|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | |
| **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, 2025 | | | | |

**CONTENT**

[LIST OF TABLES 2](#_Toc191028959)

[LIST OF FIGURES 3](#_Toc191028960)

[INTRODUCTION 4](#_Toc191028961)

[1. SOFTWARE SYSTEM DESIGN 7](#_Toc191028962)

[1.1. Database Design 7](#_Toc191028963)

[1.2. Data Used 7](#_Toc191028964)

[1.2.1. Data Objects 7](#_Toc191028965)

[1.2.2. Data Structures 7](#_Toc191028966)

[1.3. Software Project 7](#_Toc191028967)

[1.4. Design Patterns 7](#_Toc191028968)

[2. SOFTWARE SYSTEM IMPLEMENTATION 8](#_Toc191028969)

[2.1. JPA Implementation (Java Persistence API) 8](#_Toc191028970)

[2.2. DB Queries 8](#_Toc191028971)

[2.3. Algorithms 8](#_Toc191028972)

[2.4. Graphical User Interface (GUI) 8](#_Toc191028973)

[3. SOFTWARE SYSTEM QUALITY ASSURANCE 9](#_Toc191028974)

[3.1. Testing 9](#_Toc191028975)

[3.2. Code Version Control 9](#_Toc191028976)

[CONCLUSIONS 10](#_Toc191028977)

[LIST OF REFERENCES AND OTHER SOURCES OF INFORMATION 11](#_Toc191028978)

# 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. Design the initial system data and output;    2. Define the data structures used in the program;    3. Describe the structure of the software project;    4. Select and apply design patterns when designing the architecture. |
| 1. Develop the software system:    1. Implement data input/output flows;    2. Implement the program’s calculation algorithms;    3. Implement the graphical user interface (GUI); |
| 1. Ensure the management of the software system development process and quality assurance:    1. Create automated tests for code validation;    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 annotations are used. | At least 4 types of annotations are used. | At least 5 types of annotations are 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. |

# SOFTWARE SYSTEM DESIGN

## 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.

## Data Used

## 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.

## 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.

## Software Project

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

- HTML5, CSS3, JavaScript (Vanilla)

- LocalStorage for data persistence on the client-side

- MySQL for backend data storage (via XAMPP)

- jsPDF library for PDF generation

The project consists of multiple HTML pages: index.html, cart.html, login.html, register.html, and reparations.html, each with a consistent design.

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.

## 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.

# SOFTWARE SYSTEM IMPLEMENTATION

## 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.

## 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;`

## 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.

## 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.

# SOFTWARE SYSTEM QUALITY ASSURANCE

## Testing

Text.

## Code Version Control

Text.

# CONCLUSIONS

1. Conclusion.
2. Conclusion.
3. Conclusion.

|  |
| --- |
|  |

# LIST OF REFERENCES AND OTHER SOURCES OF INFORMATION