VILNIUS UNIVERSITY

FACULTY OF MATHEMATICS AND INFORMATICS

SOFTWARE ENGINEERING STUDY PROGRAMME

Laboratory work 2

Evaluation and Change Log

Gustas Šukys, Adomas Šileikis, Edvinas Burba, Leonardas Sinkevičius, Augustas Budnikas

Supervisor : Lekt. Vasilij Savin

Vilnius
2025

Contents

1 Architectural Evaluation & General Feedback

This section evaluates the alignment between the proposed API design and the project's operational requirements, highlighting key structural inefficiencies.

1.1 Domain-Centric vs. Operational Design

The current API follows a strict domain-driven structure (e.g., `price-modifier`, `order`) suited for generic public consumption, but is poorly optimized for internal UI workflows. This misalignment introduces several overheads:

- **High Integration Complexity:** The lack of aggregate endpoints forces the client to orchestrate multiple domain-specific calls for single logical actions (e.g., "Create Order"). Adhering to this design would necessitate a Backend-for-Frontend (BFF) layer, such as GraphQL, to manage consumption complexity.
- **Process Inefficiency:** The design favors "Microservices purity" over performance. By splitting simple transactional updates across multiple interdependent APIs, the architecture increases latency and complicates the integration compared to a standard process-oriented approach.

1.2 Data Modeling & Relational Logic

The underlying data structures exhibit inconsistencies that hinder system scalability:

- **Relational Gaps:** Many complex business requirements are currently addressed via structural workarounds. These could be resolved more efficiently by implementing a standardized Role-Based Access Control (RBAC) system.
- **Redundancy:** A lack of clear relational definitions between Businesses and Accounts has resulted in fragmented query logic and redundant data fields, creating risks for data synchronization.

1.3 Overall Design Evaluation

Following a comprehensive review of the provided documentation, the overall quality and readiness of the API design is rated at **`7/10`**.

While the documentation provides a solid domain-driven foundation and clear individual resource definitions, it requires significant refinement to be "production-ready" for our specific implementation. The score reflects a well-structured theoretical approach that lacks the practical "glue" and relational optimization necessary for efficient frontend consumption and high-performance business logic.

2 API changes

This section details the necessary refactoring of the API contracts to adhere to industry standards, improve security, and resolve logical inconsistencies found in the initial specifications.

2.1 General RESTful Conventions & Naming

- **Inconsistent URI Resource Naming:** The current endpoints use singular nouns (e.g., `/account`, `/function`). Standard RESTful conventions dictate the use of plural nouns for resource collections. The endpoints must be renamed (e.g., `/accounts`) to imply resource collections and align with best practices.

2.2 Account Management APIs

- **Missing Initialization Workflow:** The `POST /account` endpoint is restricted to "Create worker account," lacking a mechanism to provision high-level administrator accounts or bootstrap the system.

- **Ambiguous Collection Scope:** The `GET /account` description ("Get all accounts") is undefined in scope. It is unclear if this retrieves system-wide accounts or is scoped to a specific business context, representing a potential security risk.

- **Missing Self-Reference Endpoint:** Account retrieval currently relies solely on ID-based lookups (`/account/{id}`). A standard `/account/me` endpoint is required to allow authenticated users to retrieve their own profile information without needing to query their own database primary key.

- **Missing User-Context Association:** There is no discoverable path for a standard user to retrieve the Business ID(s) they are associated with. An endpoint is required to allow users to fetch their active business associations.

2.3 Role & Business Management

- **Incorrect Resource Hierarchy:** Roles are currently nested under `/account/role`. Semantically, roles are bound to a Business entity, not a generic Account. These endpoints must be refactored to exist under the Business domain (e.g., `/business/{businessId}/roles`) to accurately reflect the domain relationship.

- **Inefficient Data Loading (N+1 Problem):** Roles and Functions are retrieved via separate endpoints, necessitating multiple round-trips. The `GET` request for a Role should support eager loading (e.g., `?expand=functions`) to include associated functions in the payload.

- **Redundant Granularity:** The endpoint `GET /account/role/{roleId}` offers minimal utility outside the context of a business. The architecture should prioritize aggregate retrieval (Roles within a Business) rather than isolated resource management.

2.4 Data Model & Schema

- **Non-Standard Primary Key Nomenclature:** The schema uses the term `ident`. This violates established database normalization standards. The standard naming convention `id` (or `Id`) should be used to ensure compatibility with standard libraries and developer expectations.
- **Schema Redundancy:** The model includes an `identOwnerAccount` field while also implementing a "Business Owner" Role. This introduces data redundancy. Ownership should be determined dynamically via Role assignments to adhere to the Single Source of Truth (SSOT) principle.

