GiftCard and service charge added to resource enum.

OrderId column added to OrderItems table

Business logic: pending orders CAN be changed

OrderItem table instead of lists for taxes and service charges have Tax.Id and ServiceCharge.Id

Removed TransactionId from Refund and Payment

Removed UserId from Payment and Refund.

Removed PaymentId from Refund. Refund is only available for the full order not separate Payments.

Made implementation work like specified in the database. A single login for both employee and management using email. Did not create a separate id field for employees.

Added working hours and nextPaymentDue date to business model

Owner registration: POST /auth/register-owner added.
The registration was not present in API.yaml in any shape of form. So a custom endpoint was created.

Added Giftcard FK to Payment

Beauty reservations gap: no data model/endpoints for staff availability/slots, so the mockup's "choose available times" cannot be generated from the current model.

Variation support: ItemVariationOptionId FK added to OrderItem.

Changed taxes to be attached to Item instead of OrderItem entity. Taxes should be applied to specific items and not an instance of the item that OrderItem is.

Reservations and orders are implemented as separate, unlinked pages/flows, while the document's mockups and data model assume a synchronized flow (reservations leading into orders, showing linked items/services). This separation doesn't align with the intended end-to-end reservation → order flow in the data model. Misleading document mockups lead to this implementation

Catalog service is barely mentioned in high level architecture with no real explanation for what it contains or what it really does.

Payment/refund model gaps: documented payment/refund fields (transaction IDs, refunded-by user, payment linkage) are not present in the current EF models/DTOs.
Login UX mismatch: the UI mockups and PDF describe an employee-ID login flow and a pick user group screen, but the current frontend uses email/password for all users since the database model did not have a separate Employee id stored in it.

Third-party integrations not implemented: Stripe, Twilio SMS, and third-party gift card integrations described in the PDF are not implemented; payments and orders are recorded internally.

Update is blocked only for Closed orders. Pending orders are still updatable.

API.yaml mixes plural collection paths with singular item-by-id paths:
/items vs /item/{itemId}
/taxes vs /tax/{taxId}
/refunds vs /refund/{refundId}
/roles vs /role/{roleId}
Our implemented API mostly uses plural resources with /{id} (e.g., /items/{id}, /taxes/{id}),

PDF defines a detailed permission catalog (BUSINESS_VIEW, ORDER_CLOSE, etc.)
API.yaml defines Permission.code as CREATE/RETRIEVE/UPDATE/DELETE

Item type naming: PDF describes item type as product or service, while API.yaml defines item or service
Implementation: the API and frontend use ItemType.Product / ItemType.Service (serialized as product / service via the configured enum converter)

Order discount representation: PDF models discounts as a managed entity and relationship, while API.yaml models Order.discount as a number
Implementation: orders reference discounts via DiscountId (and optionally return a Discount object); order items can also reference discounts; totals are computed server-side

Payment/refund fields: PDF includes provider/created-by actors for payments and refunds, while API.yaml focuses on transactionId (and omits provider/created-by fields in the payment schema)
Implementation: payments have a Provider and BussinesId but no TransactionId/CreatedBy fields; PaidAt is set server-side; refunds are not linked to payments and the frontend refund flow is implemented by patching the order status to Refunded

Super admin and Business owner role separations were unclearly documented.
Implementation:

1. Super admins are seeded from configuration, created under a special "Platform" business, and assigned a SuperAdmin role with MANAGE_ALL permission both in production and in development.

2. Business owners are represented by Business.OwnerId and are assigned a per-business default owner role during POST /auth/register-owner
3. The frontend treats MANAGE_ALL as "super" (IsSuper) and uses the owner id match for "owner" (IsOwner); super admins can assume a business context

- Overall evaluation: 8