

# Курсовая работа

## по дисциплине

### Информационные системы

Выполнили: Кравцов Вадим Владиславович

Трофимов Владислав Дмитриевич

Группа: Р3314

Преподаватель: Коновалов Арсений Антонович

## Этап 1

### Текстовое описание предметной области

Информационная система для управления холодильником позволит вести учёт всех продуктов, хранящихся в общем холодильнике, включая информацию о владельце, дате помещения и сроке годности. Она обеспечит справедливое распределение полок между жильцами, поможет избежать хаоса и перегрузки пространства, а также позволит отслеживать, какие продукты скоро испортятся. Кроме того, система предоставит жильцам удобные уведомления и напоминания о необходимости использования или удаления продуктов.

### Зачем нужна ИС, какие задачи она позволяет решить.

#### Учитывать продукты

- Вносить продукты в систему при помещении в холодильник (с указанием владельца, даты, срока годности).
- Сканировать штрих-код или вручную добавлять продукт в базу.

#### Распределять пространство

- Холодильник будет разбит на секции/ячейки, и у студентов будет возможность выбора свободного места, где он сможет оставить свои продукты.

#### Контролировать срок годности

- Отслеживать «срок жизни» продуктов.
- Отправлять уведомления владельцу о том, что продукт скоро испортится.
- Напоминать о необходимости выбросить просрочку.

#### Предотвращать конфликты и кражи

- Фиксировать владельца каждого продукта.
- Предоставлять возможность «отметить исчезновение» продукта и вести журнал инцидентов.

#### **Повышать общую дисциплину и удобство**

- Визуализировать занятость холодильника (свободные/занятые полки).

## **Требования**

### **Функциональные требования**

**FR-1.** Система должна предоставлять регистрацию и аутентификацию пользователей через логин/пароль.

**FR-2.** Система должна обеспечивать управление профилем жильца, включая имя, комнату и настройки уведомлений (включено/выключено).

**FR-3.** Система должна предоставлять модератору возможность управления холодильниками и зонами, включая создание и редактирование зон (полки, объём/ёмкость, правила хранения).

**FR-4.** Система должна позволять добавлять продукты с указанием названия, категории, срока годности и зоны.

**FR-5.** Система должна позволять перемещать и редактировать продукты, включая смену зоны, продление срока с обоснованием, а также отметку «съеден/забран/utiлизирован».

**FR-6.** Система должна отправлять уведомления через пуш-уведомления или e-mail о сроках хранения продуктов, выполняя ежедневную проверку.

**FR-7.** Система должна предоставлять «карту холодильника» с визуализацией занятости по зонам и фильтрами (моё/чужое/просрочка).

**FR-8.** Система должна предоставлять модератору и администратору отчёты и аналитику по просрочкам, загруженности зон, инцидентам по типам и времени, а также активности жильцов.

**FR-9.** Система должна предоставлять администратору управление пользователями и ролями, включая приглашения, блокировки и назначение модераторов.

**FR-10.** Система должна вести журнал действий для аудита ключевых операций.

**FR-11.** Система должна обеспечивать обновление информации о занятости зон и ленте инцидентов в реальном времени (WebSocket/SSE).

### **Нефункциональные требования**

**NFR-1.** Система должна обеспечивать производительность: отклик API 95-го перцентиля ≤ 300 мс при 100 rps, а обновления по WebSocket доставляться менее чем за 1 секунду.

**NFR-2.** Система должна обеспечивать надёжность: ежедневные резервные копии БД и целостность данных (транзакционность CRUD-операций).

**NFR-3.** Система должна обеспечивать безопасность: хранение паролей, ролевую модель и аудит ключевых операций.

**NFR-4.** Система должна обеспечивать доступность и удобство использования: адаптивный интерфейс для мобильных и десктопных устройств.

**NFR-5.** Система должна обеспечивать логирование и мониторинг: централизованные логи с уровнями INFO/WARN/ERROR.

## Основные прецеденты (Use Cases)

ID: 1

- **Краткое описание:** Регистрация и вход пользователей в систему.
  - **Главные акторы:** Жилец, Админ
  - **Второстепенные акторы:** —
  - **Предусловия:** Есть инвайт-код холодильника или общая регистрация
  - **Основной поток:** Ввод e-mail/пароля → подтверждение → привязка к холодильнику
- 

ID: 2

- **Краткое описание:** Добавление нового продукта в холодильник.
  - **Главные акторы:** Жилец
  - **Второстепенные акторы:** —
  - **Предусловия:** Жилец зарегистрирован в системе
  - **Основной поток:** Скан QR → ввод названия/категории → выбор зоны → срок годности → «замок» (опц.) → сохранить → продукт отображается владельцу и агрегированно всем
- 

ID: 3

**Краткое описание:** Уведомление жильцов о приближении срока годности продуктов.

- **Главные акторы:** Система
  - **Второстепенные акторы:** Жильцы
  - **Предусловия:** Продукты добавлены
  - **Основной поток:** Ночной батч проверяет сроки → отправка уведомлений T-3/T-1/0 → подсветка на карте; альтернативно пользователь продлевает срок с комментарием
- 

ID: 4

- **Краткое описание:** Изменение зоны или срока хранения продукта.
- **Главные акторы:** Жилец

- **Второстепенные акторы:** —
  - **Предусловия:** Продукт добавлен
  - **Основной поток:** Открыть карточку → сменить зону/срок → сохранить; исключение: зона переполнена
- 

## ID: 5

- **Краткое описание:** Создание и регистрация инцидента с продуктом.
  - **Главные акторы:** Жилец
  - **Второстепенные акторы:** Модератор
  - **Предусловия:** —
  - **Основной поток:** Создать инцидент (тип, описание, фото) → уведомить модератора → инцидент в статусе «новый»
- 

## ID: 6

**Краткое описание:** Просмотр отчётов и аналитики по холодильнику.

- **Главные акторы:** Модератор, Админ
  - **Второстепенные акторы:** —
  - **Предусловия:** Данные о продуктах, инцидентах и уборках доступны
  - **Основной поток:** Выбрать период → отчёты (просрочки, занятость, инциденты) → экспорт
- 

## ID: 7

- **Краткое описание:** Управление пользователями и ролями в системе.
- **Главные акторы:** Админ
- **Второстепенные акторы:** Модератор, Жильцы
- **Предусловия:** Пользователи зарегистрированы
- **Основной поток:** Пригласить/заблокировать → назначить модератора → сброс пароля