2.5 Tag Management

- **Missing Association Endpoints:** The Tag API currently supports only basic CRUD operations. However, it lacks endpoints that allow tags to be applied to or removed from domain resources such as Services, Items and Item Options.
- **Missing Reverse Lookup Endpoints:** The API does not provide any mechanism to retrieve domain resources by tag (e.g., fetching all Services or Items associated with a given tag). Without such endpoints, tags cannot be effectively used for filtering, categorization, or discovery purposes.
- **Limited Functional Purpose:** Due to the absence of both association and reverse lookup endpoints, tags currently serve no practical operational role beyond static metadata storage, which contradicts their intended use as dynamic classification tools.

2.6 Order APIs

- **Inconsistent Identifier Naming:** The Order API uses both `{id}` and `{orderId}` to refer to the same resource. This inconsistency violates RESTful naming conventions and increases the risk of implementation errors on the client side. A single, consistent identifier name should be used across all endpoints.
- **Missing Order Deletion Endpoint:** The API does not provide an endpoint for deleting or cancelling an order (e.g., `DELETE /order/{orderId}`). This restricts proper order lifecycle management and limits error recovery for invalid or abandoned orders.
- **Missing Payment Linkage Management:** The specification does not clearly define endpoints for associating, retrieving, or managing payments related to an order. Without explicit linkage mechanisms, it is unclear how payment entities are connected to orders, making payment state tracking and reconciliation unreliable.

2.7 Service Model and APIs

- **Missing Service Availability Definition:** The Service model does not define a provisioning time range (e.g., available from/to hours or dates). Without this information, the system cannot deterministically calculate or render available time slots in the UI. The model must include explicit availability constraints, including start time and end time.
- **Missing Provisioning Interval:** The Service definition lacks a provisioning interval (e.g., 15, 30, 60 minutes). This value is required to generate discrete reservation slots and must be modeled as a first-class attribute of the Service entity.
- **Overly Restrictive Employee Assignment:** The current data model allows only a single servicing employee per Service. In most real-world scenarios (e.g., barbershops, salons, clinics), multiple employees are capable of performing the same Service. The model must support a many-to-many relationship between Services and Accounts (employees).
- **Missing Service–Employee Management Endpoints:** The API does not specify how employees are associated with or removed from a Service. To address this gap, additional endpoints were introduced (`POST /service/{serviceId}/employee/{accountId}` and `DELETE /service/{serviceId}/employee/{accountId}`) to explicitly manage these associations.

2.8 Reservations Model and APIs

- **Redundant User References:** The Reservation model contains two fields, `identUser` and `identServicingAccount`, which both appear to reference the employee associated with the reservation. These fields were merged into a single canonical reference to represent the account that will service that customer.
- **Missing Payment Association:** The original specification does not define how Payments are linked to Reservations. To resolve this integration gap, a dedicated endpoint was introduced (`/reservation/{reservationId}/payment/{paymentId}`) to explicitly associate a Payment with a Reservation.

2.9 Gift Card Management

- **Missing Domain Model and APIs:** Although Gift Cards are referenced conceptually in the project documentation, no concrete data model or API endpoints were defined. A complete Gift Card model was introduced along with standard CRUD endpoints.
- **Payment Integration:** Gift Cards were integrated into the Payment flow, allowing Payments to be created using an existing Gift Card balance. This requires explicit validation of available balance and transactional consistency to prevent overuse or double-spending.

3 UI changes

This section documents the specific changes and refinements made to the user interface during the technical design phase. These updates focus on alignment with the design system, improved user workflows, and the completion of missing interface components.

3.1 Core System Flows

As the initial documentation did not provide designs for administrative and authentication modules, the following flows were implemented from scratch to ensure a fully operational ecosystem:

- **Authentication and Onboarding:** Full implementation of user login, registration, and password recovery flows.
- **Business Management:** Development of the business creation wizard and organizational settings.
- **User and Access Control:** UI for managing employees and defining granular roles and permissions.
- **Resource and Catalog Management:** Comprehensive interfaces for managing services, tags, items, item options, and price modificators.

3.2 General Changes

- **Orders Page:** The order cards have been adapted to the official design library. They now provide more comprehensive information at a glance to enhance usability and the overall user experience.
- **Tables Page:** This page has been completely removed from the project scope as it was not referenced in any other part of the system documentation or logic.
- **Global Implementation:** All other application pages and modals not specified in the initial documentation have been implemented from scratch to ensure a complete and functional user journey.

3.3 Order Editing Page

Several modifications were made to the Order Editing interface to improve functional clarity:

- **Search and Tags:** The search bar and tags have been unified into a single horizontal line, with tags transitioned into a dropdown format. This change ensures alignment with the global system design and optimizes screen real estate.
- **Content Accuracy:** Proper content structures for items have been defined and implemented, as this data was missing in the original wireframe stage.

- **Billing Actions:** Buttons related to billing have been relocated to the primary action area to centralize core functions and improve the user flow.
- **Order Summary:** The items list within the order summary was updated to include better placement for edit buttons. Additionally, a new UI component was added to display item options, addressing UX gaps and missing content in previous versions.

3.4 Modals and Windows

- **Order Item Editing Window:** The selection of item options has been updated to a dropdown menu. The overall layout of the window was restructured to provide a better visual hierarchy and maintain consistency with the full system UI.