TECHNOLOGIJŲ FAKULTETAS
INŽINERIJOS IR INFORMATIKOS KATEDRA

TEMOS PAVADINIMAS

Kursinis darbas

Informatikos studijų programos
valstybinis kodas 6531BX004
Informatikos studijų krypties

Autorius Vardas Pavardė

(parašas) *(data)*

Vadovas doc. dr. Aleksas Narščius

(parašas) *(data)*

Klaipėda, 2023

CONTENT

LIST OF TABLES 2
LIST OF FIGURES 3
INTRODUCTION 4
1. SOFTWARE SYSTEM DESIGN 7
1.1. DATABASE DESIGN 7
1.2. DATA USED 7
1.2.1. DATA OBJECTS 7
1.2.2. DATA STRUCTURES 7
1.3. SOFTWARE PROJECT 8
1.4. DESIGN PATTERNS 8
2. SOFTWARE SYSTEM IMPLEMENTATION . **ERROR! BOOKMARK NOT DEFINED.**
2.1. JPA IMPLEMENTATION (JAVA PERSISTENCE API)**ERROR! BOOKMARK NOT D**
2.2. DB QUERIES ..... **ERROR! BOOKMARK NOT DEFINED.**
2.3. ALGORITHMS ..... **ERROR! BOOKMARK NOT DEFINED.**
2.4. GRAPHICAL USER INTERFACE (GUI)**ERROR! BOOKMARK NOT DEFINED.**
3. SOFTWARE SYSTEM QUALITY ASSURANCE 12
3.1. TESTING 12
3.2. CODE VERSION CONTROL 12
CONCLUSIONS 14
LIST OF REFERENCES AND OTHER SOURCES OF INFORMATION 15

LIST OF TABLES

LIST OF FIGURES

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 MySQL database, developed and managed via XAMPP. The schema includes four main tables:

- Users: Stores user registration and login data.
- Products: Contains product listings, including watches and accessories.
- Cart: Stores the relationship between users and products added to their shopping cart.
- Repairs: Contains information and images related to repair requests submitted by users.

Relationships implemented:

- One-to-many: A user can submit multiple repair entries.
- One-to-many: One user can have many cart entries.
- Many-to-many: Products can appear in multiple user carts, and users can have multiple products.

Each table contains at least 20 records and more than 3 fields. Data supports UTF-8 encoding to handle Lithuanian and other special characters.

1.2. Data Used

1.2.1. Data Objects

The project includes multiple data objects reflecting real-world entities:

- User: First name, last name, email, and password.
- Product: Name, price, category, and image path.
- Cart: Stores user-related product selections (array of Product objects).
- Repair: Includes a title and the uploaded image for completed repairs.

The Product object is composite, as it includes nested data (category and image).

1.2.2. Data Structures

To manage and manipulate data, the system uses:

- Arrays: Store multiple cart items and product data for filtering, sorting, and rendering.
- Objects: Represent user, product, cart, and repair entities.
- Nested objects: Example: Cart contains arrays of Product objects.

The structure is designed to be extensible. New product types, features (like discounts), or cart-related logic can be added with minimal changes to the structure.

1.3. Software Project

The application is a web-based online store named Oñate Watch and Jewelry, developed using:

- Frontend: HTML5, CSS3, and Vanilla JavaScript.
- Backend (planned/partial): PHP (procedural) via XAMPP, connecting to MySQL.
- Client-side storage: LocalStorage is used to store session data and cart contents.
- Library: jsPDF is used for generating downloadable PDF receipts.

Pages in the project:

- index.php: Product listing and interaction.
- login.php and register.php: User authentication.
- cart.php: Shows selected products, allows removal or purchase.
- reparations.php: Upload and display of repair entries.
- logout.php: Ends session and returns to home.

Architecture:

- Frontend: Handles all dynamic UI behavior — dark mode, product filtering, cart updates.
- Backend (in progress): Receives AJAX requests for CRUD operations and session handling.

Operational Flow:

- A user registers or logs in.
- Products are browsed and added to the cart.
- Cart can be cleared or finalized.
- Repair entries with images can be uploaded.
- A PDF summary is generated for the purchase.

1.4. Design Patterns

The system applies several design patterns across the frontend logic, ensuring better structure and future expandability:

- Creational Patterns
    - Singleton: Ensures only one active user session at a time (handled via localStorage).
    - Factory: Product creation logic is ready to be generalized for backend instantiation.

- Structural Patterns
    - Module Pattern: Functions are grouped and logically organized inside script.js.
    - Facade: The checkout() function centralizes PDF generation, cart clearing, and confirmation.

- Behavioral Patterns
    - Observer: Cart updates and dark mode toggling react to state changes.
    - Command: Functions like removeFromCart() simulate undo-like behavior.
    - Strategy: addToCart() can switch from localStorage to DB logic in the future.

These patterns collectively improve the system's maintainability, scalability, and separation of concerns.

2. SOFTWARE SYSTEM IMPLEMENTATION

2.1. JPA Implementation (Java Persistence API)

While JPA is Java-specific, the equivalent in this PHP project involves:

- Defining entities (users, products, cart, repairs) via MySQL tables.
- Performing CRUD operations via PHP and mysqli or PDO queries.
- Implementing entity relationships through foreign keys.

Planned or partial backend implementation in add_to_cart.php and similar files allows interaction with session data and prepares for real DB integration.

2.2. DB Queries

Although the current system operates using client-side data storage via localStorage, the database structure has been fully prepared and the backend is partially implemented using PHP. The database includes four tables: users, products, cart, and repairs, with appropriate relationships defined using foreign keys.

Planned SQL queries are designed to support the following functionalities:

- Filtering products by category and sorting by price.
- Retrieving all products added by a specific user.
- Displaying joined information across multiple tables for administrative or summary purposes.

2.3. Algorithms

The system implements several algorithmic operations using JavaScript to provide dynamic interaction and enhance user experience. These algorithms operate on an array of product objects and allow real-time manipulation of the data on the client side.

The following operations are implemented:

- Search**:** Products can be searched by name using a case-insensitive match.
- Filter: Products are filtered by category (e.g., Smartwatch, Woman, Man).
- Sort: The user can sort products by ascending or descending price.

These operations are fully integrated into the user interface, allowing seamless search, category filtering, and sorting without page reloads. The system's data structure is designed for extensibility, making it easy to introduce new algorithmic operations in the future.

2.4. Graphical User Interface (GUI)

The application features a composite web-based graphical user interface, developed using HTML, CSS, JavaScript, and PHP. The interface is designed to be intuitive, modular, and fully responsive across various devices.

Each section of the system is implemented as a separate PHP file, including:

- index.php: Displays all products with options to add to cart and view modals.
- cart.php: Shows selected products with options to remove, clear, or purchase.
- login.php and register.php: Handle user authentication.
- reparations.php: Allows users to upload and preview repair entries.
- logout.php: Ends the user session and redirects to the home page.

The GUI includes the following features:

- Dynamic product display using JavaScript, including modals for detail view.
- Interactive cart management with instant update and session persistence.
- Repair entry upload with image preview and deletion.
- Dark mode toggle, allowing users to switch themes dynamically.
- PDF generation for checkout using the jsPDF library.
- Responsiveness, supporting mobile and desktop resolutions with adaptive layout.

Buttons include hover effects and active states. JavaScript handles data binding, DOM updates, and client-side storage. The user interface provides immediate feedback, contributing to a professional and user-friendly experience.

3. SOFTWARE SYSTEM QUALITY ASSURANCE

3.1. Testing

Due to the individual scope of the project, automated testing was not fully implemented. However, manual testing was performed during development to ensure the core functionalities worked as expected in the browser.

Manual Testing Process

All core features were tested manually using the browser environment:

- Login and registration: Verified correct input handling, error messages for empty fields, and redirect after successful login.
- Cart system: Tested product addition, removal, and clearing of the cart.
- Product filtering and sorting: Validated behavior of search, category filters, and price sorting.
- Dark mode toggle: Checked visual changes and interaction logic.
- Repair entry upload: Confirmed image preview and delete functionality.

Planned but Not Implemented

Although no automated testing tools (like Jest or PHPUnit) were used, the project structure supports future testability. For instance:

- JavaScript functions are modular and could be tested with frameworks like Jest.
- PHP session handling can later be tested using PHPUnit.
- Frontend DOM manipulation and event logic could be validated with tools like Cypress or Playwright.

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:

GitLab 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 a 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