## Стек

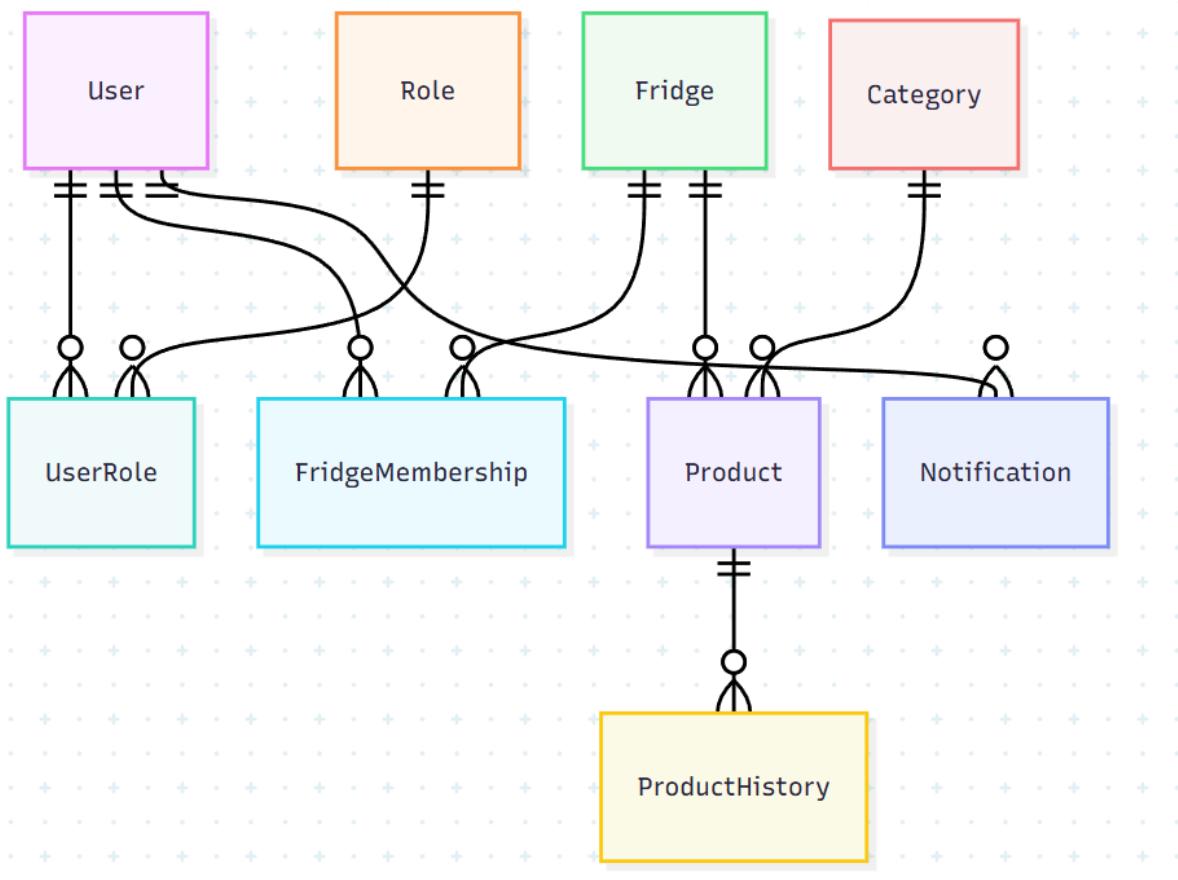
- Front: Thymeleaf
- Backend: Spring MVC
- БД: PostgreSQL

## Этап 2

### ER-модель

#### Сущности и ключевые атрибуты

1. **User (Жилец/Пользователь)**  
user\_id (PK), email (U), пароль\_хэш, имя, комната, notif\_email\_on (bool), notif\_push\_on (bool), created\_at, blocked\_at (NULL)
2. **Role (Роль системы)**  
role\_id (PK), code (U) [ADMIN, MODERATOR, USER], name
3. **UserRole (Пользователь↔Роль, M:N)**  
(PK user\_id,role\_id) — связь User–Role
4. **Fridge (Холодильник)**  
fridge\_id (PK), name, location, created\_at, invite\_required (bool)
5. **FridgeMembership (Пользователь↔Холодильник, M:N с атрибутами)**  
(PK fridge\_id,user\_id), is\_moderator (bool), joined\_at, left\_at (NULL)
6. **Zone (Зона/полка/ящик)**  
zone\_id (PK), fridge\_id (FK), name, capacity\_units (INT), capacity\_volume\_l (NUMERIC, NULL), is\_active (bool), sort\_order
7. **Category (Категория продукта)**  
category\_id (PK), name (U), perishable\_days\_default (INT, NULL)
8. **Product (Конкретная единица продукта)**  
product\_id (PK), owner\_id (FK→User), zone\_id (FK→Zone), name, category\_id (FK), barcode (NULL), expires\_at (DATE/TIMESTAMP), placed\_at, locked (bool, «замок»), status [ACTIVE|EATEN|TAKEN|DISPOSED|EXPIRED]
9. **ProductHistory (История изменений продукта)**  
history\_id (PK), product\_id (FK), event\_type [MOVE|EXTEND|STATUS|EDIT], from\_zone\_id (NULL), to\_zone\_id (NULL), old\_expires\_at (NULL), new\_expires\_at (NULL), comment, actor\_id (FK→User), created\_at
10. **Notification (Журнал отправленных уведомлений)**  
notification\_id (PK), user\_id (FK), product\_id (FK, NULL), channel [EMAIL|PUSH], template [TTL\_T3|TTL\_T1|TTL\_0|INCIDENT|OTHER], sent\_at, status [SENT|FAILED], error\_msg (NULL)



## Даталогическая модель

```

-- =====
-- ===== Роли и пользователи =====
-- =====

CREATE TABLE roles (
    role_id BIGSERIAL PRIMARY KEY,
    code TEXT UNIQUE NOT NULL CHECK (code IN ('ADMIN', 'MODERATOR',
'USER')),
    name TEXT NOT NULL
);

CREATE TABLE users (
    user_id BIGSERIAL PRIMARY KEY,
    email TEXT UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    name TEXT NOT NULL,
    room TEXT,
    notif_email_on BOOLEAN DEFAULT TRUE,
    notif_push_on BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT NOW(),
    blocked_at TIMESTAMP NULL
);
  
```

```
CREATE TABLE user_roles (
    user_id BIGINT NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    role_id BIGINT NOT NULL REFERENCES roles(role_id) ON DELETE CASCADE,
    PRIMARY KEY (user_id, role_id)
);

-- =====
-- ===== Холодильники и участники =====
-- =====

CREATE TABLE fridges (
    fridge_id BIGSERIAL PRIMARY KEY,
    name TEXT NOT NULL,
    location TEXT,
    created_at TIMESTAMP DEFAULT NOW(),
    invite_required BOOLEAN DEFAULT FALSE
);

CREATE TABLE fridge_memberships (
    fridge_id BIGINT NOT NULL REFERENCES fridges(fridge_id) ON DELETE CASCADE,
    user_id BIGINT NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    is_moderator BOOLEAN DEFAULT FALSE,
    joined_at TIMESTAMP DEFAULT NOW(),
    left_at TIMESTAMP NULL,
    PRIMARY KEY (fridge_id, user_id)
);

CREATE TABLE zones (
    zone_id BIGSERIAL PRIMARY KEY,
    fridge_id BIGINT NOT NULL REFERENCES fridges(fridge_id) ON DELETE CASCADE,
    name TEXT NOT NULL,
    capacity_units INT NOT NULL,
    capacity_volume_l NUMERIC(8,2),
    is_active BOOLEAN DEFAULT TRUE,
    sort_order INT DEFAULT 0
);

-- =====
-- ===== Категории и продукты =====
-- =====

CREATE TABLE categories (
    category_id BIGSERIAL PRIMARY KEY,
```

```
    name TEXT UNIQUE NOT NULL,
    perishable_days_default INT
);

CREATE TABLE products (
    product_id BIGSERIAL PRIMARY KEY,
    owner_id BIGINT NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    zone_id BIGINT NOT NULL REFERENCES zones(zone_id) ON DELETE CASCADE,
    category_id BIGINT REFERENCES categories(category_id) ON DELETE SET
NULL,
    name TEXT NOT NULL,
    barcode TEXT,
    expires_at TIMESTAMP,
    placed_at TIMESTAMP DEFAULT NOW(),
    locked BOOLEAN DEFAULT FALSE,
    status TEXT NOT NULL CHECK (status IN
('ACTIVE', 'EATEN', 'TAKEN', 'DISPOSED', 'EXPIRED'))
);

-- =====
-- ===== История изменений продуктов =====
-- =====

CREATE TABLE product_history (
    history_id BIGSERIAL PRIMARY KEY,
    product_id BIGINT NOT NULL REFERENCES products(product_id) ON DELETE
CASCADE,
    event_type TEXT NOT NULL CHECK (event_type IN
('MOVE', 'EXTEND', 'STATUS', 'EDIT')),
    from_zone_id BIGINT REFERENCES zones(zone_id) ON DELETE SET NULL,
    to_zone_id BIGINT REFERENCES zones(zone_id) ON DELETE SET NULL,
    old_expires_at TIMESTAMP,
    new_expires_at TIMESTAMP,
    comment TEXT,
    actor_id BIGINT REFERENCES users(user_id) ON DELETE SET NULL,
    created_at TIMESTAMP DEFAULT NOW()
);

-- =====
-- ===== Уведомления =====
```

```
-- =====

CREATE TABLE notifications (
    notification_id BIGSERIAL PRIMARY KEY,
    user_id BIGINT NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    product_id BIGINT REFERENCES products(product_id) ON DELETE SET NULL,
    channel TEXT NOT NULL CHECK (channel IN ('EMAIL','PUSH')),
    template TEXT NOT NULL CHECK (template IN
    ('TTL_T3','TTL_T1','TTL_0','INCIDENT','OTHER')),
    sent_at TIMESTAMP DEFAULT NOW(),
    status TEXT NOT NULL CHECK (status IN ('SENT','FAILED')),
    error_msg TEXT
);
```

