items in a collection.	Three operations are performed: Searching for items in a collection, Selection (filtering) of elements in a collection, Sorting items in a collection.
Graphical user interface (2.3)	10 %	There is a graphical user interface that displays and manipulates the data.	There is a composite graphical user interface that displays and manipulates the data.	There are several different composite GUIs (mobile app, web, etc.) that display and manipulate data.
Testing (3.1)	4 %	The created code is tested with automatic tests (coverage 20%).	The created code is tested with automatic tests (coverage 50%).	The created code is tested with automatic tests (coverage 70%).
	3 %	At least 3 types of assert methods are used.	At least 4 types of assert methods are used.	At least 5 types of assert methods are used.
	3 %	At least 3 types of	At least 4 types of	At least 5 types of annotations are

		annotations are used.	annotations are used.	used.
	4 %	One of the testing categories was implemented: Exception testing, Performance testing, Parametrized tests.	Two of the testing categories were implemented: Exception testing, Performance testing, Parametrized tests.	Three of the testing categories were implemented: Exception testing, Performance testing, Parametrized tests.
Code version control (3.2)	8 %	Minimum 25% weekly code submissions.	Minimum of 50% weekly code submissions.	Minimum of 75% weekly code submissions
Rationale for decisions (List of information sources)	13 %	At least 5 scientific sources are cited during design and implementation.	At least 8 scientific sources are cited during design and implementation.	When designing and implementing, cite at least 10 scientific sources.

1. SOFTWARE SYSTEM DESIGN

1.1. Database Design

The system uses a relational database implemented in MySQL via XAMPP.

The database is designed with four main tables:

Users: Stores user registration information.

Products: Contains information about watches and accessories for sale.

Cart: Stores products added to each user’s cart.

Repairs: Stores uploaded information about completed repairs.

Relationships

One-to-many: One user can have many cart items.

Many-to-many: Users can have multiple products in their cart, and each product can appear in many carts.

One-to-many: A user can submit multiple repair entries.

All tables contain at least 20 entries and are populated with realistic data for testing purposes. The database supports UTF-8 encoding, including Lithuanian characters.

1.2. Data Used

1.2.1. Data Objects

The system uses multiple data objects, each representing a real-world entity:

User object: Stores the logged-in user’s details (first name, last name, email, password).

Product object: Includes name, price, category, and image.

Cart object: Contains an array of Product objects associated with a user.

Repair object: Contains the title and uploaded image of a completed repair.

The Product object is composite, including nested information such as image path and category.

1.2.2. Data Structures

The system uses various data structures:

Arrays: Used to store and manage multiple cart items.

Objects: Represent users, products, and repairs.

Nested objects: The Cart array holds multiple composite Product objects.

The combination of arrays and objects enables dynamic operations such as filtering, adding, removing, and mapping items. The system is also extensible: new product types or cart features can be added with minimal changes to the code.

1.3. Software Project

The software project is a web-based application built with the following technologies:

- HTML5, CSS3, JavaScript (Vanilla) for frontend.
- LocalStorage for data persistence on the client-side
- MySQL for backend data storage (via XAMPP) for backend.
- jsPDF library for PDF generation

The project consists of multiple HTML pages:

- index.html: Homepage for browsing products.
- cart.html: Displays the user´s cart.
- login.html: User login page.
- register.html: User registration page.
- reparations.html: Repair services page.

Architecture:

Frontend: Handles UI interactions, localStorage, PDF generation, and user flow.

Backend: Will use PHP to connect to MySQL and perform CRUD operations.

Operational Flow:

1. User registers and logs in.

2. Browses products and adds to cart.

3. Can clear or buy cart items.

4. Repair images can be uploaded and listed.

5. A downloadable PDF is generated upon checkout.

1.4. Design Patterns

The following design patterns are used or proposed in the system:

Creational Patterns:

Singleton: Ensures there is only one user session at a time (via localStorage).

Factory: Could be used to instantiate Product objects dynamically in future backend integration.

Structural Patterns:

Module Pattern: The script.js organizes multiple related functions and could be separated in modules.

Facade Pattern: The checkout function acts as a facade by calling PDF generation, clearing cart, and showing confirmation.

Behavioral Patterns:

Observer: The dark mode toggle and cart display react to changes in state.

Command: The removeFromCart() function simulates undo logic.

Strategy: The addToCart() function could switch between localStorage or database mode in future enhancements.

These patterns support extensibility, modularity, and reusability across the system.

2. SOFTWARE SYSTEM IMPLEMENTATION

2.1. JPA Implementation (Java Persistence API)

The project uses a MySQL database accessed via XAMPP. While the implementation is currently frontend-only, the system is designed for easy extension with a backend using PHP.

Planned or partial implementation:

A PHP backend will handle CRUD operations (Create, Read, Update, Delete) on the database tables `users`, `products`, `cart`, and `repairs`.
Relationships between tables are respected (e.g., foreign keys for user-cart or user-repair).
Entity structures are designed for extendibility (e.g., composite product objects).

PHP classes will mirror the current frontend objects (User, Product, Cart, Repair) and perform operations accordingly.

2.2. DB Queries

The current system uses localStorage for temporary client-side storage. However, the project is prepared for transition to full database operations via PHP and MySQL.

Planned compound queries:

Join between users and cart to retrieve all products for a user.
Join between cart and products to display full product details.
Filtering and sorting queries based on product category, price, or name.