## Скрипты

```
-- === Типы ENUM ===

CREATE TYPE status_enum AS ENUM
('ACTIVE','EATEN','TAKEN','DISPOSED','EXPIRED');
CREATE TYPE event_type_enum AS ENUM ('MOVE','EXTEND','STATUS','EDIT');
CREATE TYPE channel_enum AS ENUM ('EMAIL','PUSH');
CREATE TYPE template_enum AS ENUM
('TTL_T3','TTL_T1','TTL_0','INCIDENT','OTHER');
CREATE TYPE notif_status_enum AS ENUM ('SENT','FAILED');

-- === Таблицы ===

-- Роли и пользователи

CREATE TABLE roles (
    role_id BIGSERIAL PRIMARY KEY,
    code TEXT UNIQUE NOT NULL CHECK (code IN ('ADMIN','MODERATOR','USER')),
    name TEXT NOT NULL
);

CREATE TABLE users (
    user_id BIGSERIAL PRIMARY KEY,
    email TEXT UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    name TEXT NOT NULL,
    room TEXT,
    notif_email_on BOOLEAN NOT NULL DEFAULT TRUE,
    notif_push_on BOOLEAN NOT NULL DEFAULT TRUE,
    created_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT now(),
    blocked_at TIMESTAMP WITH TIME ZONE
);
```

```

CREATE TABLE user_roles (
    user_id BIGINT NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    role_id BIGINT NOT NULL REFERENCES roles(role_id) ON DELETE CASCADE,
    PRIMARY KEY (user_id, role_id)
);

-- Холодильники и участники
CREATE TABLE fridges (
    fridge_id BIGSERIAL PRIMARY KEY,
    name TEXT NOT NULL,
    location TEXT,
    created_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT now(),
    invite_required BOOLEAN NOT NULL DEFAULT FALSE
);

CREATE TABLE fridge_memberships (
    fridge_id BIGINT NOT NULL REFERENCES fridges(fridge_id) ON DELETE CASCADE,
    user_id BIGINT NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    is_moderator BOOLEAN NOT NULL DEFAULT FALSE,
    joined_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT now(),
    left_at TIMESTAMP WITH TIME ZONE,
    PRIMARY KEY (fridge_id, user_id),
    -- базовая проверка: joined_at ≤ left_at когда left_at задан
    CONSTRAINT chk_fridge_membership_dates CHECK (left_at IS NULL OR left_at
    ≥ joined_at)
);

-- Зоны
CREATE TABLE zones (
    zone_id BIGSERIAL PRIMARY KEY,
    fridge_id BIGINT NOT NULL REFERENCES fridges(fridge_id) ON DELETE CASCADE,
    name TEXT NOT NULL,
    capacity_units INT NOT NULL CHECK (capacity_units ≥ 0),
    capacity_volume_l NUMERIC(10,2),
    is_active BOOLEAN NOT NULL DEFAULT TRUE,
    sort_order INT NOT NULL DEFAULT 0
);

-- Категории
CREATE TABLE categories (
    category_id BIGSERIAL PRIMARY KEY,
    name TEXT UNIQUE NOT NULL,
    perishable_days_default INT CHECK (perishable_days_default IS NULL OR
    perishable_days_default ≥ 0)
);

```

```
);

-- Продукты
CREATE TABLE products (
    product_id BIGSERIAL PRIMARY KEY,
    owner_id BIGINT NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    zone_id BIGINT NOT NULL REFERENCES zones(zone_id) ON DELETE CASCADE,
    category_id BIGINT REFERENCES categories(category_id) ON DELETE SET
NULL,
    name TEXT NOT NULL,
    barcode TEXT,
    expires_at TIMESTAMP WITH TIME ZONE,
    placed_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT now(),
    locked BOOLEAN NOT NULL DEFAULT FALSE,
    status status_enum NOT NULL DEFAULT 'ACTIVE'
);
;

-- История
CREATE TABLE product_history (
    history_id BIGSERIAL PRIMARY KEY,
    product_id BIGINT NOT NULL REFERENCES products(product_id) ON DELETE
CASCADE,
    event_type event_type_enum NOT NULL,
    from_zone_id BIGINT REFERENCES zones(zone_id) ON DELETE SET NULL,
    to_zone_id BIGINT REFERENCES zones(zone_id) ON DELETE SET NULL,
    old_expires_at TIMESTAMP WITH TIME ZONE,
    new_expires_at TIMESTAMP WITH TIME ZONE,
    comment TEXT,
    actor_id BIGINT REFERENCES users(user_id) ON DELETE SET NULL,
    created_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT now()
);
;

-- Уведомления
CREATE TABLE notifications (
    notification_id BIGSERIAL PRIMARY KEY,
    user_id BIGINT NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    product_id BIGINT REFERENCES products(product_id) ON DELETE SET NULL,
    channel channel_enum NOT NULL,
    template template_enum NOT NULL,
    sent_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT now(),
    status notif_status_enum NOT NULL,
    error_msg TEXT
);
;

-- Индексы полезные
CREATE INDEX idx_products_zone ON products(zone_id);
```

```

CREATE INDEX idx_products_owner ON products(owner_id);
CREATE INDEX idx_zones_fridge ON zones(fridge_id);
CREATE INDEX idx_fridge_memberships_user ON fridge_memberships(user_id);

-- === Функции и триггеры для целостности логики ===

CREATE OR REPLACE FUNCTION products_before_insert_update()
RETURNS TRIGGER AS $$
DECLARE
    cat_days INT;
    zone_fridge_id BIGINT;
    membership_exists BOOLEAN;
BEGIN
    IF NEW.placed_at IS NULL THEN
        NEW.placed_at := now();
    END IF;

    IF NEW.expires_at IS NULL AND NEW.category_id IS NOT NULL THEN
        SELECT perishable_days_default INTO cat_days FROM categories WHERE
category_id = NEW.category_id;
        IF cat_days IS NOT NULL THEN
            NEW.expires_at := NEW.placed_at + (cat_days || ' days')::interval;
        END IF;
    END IF;

    IF NEW.expires_at IS NOT NULL AND NEW.expires_at <= now() THEN
        NEW.status := 'EXPIRED';
    ELSE
        IF NEW.status IS NULL THEN
            NEW.status := 'ACTIVE';
        END IF;
    END IF;

    SELECT fridge_id INTO zone_fridge_id FROM zones WHERE zone_id =
NEW.zone_id;
    IF zone_fridge_id IS NULL THEN
        RAISE EXCEPTION 'Zone % does not exist', NEW.zone_id;
    END IF;

    SELECT EXISTS (
        SELECT 1 FROM fridge_memberships
        WHERE fridge_id = zone_fridge_id
        AND user_id = NEW.owner_id
        AND (left_at IS NULL OR left_at >= NEW.placed_at)
    ) INTO membership_exists;

```

```

IF NOT membership_exists THEN
    RAISE EXCEPTION 'Owner (user_id=%) is not an active member of the
fridge owning zone %', NEW.owner_id, NEW.zone_id;
END IF;

RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_products_before_ins_upd
BEFORE INSERT OR UPDATE ON products
FOR EACH ROW EXECUTE FUNCTION products_before_insert_update();

/*
Функция: product_history_after_insert()
- если событие MOVE: обновляет products.zone_id = to_zone_id
- если событие EXTEND: обновляет products.expires_at = new_expires_at и
может обновить status
- делает простую синхронизацию истории в основном объекте
*/
CREATE OR REPLACE FUNCTION product_history_after_insert()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.event_type = 'MOVE' THEN
        IF NEW.to_zone_id IS NOT NULL THEN
            UPDATE products SET zone_id = NEW.to_zone_id WHERE product_id =
NEW.product_id;
        END IF;
    ELSIF NEW.event_type = 'EXTEND' THEN
        IF NEW.new_expires_at IS NOT NULL THEN
            UPDATE products
            SET expires_at = NEW.new_expires_at,
                status = CASE WHEN NEW.new_expires_at ≤ now() THEN
'EXPIRED' ELSE status END
                WHERE product_id = NEW.product_id;
        END IF;
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_product_history_after_ins
AFTER INSERT ON product_history

```

```

FOR EACH ROW EXECUTE FUNCTION product_history_after_insert();

-- === Пример вспомогательных представлений (optional) ===
CREATE VIEW vw_products_with_category AS
SELECT p.*, c.name AS category_name, z.fridge_id
FROM products p
LEFT JOIN categories c ON p.category_id = c.category_id
LEFT JOIN zones z ON p.zone_id = z.zone_id;

```

## Скрипт тестовых данных

```

-- Очистка
TRUNCATE notifications, product_history, products, categories, zones,
fridge_memberships, fridges, user_roles, users, roles RESTART IDENTITY
CASCADE;

-- Роли
INSERT INTO roles (code, name) VALUES
('ADMIN','Administrator'),
('MODERATOR','Moderator'),
('USER','User');

-- Пользователи
INSERT INTO users
(email,password_hash,name,room,notif_email_on,notif_push_on)
VALUES
('alice@example.com','hash_alice','Alice','101', TRUE, TRUE),
('bob@example.com','hash_bob','Bob','102', TRUE, FALSE),
('eve@example.com','hash_eve','Eve','103', FALSE, TRUE);

-- user_roles
INSERT INTO user_roles (user_id, role_id)
VALUES
(1, 1), -- Alice → ADMIN
(2, 3), -- Bob → USER
(3, 3); -- Eve → USER

-- Fridge
INSERT INTO fridges (name, location, invite_required)
VALUES ('Kitchen Fridge', 'Apartment 1, Kitchen', FALSE);

-- Fridge memberships
INSERT INTO fridge_memberships (fridge_id, user_id, is_moderator,
joined_at)
VALUES
(1, 1, TRUE, now() - interval '30 days'), -- Alice moderator

```

```
(1, 2, FALSE, now() - interval '10 days'); -- Bob member

-- Zones
INSERT INTO zones (fridge_id, name, capacity_units, capacity_volume_l,
is_active, sort_order)
VALUES
(1, 'Top Shelf', 20, 50.0, TRUE, 1),
(1, 'Bottom Drawer', 10, 30.0, TRUE, 2);

-- Categories
INSERT INTO categories (name, perishable_days_default)
VALUES
('Dairy', 7),
('Veggies', 10),
('Canned', NULL);

-- Продукты
INSERT INTO products (owner_id, zone_id, category_id, name, barcode)
VALUES
(1, 1, 1, 'Strawberry Yogurt 150g', '0123456789012');

-- 2) морковь (Veggies) с явным сроком
INSERT INTO products (owner_id, zone_id, category_id, name, expires_at)
VALUES
(2, 2, 2, 'Carrots (1 kg)', now() + interval '8 days');

-- 3) консервированная фасоль (Canned) без срока (non-perishable)
INSERT INTO products (owner_id, zone_id, category_id, name)
VALUES
(2, 1, 3, 'Canned Beans 400g');

-- История: MOVE продукта 2 в зону 1
INSERT INTO product_history (product_id, event_type, from_zone_id,
to_zone_id, actor_id, comment)
VALUES
(2, 'MOVE', 2, 1, 2, 'Moved to top shelf');

-- EXTEND: продление срока продукта 1
INSERT INTO product_history (product_id, event_type, old_expires_at,
new_expires_at, actor_id, comment)
VALUES
(1, 'EXTEND', (SELECT expires_at FROM products WHERE product_id = 1),
(SELECT expires_at FROM products WHERE product_id = 1) + interval '3 days',
1, 'Extended by owner');

-- Уведомление (пример)
```

```

INSERT INTO notifications (user_id, product_id, channel, template, status)
VALUES
(1, 1, 'EMAIL', 'TTL_T1', 'SENT'),
(2, 2, 'PUSH', 'TTL_T3', 'SENT');

-- Проверка выборок
-- SELECT * FROM users;
-- SELECT * FROM products;
-- SELECT * FROM product_history;
-- SELECT * FROM notifications;

```

## ФУНКЦИИ

```

-- =====
-- 1. Добавление участника в холодильник
-- =====

CREATE OR REPLACE FUNCTION fn_add_fridge_member(p_fridge_id BIGINT,
p_user_id BIGINT, p_is_moderator BOOLEAN DEFAULT FALSE)
RETURNS VOID
LANGUAGE plpgsql AS $$

BEGIN
    -- проверка наличия холодильника и пользователя
    PERFORM 1 FROM fridges WHERE fridge_id = p_fridge_id;
    IF NOT FOUND THEN
        RAISE EXCEPTION 'Fridge % not found', p_fridge_id;
    END IF;

    PERFORM 1 FROM users WHERE user_id = p_user_id;
    IF NOT FOUND THEN
        RAISE EXCEPTION 'User % not found', p_user_id;
    END IF;

    INSERT INTO fridge_memberships(fridge_id, user_id, is_moderator,
joined_at, left_at)
    VALUES (p_fridge_id, p_user_id, p_is_moderator, now(), NULL)
    ON CONFLICT (fridge_id, user_id) DO UPDATE
        SET is_moderator = EXCLUDED.is_moderator,
            joined_at = COALESCE(fridge_memberships.joined_at,
EXCLUDED.joined_at),
            left_at = NULL; -- re-activate if was left
END;
$$;

-- =====
-- 2. Удаление (отметка left_at) участника холодильника

```

```

-- =====
CREATE OR REPLACE FUNCTION fn_remove_fridge_member(p_fridge_id BIGINT,
p_user_id BIGINT)
RETURNS VOID
LANGUAGE plpgsql AS $$

BEGIN
    UPDATE fridge_memberships
    SET left_at = now()
    WHERE fridge_id = p_fridge_id AND user_id = p_user_id AND left_at IS NULL;

    IF NOT FOUND THEN
        RAISE NOTICE 'No active membership found for fridge=% user=%',
p_fridge_id, p_user_id;
    END IF;
END;
$$;

-- =====
-- 3. Добавить продукт (возвращает product_id)
-- =====

CREATE OR REPLACE FUNCTION fn_add_product(
    p_owner_id BIGINT,
    p_zone_id BIGINT,
    p_category_id BIGINT DEFAULT NULL,
    p_name TEXT,
    p_barcode TEXT DEFAULT NULL,
    p_expires_at TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    p_placed_at TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    p_locked BOOLEAN DEFAULT FALSE,
    p_status status_enum DEFAULT 'ACTIVE'
)
RETURNS BIGINT
LANGUAGE plpgsql AS $$

DECLARE
    v_product_id BIGINT;
BEGIN
    INSERT INTO products(owner_id, zone_id, category_id, name, barcode,
expires_at, placed_at, locked, status)
    VALUES (p_owner_id, p_zone_id, p_category_id, p_name, p_barcode,
p_expires_at, p_placed_at, p_locked, p_status)
    RETURNING product_id INTO v_product_id;

    RETURN v_product_id;
END;
$$;

```

```
-- =====
-- 4. Получить продукты, у которых expires_at в пределах N дней (для
уведомлений)
-- =====
CREATE OR REPLACE FUNCTION fn_get_products_expiring_within(p_fridge_id
BIGINT, p_days INT)
RETURNS TABLE (
product_id BIGINT,
name TEXT,
expires_at TIMESTAMP WITH TIME ZONE,
owner_id BIGINT,
zone_id BIGINT,
fridge_id BIGINT
)
LANGUAGE plpgsql AS $$

BEGIN
RETURN QUERY
SELECT p.product_id, p.name, p.expires_at, p.owner_id, p.zone_id,
z.fridge_id
FROM products p
JOIN zones z ON z.zone_id = p.zone_id
WHERE z.fridge_id = p_fridge_id
AND p.expires_at IS NOT NULL
AND p.expires_at <= now() + (p_days || ' days')::interval
AND p.status = 'ACTIVE';
END;
$$;

-- =====
-- 5. Получить все продукты пользователя
-- =====
CREATE OR REPLACE FUNCTION fn_get_user_products(p_user_id BIGINT)
RETURNS TABLE (
product_id BIGINT,
name TEXT,
expires_at TIMESTAMP WITH TIME ZONE,
status status_enum,
zone_id BIGINT,
fridge_id BIGINT
)
LANGUAGE plpgsql AS $$

BEGIN
RETURN QUERY
SELECT p.product_id, p.name, p.expires_at, p.status, p.zone_id,
z.fridge_id
FROM products p
```

```

JOIN zones z ON z.zone_id = p.zone_id
WHERE p.owner_id = p_user_id
ORDER BY p.expires_at NULLS LAST, p.name;
END;
$$;

-- =====
-- 6. Создать уведомление
-- =====

CREATE OR REPLACE FUNCTION fn_create_notification(
    p_user_id BIGINT,
    p_product_id BIGINT DEFAULT NULL,
    p_channel channel_enum,
    p_template template_enum,
    p_status notif_status_enum DEFAULT 'SENT',
    p_error_msg TEXT DEFAULT NULL
)
RETURNS BIGINT
LANGUAGE plpgsql AS $$

DECLARE
    v_id BIGINT;
BEGIN
    INSERT INTO notifications(user_id, product_id, channel, template, sent_at,
status, error_msg)
    VALUES (p_user_id, p_product_id, p_channel, p_template, now(), p_status,
p_error_msg)
    RETURNING notification_id INTO v_id;
    RETURN v_id;
END;
$$;

```

## Индексы

```

-- 1) Быстрый поиск по срокам годности – используется для генерации
уведомлений
CREATE INDEX IF NOT EXISTS idx_products_expires_at ON products
(expires_at);

-- 2) Частые фильтры по статусу (напр., показ активных/истёкших продуктов)
CREATE INDEX IF NOT EXISTS idx_products_status ON products (status);

-- 3) Частые запросы "все продукты конкретного владельца с определённым
статусом"
CREATE INDEX IF NOT EXISTS idx_products_owner_status ON products (owner_id,
status);

```

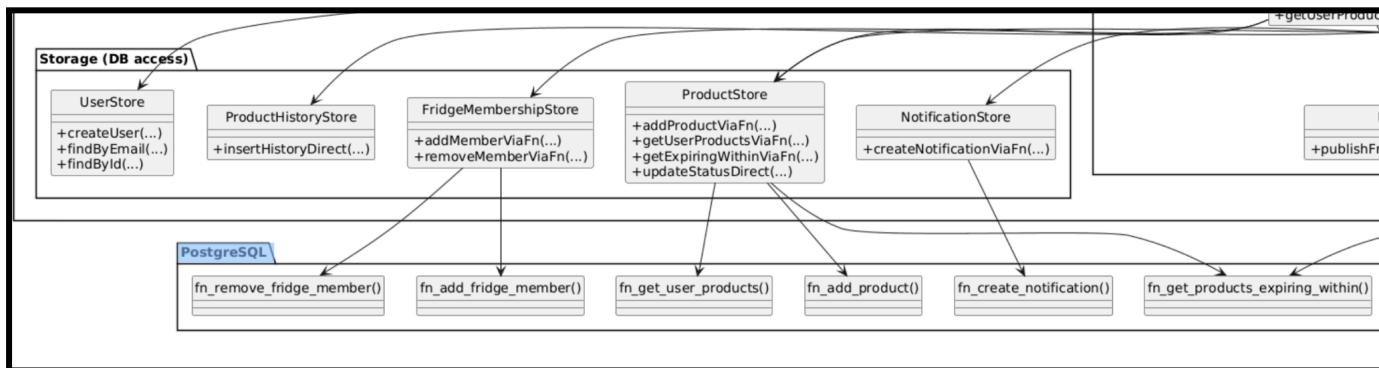
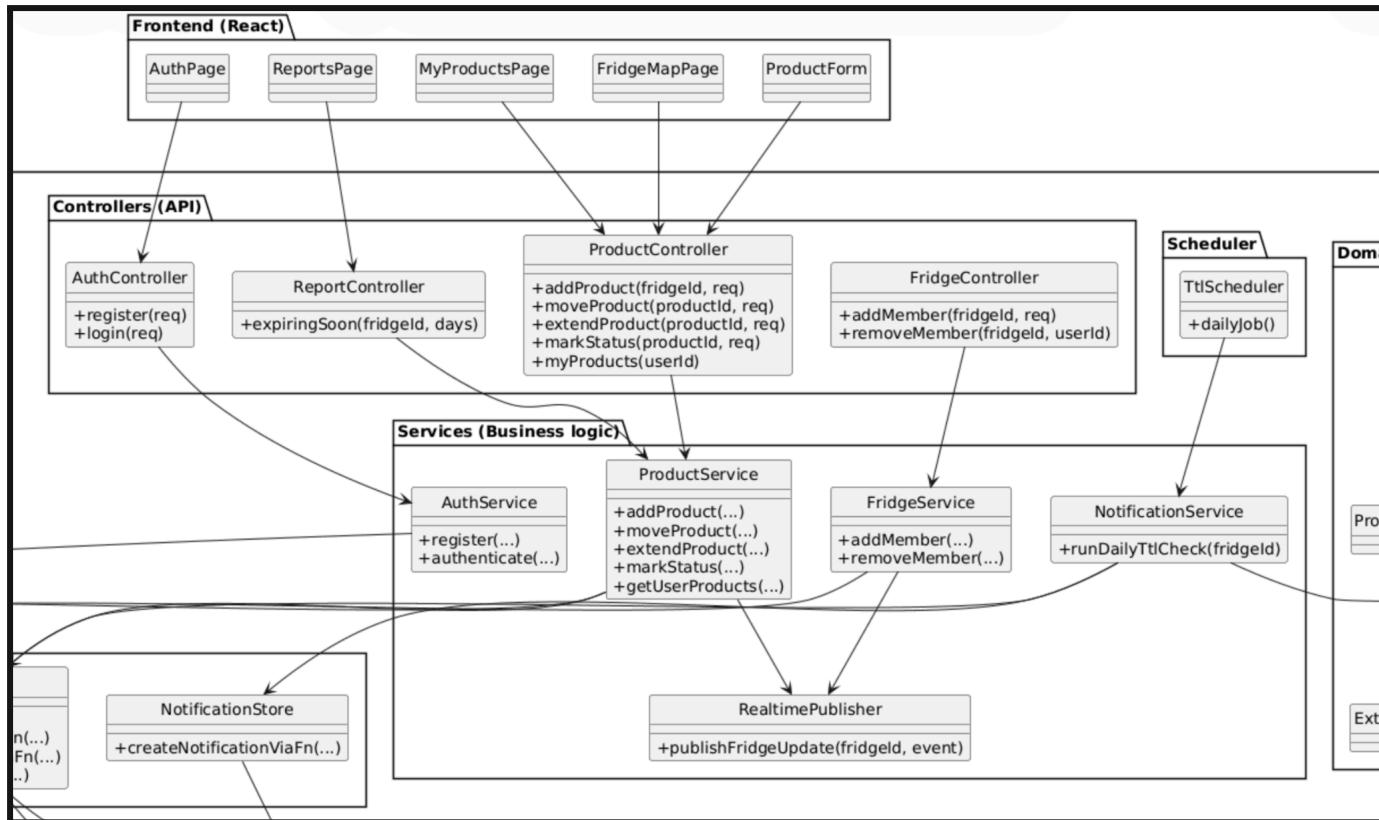
```
-- 4) Поиск продуктов внутри зоны по статусу (например: показать все
просроченные в зоне)
CREATE INDEX IF NOT EXISTS idx_products_zone_status ON products (zone_id,
status);

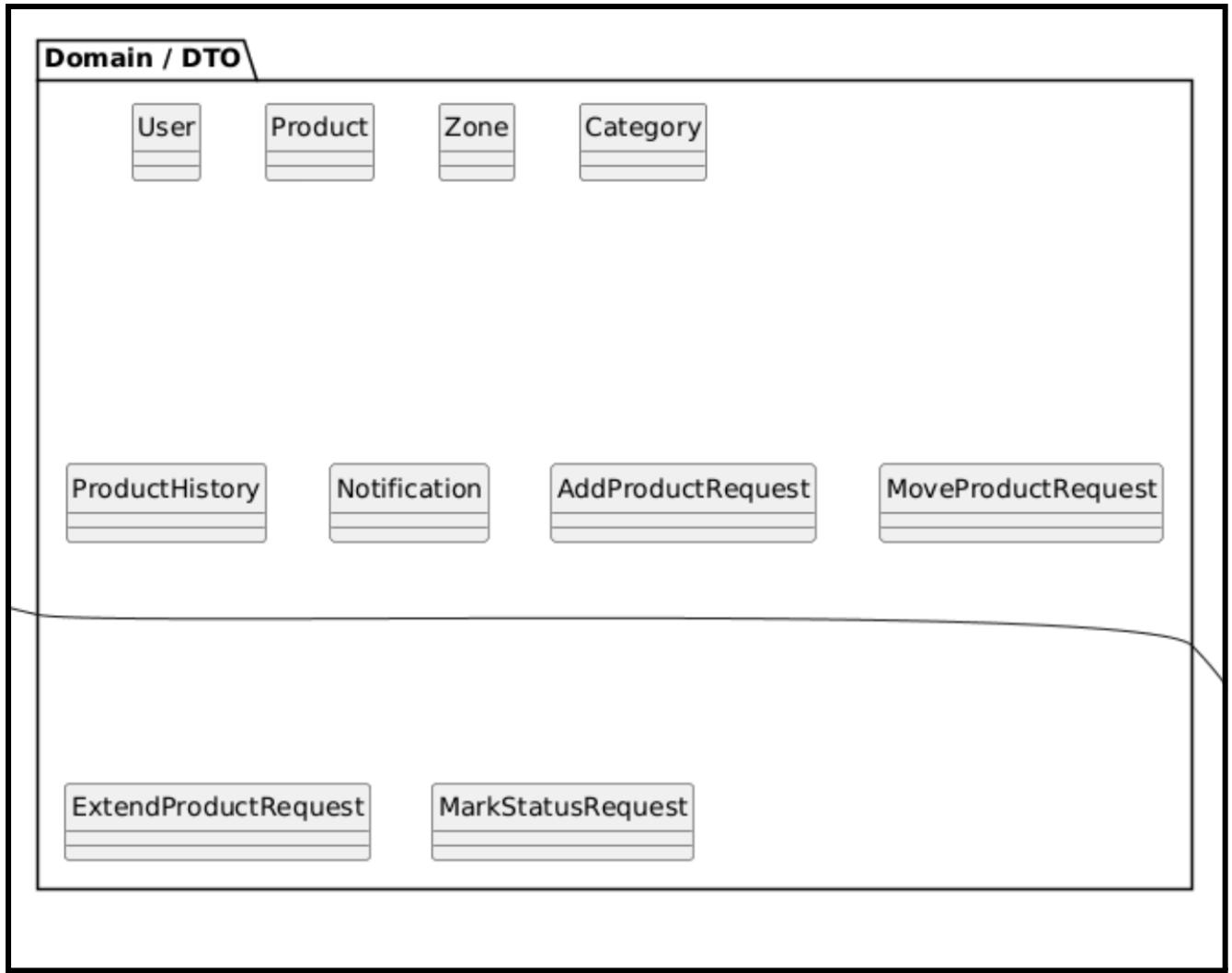
-- 5) Индекс по уведомлениям: выборка по пользователю и времени (показать
историю уведомлений)
CREATE INDEX IF NOT EXISTS idx_notifications_user_sentat ON notifications
(user_id, sent_at DESC);

-- 6) Индексы для истории: быстрый доступ к историям продукта по времени
CREATE INDEX IF NOT EXISTS idx_product_history_product_created_at ON
product_history (product_id, created_at DESC);
```

## Этап 3

Диаграмма классов, представляющая общую архитектуру системы





## 1. Общая архитектура (класс-диаграмма — текстовое описание)

- Пакет **model** (сущности, отображаемые в таблицы базы):
  - **User** (`userId`, `email`, `passwordHash`, `name`, `room`, `notifEmailOn`, `notifPushOn`, `createdAt`, `blockedAt`, `roles`)
  - **Role** (`roleId`, `code`, `name`)
  - **Fridge** (`fridgeld`, `name`, `location`, `ownerId`, `inviteRequired`)
  - **Zone** (`zoneld`, `fridgeld`, `name`, `capacityUnits`, `isActive`, `sortOrder`)
  - **Product** (`productId`, `ownerId`, `zoneld`, `categoryId`, `name`, `barcode`, `expiresAt`, `placedAt`, `status`, `locked`)
  - **ProductHistory** (`historyId`, `productId`, `eventType`, `fromZoneld`, `toZoneld`, `oldExpiresAt`, `newExpiresAt`, `actorId`, `comment`, `createdAt`)
  - **Notification** (`notificationId`, `userId`, `productId`, `channel`, `template`, `sentAt`, `status`, `errorMsg`)
  - **AuditLog** (`auditId`, `actorEmail`, `action`, `entityType`, `entityId`, `details`, `createdAt`)

- Пакет **repository**:
  - Jpa репозитории для CRUD: **UserRepository**, **RoleRepository**, **FridgeRepository**, **ZoneRepository**, **ProductRepository**, **ProductHistoryRepository**, **NotificationRepository**, **AuditLogRepository**.
  - Дополнительно — низкоуровневые репозитории или **Gateway**, вызывающие PL/pgSQL функции (**JdbcTemplate** или **@Query native**). Например: **DbFunctionRepository** / **DatabaseFunctionGateway**.
- Пакет **service** (слой бизнес-логики):
  - **UserService** (регистрация, профиль, управление ролями)
  - **FridgeService** (создание/редактирование холодильников, приглашения, члены)
  - **ProductService** (добавление продуктов, перемещение, продление срока, изменение статуса)
  - **NotificationService** (поиск продуктов с близким сроком, создание уведомлений, отправка)
  - **EmailService** (формирование и отправка писем, шаблоны Thymeleaf; **@Async** для отправки)
  - **AuditService** (логирование действий в таблицу `audit_logs`)
  - **AdminService** / **ModeratorService** (функции админов/модераторов)
- Пакет **controller**:
  - **AuthController**, **DashboardController**, **FridgeWebController**, **ProductWebController**, **AdminController**, **ModeratorController**, **DevController**
- Вспомогательные: **dto** (**RegisterForm**, **ExpiryProduct**), **security** (**CustomUserDetailsService**, **SecurityConfig**), **config** (**MailConfig**, **StartupDataInitializer**).

Диаграмма классов должна отражать зависимости:

- Controllers → Services → Repositories/DbFunctionGateway → DB
- Services используют модели (entities) и DTOs; EmailService использует Thymeleaf TemplateEngine + JavaMailSender.

## 2. Уровень хранения (реализовано в DB, PL/pgSQL)

В **schema.sql** реализованы таблицы, триггеры и PL/pgSQL-функции.

Ключевые функции, которые обязательно используются бизнес-логикой:

- **fn\_add\_fridge\_member(p\_fridge\_id, p\_user\_id, p\_is\_moderator)** — добавить/обновить членство в холодильнике.

- `fn_remove_fridge_member(p_fridge_id, p_user_id)` — пометить уход участника.
- `fn_add_product(p_owner_id, p_zone_id, p_category_id, p_name, p_barcode, p_expires_at, ...)` RETURNS BIGINT — создание продукта, возвращает id.
- `fn_get_products_expiring_within(p_fridge_id, p_days)` RETURNS TABLE(...) — возвращает продукты, сроки которых истекут в p\_days.
- `fn_get_user_products(p_user_id)` — список продуктов пользователя.
- `fn_create_notification(p_user_id, p_product_id, p_channel, p_template, p_status, p_error_msg)` — создание записи уведомления.
- `fn_move_product(p_product_id, p_to_zone_id, p_actor_id, p_comment)` — перемещение продукта (помещает запись в product\_history, а триггер обновляет реальную зону).
- `fn_extend_product(p_product_id, p_new_expires_at, p_actor_id, p_comment)` — продление срока (через history триггер обновит expires\_at).
- `fn_set_product_status(p_product_id, p_status, p_actor_id, p_comment)` — смена статуса (и запись в history).
- `fn_get_fridge_map(p_fridge_id)` — карта холодильника с подсчётом активных и просроченных элементов.

Также есть триггеры:

- `products_before_insert_update()` — автоматическая установка expires\_at по категории, проверка членства и т.д.
- `product_history_after_insert()` — обновление продуктов в ответ на вставку события в историю.

**Ключевая архитектурная идея: операции записи/изменения состояния на уровне хранения делаются через вызов этих функций — бизнес-логика не должна вручную изменять таблицы вместо функций. Это обеспечивает целостность, бизнес-правила в одном месте и возможность переиспользования.**

---

### 3. Как бизнес-логика использует функции БД (практическая реализация)

В Java-слое вызов PL/pgSQL-функций выполнен через одну из следующих техник (в проекте — через `JdbcTemplate / NamedParameterJdbcTemplate` или `@Query(nativeQuery=true)`):

Примеры вызовов:

- Добавление продукта (ProductService):

- Контракт: `addProduct(ownerId, zoneId, categoryId, name, barcode, expiresAt) → возвращает productId.`
  - Реализация (псевдокод):
    - `Long id = jdbcTemplate.queryForObject("SELECT fn_add_product(?,?,?,?,?, ?, ?, ?)", Long.class, ownerId, zoneId, categoryId, name, barcode, expiresAt, placedAt, locked);`
    - Возвращаем id и затем читаем сущность через `ProductRepository.findById(id)` для отображения.
- Получение просроков (NotificationService):
  - `List<ExpiryProduct> rows = jdbcTemplate.query("SELECT * FROM fn_get_products_expiring_within(?, ?)", rowMapper, fridgeld, days);`
  - Далее группировка по ownerId и подготовка письма.
- Создание уведомления:
  - `Long notifId = jdbcTemplate.queryForObject("SELECT fn_create_notification(?,?,?,?,?,?)", Long.class, userId, productId, channel, template, status, errorMsg);`
- Добавление/удаление участника холодильника:
  - `jdbcTemplate.update("SELECT fn_add_fridge_member(?, ?, ?)", fridgeld, userId, isModerator);`
- Перемещение/продление статуса продукта:
  - `jdbcTemplate.update("SELECT fn_move_product(?, ?, ?)", productId, toZoneId, actorId);`

Все изменения состояния продукта (перемещение, изменение даты) делаются через функции и триггеры БД, а не прямыми UPDATE/INSERT в таблицы products/product\_history.

---

#### **4. Реализованные бизнес-процессы (подробно)**

**Ниже — основные процессы с шагами, контрактами и использованными DB-функциями.**

##### **4.1 Регистрация пользователя**

- Входные данные: `RegisterForm (email, name, password, room, notifEmailOn, notifPushOn).`
- Контракт `UserService.register(RegisterForm) → User`
- Шаги:
  - Проверка уникальности email (`UserRepository.findByEmail`).
  - Создание сущности User и сохранение через `UserRepository.save (INSERT в users).`

- (Можно использовать DB-функцию для более сложных сценариев, но базовый `insert` через JPA допустим для создания пользователя — в задании главное, чтобы уровень хранения для специфических доменных операций использовал pl/pgsql).
- Ассоциация роли USER (`RoleRepository.findByCode("USER")` и вставка в `user_roles`).
- Вызов `EmailService.sendRegistrationEmail(User)` — асинхронно отправляет приветственное письмо.
- Отказоустойчивость:
  - Email отправляется best-effort: ошибка отправки логируется, но регистрация успешна.

**Диаграмма последовательности:** User -> AuthController.postRegister -> UserService.register -> UserRepository.save -> (commit) -> EmailService.sendRegistrationEmail (async) -> JavaMailSender.send -> SMTP server.

#### 4.2 Добавление продукта

- Контракт `ProductService.addProduct(ownerId, zoneId, categoryId, name, barcode, expiresAt) → productId`
- Шаги:
  1. Бизнес-валидации (наличие зоны, членство пользователя в холодильнике) — хотя триггер в DB уже проверяет членство, сервис может предварительно валидировать и выдать более удобную ошибку.
  2. Вызов `fn_add_product(...)` через `JdbcTemplate` — Функция вернёт `product_id`.
  3. Запись аудита: `AuditService.log(actorEmail, "ADD_PRODUCT", "product", productId, details)`.
- Причина использования DB-функции: правила в триггерах и логика создания продукта остаются на уровне хранения.

#### 4.3 Перемещение продукта (MOVE) / Продление срока (EXTEND) / Смена статуса

- Контракт `ProductService.moveProduct(productId, toZoneId, actorId, comment)`
- Шаги:
  1. Service вызывает `fn_move_product(productId, toZoneId, actorId, comment)`.
  2. `fn_move_product` вставляет запись в `product_history`; триггер `product_history_after_insert` обновляет таблицу `products` — это гарантирует единый центр бизнес-правил.
- Аналогично для `fn_extend_product` и `fn_set_product_status`.

#### 4.4 Уведомления о скором истечении срока (ежедневная/периодическая задача)

- Описание:
  - Периодически (например, cron каждые 6 часов) NotificationService запускает batch:
    - Для каждого холодильника выполняет fn\_get\_products\_expiring\_within(fridgeId, days).
    - Группирует результаты по owner\_id.
    - Для каждого owner формирует HTML шаблон (Thymeleaf) и вызывает EmailService.sendExpiryReminder(user, products, days).
    - После отправки вызывает fn\_create\_notification для каждой записи (статус SENT или FAILED).
- В реализации NotificationService используется:
  - JdbcTemplate для получения строк из fn\_get\_products\_expiring\_within.
  - EmailService (async) для отправки писем.
  - fn\_create\_notification для записи факта уведомления.
- Контроль повторов:
  - Если уведомление уже создано ранее (логика может использовать существующие уведомления), сервис может проверять и избегать дублирования.

#### 4.5 Управление холодильником и участниками

- Добавление/удаление члена холодильника — через fn\_add\_fridge\_member / fn\_remove\_fridge\_member (обязательно).
- При создании холодильника владелец автоматически добавляется как участник (через вызов fn\_add\_fridge\_member в транзакции создания).

#### 4.6 Администрирование и аудит

- Админ может просматривать список пользователей, менять роли — операции через сервис AdminService, при изменениях создаются записи в audit\_logs (через AuditService), чтобы зафиксировать actor\_email, действие, entity\_type и детали.
- AuditService записывает логи через AuditLogRepository, или в критических сценариях может вызывать хранимую функцию

---

### 5. Примеры контрактов (вход/выход, ошибки)

- ProductService.addProduct

- Вход: AddProductDTO {ownerId, zoneId, categoryId, name, barcode, expiresAt}
    - Выход: productId (Long) или BusinessException
    - Ошибки: ZoneNotFoundException, UserNotMemberException, ConstraintViolationException (проброс из DB)
  - NotificationService.runExpiryBatch
    - Вход: fridgeld, daysBefore
    - Выход: createdNotificationsCount (int)
    - Ошибки: логируются и возвращается частичный результат при ошибках отправки
  - UserService.register
    - Вход: RegisterForm
    - Выход: User
    - Ошибки: EmailExistsException, ValidationException
- 

## 6. Транзакции и атомарность

- По архитектуре:
    - Операции, которые затрагивают несколько таблиц (например, создание продукта + добавление в history), выполняются через одну DB-функцию либо в пределах одной транзакции Spring (@Transactional), чтобы обеспечить атомарность.
    - Поскольку многие функции создают запись в product\_history, а триггеры затем модифицируют products, важно вызывать их внутри транзакции сочные ожидания завершения — Spring автоматически управляет транзакциями для методов помеченных @Transactional.
  - Email отправка выполняется асинхронно и не должна блокировать основной транзакционный поток; поэтому регистрация НЕ откладывается на успешную отправку письма (best-effort).
- 

## 7. Обработка ошибок и логирование

- Локальная валидация в сервисах: проверка наличия сущностей, прав доступа (только участник холодильника может добавлять продукт).
- Ошибки БД (raise exception в функциях) пробрасываются как DataAccessException; сервисы ловят их, преобразуют в понятные бизнес-исключения и логируют.

- Email отправка обернута в try/catch и логируется: если email не доставлен, создаётся запись в notifications с статусом FAILED (fn\_create\_notification), и в Audit/log сохраняется причина.
  - AuditService фиксирует изменения сущностей и критические события (создание продукта, смена статуса, добавление члена холодильника).
- 

## 8. Контроль требований этапа (проверочные пункты)

- Все операции уровня хранения, требующие специфической доменной логики, реализованы через PL/pgSQL-функции (fn\_add\_product, fn\_move\_product, fn\_get\_products\_expiring\_within и т.д.). Строгое требование этапа выполнено: сервисы вызывают именно эти функции, а не выполняют аналогичных UPDATE/INSERT напрямую.
  - Уровень бизнес-логики:
    - Использует вызовы этих функций (через JdbcTemplate / native query).
    - Дополняет их валидацией, авторизацией и логикой агрегации (группировка уведомлений, формирование писем).
  - Документированы возможные ошибки, обработка и тестовые сценарии.
- 

## 9. Пример последовательности (Sequence) — «Уведомление о скором истечении срока»

1. Scheduler (NotificationService scheduled task) запускает runExpiryBatch(fridgeld, days).
2. NotificationService вызывает fn\_get\_products\_expiring\_within(fridgeld, days) → получает таблицу продуктов.
3. NotificationService группирует записи по owner\_id.
4. Для каждого owner:
  - NotificationService вызывает EmailService.sendExpiryReminder(user, products, days) (асинхронно).
  - После отправки/попытки вызывает fn\_create\_notification(userId, productId, CHANNEL=EMAIL, TEMPLATE=TTL\_T3, status=SENT/FAILED, error\_msg).
5. EmailService генерирует HTML через Thymeleaf (email/expiry.html) и вызывает mailSender.send(mime).

6. mailSender взаимодействует с SMTP; при успешном ответе SMTP возвращается 250; EmailService.log фиксирует успех, NotificationService записывает статус SENT.
- 

## 10. Особенности реализации (практические замечания)

- Триггеры выполняют часть бизнес-логики (например, products\_before\_insert\_update выполняет проверку членства и вычисляет expires\_at по категории). Поэтому сервисы не дублируют эти проверки, но могут предварительно проверять для более дружелюбных сообщений об ошибках.
- Использование DB-функций облегчает консистентность: все потоки и внешние сервисы, вызывающие базу (включая административные инструменты), будут использовать те же правила.
- Для отправки почты EmailService помечен @Service и использует @Async; MailConfig обеспечивает fallback (NoOpMailSender) для среды без SMTP, но в production следует настраивать реальные SMTP-credentials.