Example SQL queries:

`SELECT * FROM products WHERE category = 'smartwatch' ORDER BY price ASC;`
`SELECT u.first_name, p.name FROM users u JOIN cart c ON u.id = c.user_id JOIN products p ON c.product_id = p.id;`

2.3. Algorithms

The following algorithmic operations are implemented in JavaScript:
Search: The user can implement a product search function by name.
Filter: Products can be filtered by category (e.g., Smartwatch, Woman).
Sort: Products can be sorted by price ascending or descending.

These operations are performed on the array of product objects stored in localStorage or loaded from the backend.

2.4. Graphical User Interface (GUI)

The application uses a composite web graphical user interface built with HTML, CSS, and JavaScript.

The GUI includes:

- Product listing with modal previews.
- Login and registration forms.
- Cart system with remove/clear/buy buttons.
- Dark mode toggle.
- Repair image upload with preview and delete option.

The interface is responsive and user-friendly, adapting to different screen sizes. Buttons have visual feedback, hover animations, and custom styles. Data is displayed dynamically using JavaScript, creating an interactive experience.

Each page acts as a different view, making the interface composite and modular.

3. SOFTWARE SYSTEM QUALITY ASSURANCE

3.1. Testing

The system includes a comprehensive suite of unit tests written in PHP using PHPUnit. These tests ensure the functionality and reliability of the system's core features. Below is an overview of the test files and their purposes:

- LoginRegisterTest.php
    - Purpose: Tests the user registration and login functionality.
    - Key Tests:
        1. Verifies successful user registration with valid data.
        2. Ensures user authentication works with correct credentials.
        3. Confirms authentication fails with invalid credentials.

- ReparationsTest.php
    - Purpose: Tests the repair request submission and retrieval functionality.
    - Key Tests:
        1. Validates successful submission of a repair request with all required fields.
        2. Ensures repair requests can be fetched from the database.

- GeneratePdfTest.php
    - Purpose: Tests the PDF generation functionality for cart orders.
    - Key Tests:
        1. Ensures an error is displayed when attempting to generate a PDF with an empty cart.
        2. Confirms an error is displayed if the user is not logged in.
        3. Verifies successful PDF generation when the cart contains items and the user is logged in.

- UpdateUserTest.php
    - Purpose: Tests the user profile update functionality.
    - Key Tests:
        1. Verifies successful updates to user details (username, email, password).
        2. Detects conflicts when attempting to update to an already-taken username or email.

- CartTest.php
    - Purpose: Tests the shopping cart functionality.
    - Key Tests:
        1. Ensures products can be added to the cart.
        2. Verifies products can be removed from the cart.
        3. Confirms the cart can be cleared entirely.
        4. Measures the performance of cart operations with a large number of items.

- DatabaseTest.php
    - Purpose: Tests the database connection functionality.
    - Key Tests:

1. Verifies a successful connection to the database.
2. Ensures an exception is thrown for invalid database connection credentials.

- UserTest.php
    - Purpose: Tests the user-related operations in the system.
    - Key Tests:
        1. Verifies successful user registration.
        2. Checks the availability of usernames and emails.
        3. Ensures user authentication works with valid credentials and fails with invalid ones.
        4. Confirms exceptions are thrown for invalid user data.

3.2. Code Version Control

The project was completed in a short time frame, and therefore full Git version tracking was not used throughout development. However, GitLab was used to store the final version of the project and to submit the code and documentation.

Git Usage Overview

While continuous commits were not made, a Git repository was created to upload and organize all project files:

GitHub Repository: https://github.com/Dinama-20/Software-Development.git

The final version of the system, including all .php, .js, .css, .sql, and .docx files, was uploaded and structured appropriately.

Lessons and Future Use

This experience highlighted the importance of using Git from the beginning of a project, even for individual assignments. In future projects, version control will be used to:

- Track changes over time and prevent accidental data loss.
- Document the development process through commit messages.
- Collaborate more efficiently if needed.

CONCLUSIONS

The development of the "Oñate Watch and Jewelry" web system has provided an opportunity to apply and integrate a wide range of software engineering concepts. From data modeling and algorithm design to user interface development and testing, each stage of the project was carefully designed to ensure functionality, scalability, and user experience.

Key Achievements:

- Developed a fully functional online store with product listing, cart, user login, and repair request functionality.
- Designed a relational database with four interconnected tables, supporting realistic use cases.
- Implemented product filtering, search, and sorting algorithms in JavaScript.
- Structured the interface with reusable code and modular PHP pages.
- Applied multiple design patterns, enhancing the maintainability of the code.
- Created a dynamic and responsive user interface, including dark mode and PDF export features.

The main technical challenges included ensuring session persistence using PHP, syncing cart state across pages, and preparing the system for future MySQL integration. Minor issues, such as the dark mode toggle and cart data not saving, were resolved through debugging and refactoring.

LIST OF REFERENCES AND OTHER SOURCES OF INFORMATION

1. PHP Manual. PHP: Hypertext Preprocessor Official Documentation. Available at: https://www.php.net/manual/en/

2. MySQL Documentation. MySQL 8.0 Reference Manual. Oracle. Available at: https://dev.mysql.com/doc/

3. Mozilla Developer Network (MDN). JavaScript Reference. Mozilla. Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript

4. W3Schools. HTML, CSS, and JavaScript Tutorials. Available at: https://www.w3schools.com/

5. jsPDF GitHub Repository. Parallax/jsPDF. Available at: https://github.com/parallax/jsPDF

6. Apache Friends. XAMPP Official Website. Available at: https://www.apachefriends.org/

7. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

8. Stack Overflow. Community-driven Q&A for coding problems. Available at: https://stackoverflow.com/

9. Git SCM. Pro Git Book. Available at: https://git-scm.com/book/en/v2

10. Mozilla Developer Network (MDN). Window.localStorage - Web APIs. Available at: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage