

# 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

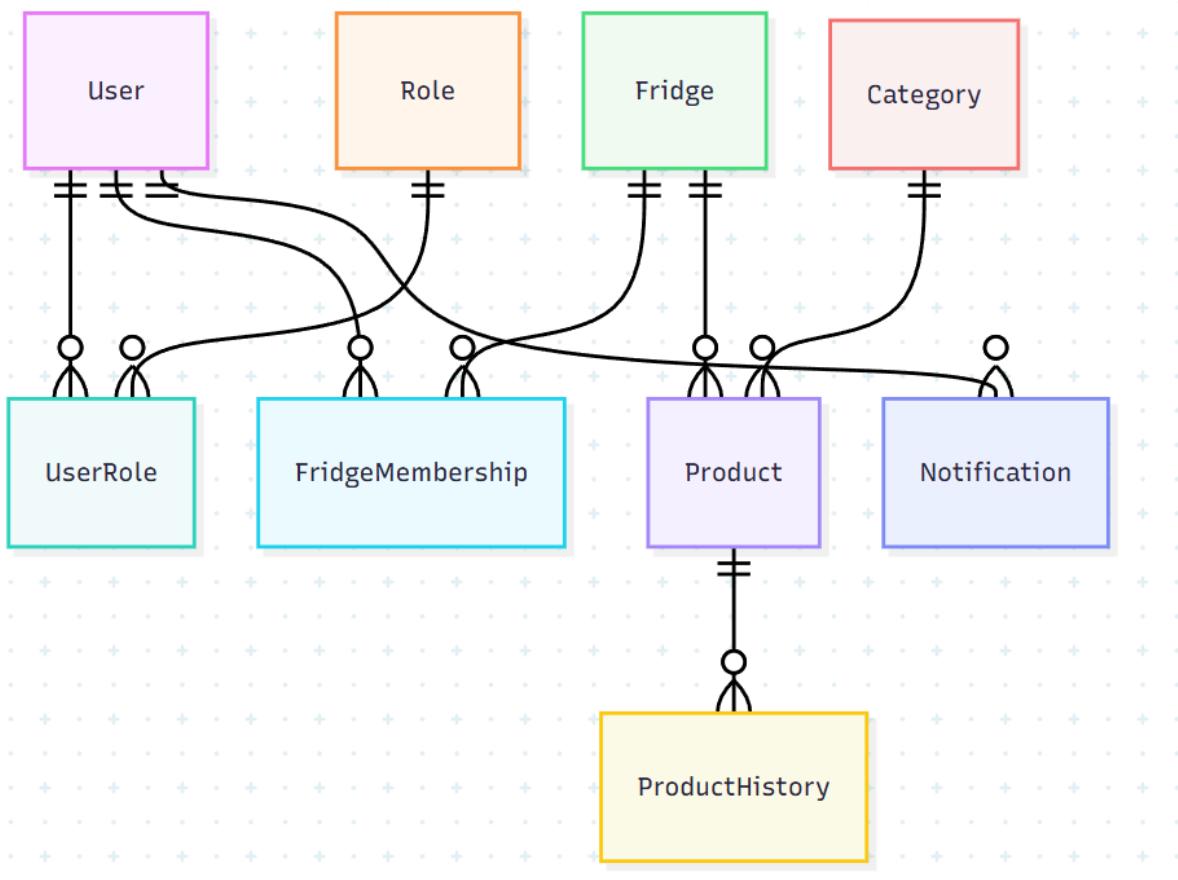
- Front: React
- Backend: Spring MVC
- БД: PostgreSQL

## 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);

```

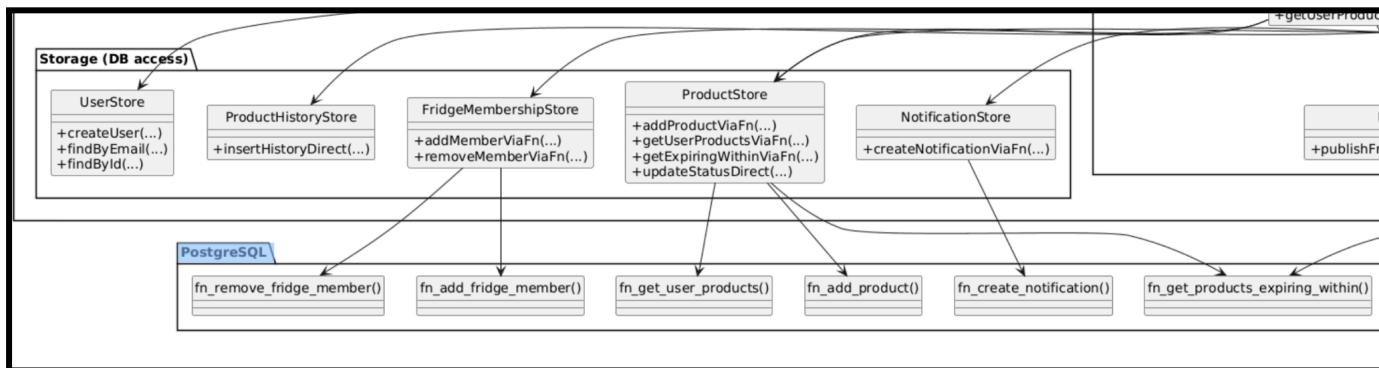
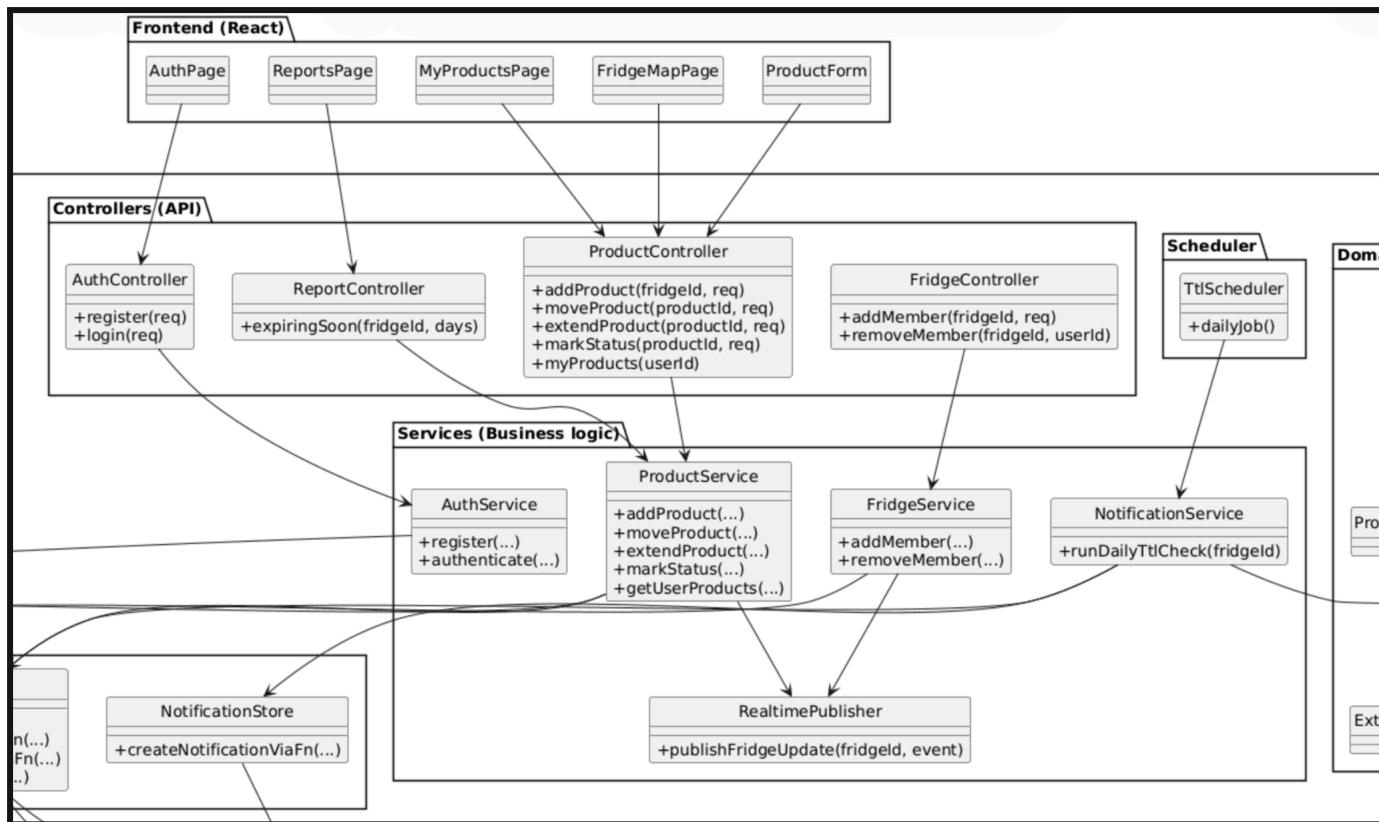
```
-- 4) Поиск продуктов внутри зоны по статусу (например: показать все
просроченные в зоне)
CREATE INDEX IF NOT EXISTS idx_products_zone_status ON products (zone_id,
status);

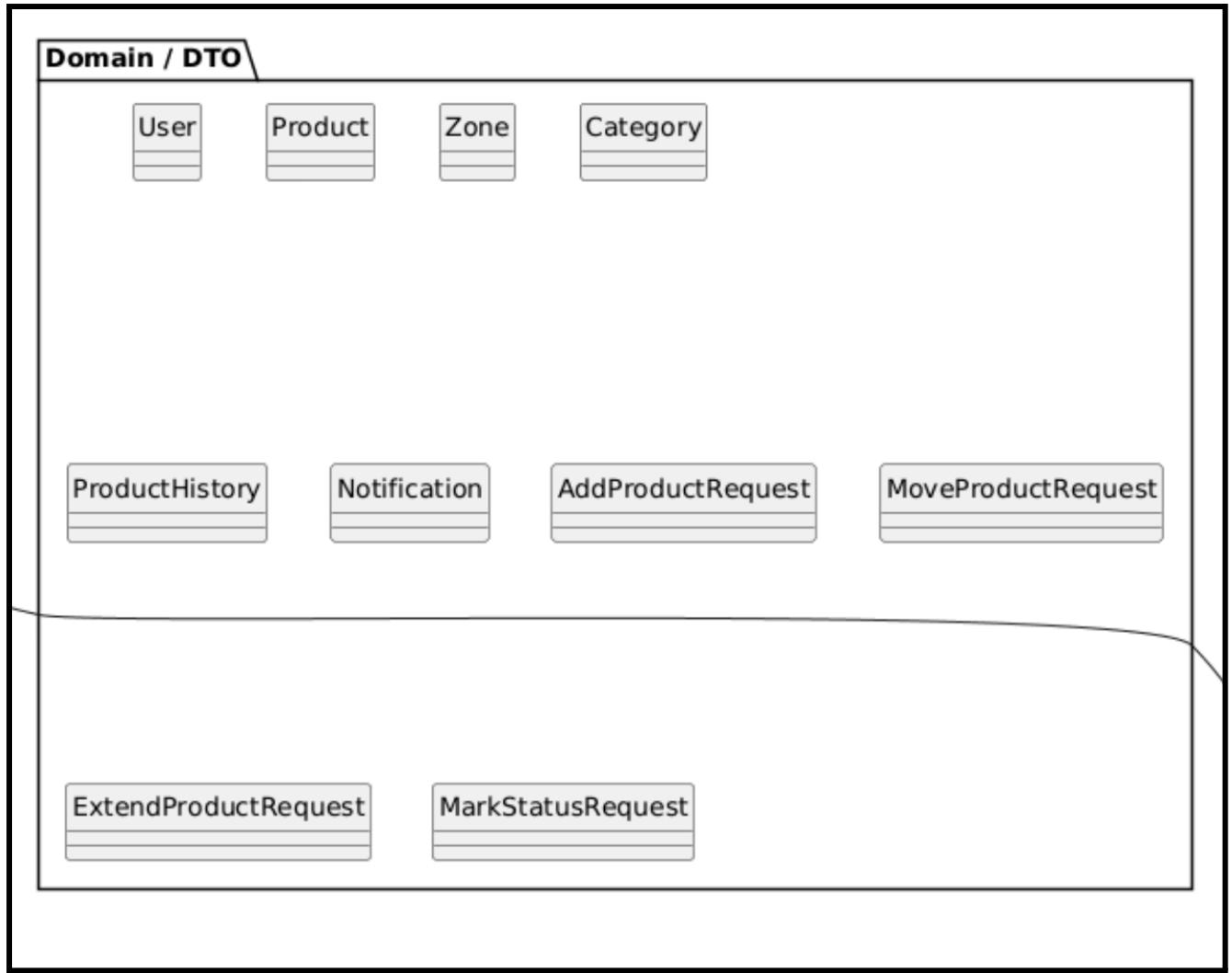
-- 5) Индекс по уведомлениям: выборка по пользователю и времени (показать
историю уведомлений)
CREATE INDEX IF NOT EXISTS idx_notifications_user_sentat ON notifications
(user_id, sent_at DESC);

-- 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





## Бизнес-логика нашей ИС.

### 1. Слой представления (web-контроллеры)

#### ProductController

Отвечает за REST-эндпоинты работы с продуктами текущего пользователя.

- `Long addProduct(AddProductRequest request)`  
Принимает данные нового продукта от клиента и передаёт их в сервисный слой, где через БД-функцию `fn_add_product` создаётся запись о продукте. Возвращает идентификатор созданного продукта.
- `List<UserProductDto> getMyProducts()`  
Возвращает список продуктов, принадлежащих текущему пользователю, используя pl/pgsql-функцию `fn_get_user_products`.
- `ProductResponse getProduct(Long productId)`  
Возвращает подробную информацию о конкретном продукте по его идентификатору.

## **FridgeController**

Реализует REST-интерфейс для управления холодильниками и их участниками.

- `FridgeResponse createFridge(CreateFridgeRequest request)`  
Создаёт новый холодильник, сохраняет его в базе данных и возвращает краткое описание созданного объекта.
- `List<FridgeResponse> getAllFridges()`  
Возвращает список всех холодильников, доступных в системе.
- `void addMember(Long fridgeId, AddFridgeMemberRequest request)`  
Добавляет пользователя в холодильник, вызывая pl/pgsql-функцию `fn_add_fridge_member`.
- `void removeMember(Long fridgeId, Long userId)`  
Логически удаляет пользователя из холодильника через функцию `fn_remove_fridge_member`.

## **UserController**

Предоставляет REST-эндпоинты для базовых операций с пользователями.

- `UserResponse createUser(CreateUserRequest request)`  
Создаёт нового пользователя, сохраняет его в таблице `users` и возвращает информацию о нём.
- `UserResponse getUser(Long userId)`  
Возвращает информацию о пользователе по его идентификатору.
- `List<UserResponse> getAllUsers()`  
Возвращает список всех пользователей системы.

## **DictionaryController**

Содержит справочные эндпоинты для фронтенда.

- `List<CategoryResponse> getCategories()`  
Возвращает список всех категорий продуктов из таблицы `categories`.
- `List<ZoneResponse> getZonesByFridge(Long fridgeId)`  
Возвращает все зоны, принадлежащие указанному холодильнику.

## **NotificationController**

Используется для запуска процесса формирования уведомлений.

- `void sendExpiryNotifications(Long fridgeId, int daysBefore)`  
Запускает бизнес-логику поиска продуктов с истекающим сроком годности и записи уведомлений в таблицу `notifications` через pl/pgsql-функции.

## 2. Сервисный слой (бизнес-логика)

### **ProductService**

Определяет операции над продуктами на уровне бизнес-логики.

- Long addProduct(...) – создание продукта.
- List<UserProductDto> getUserProducts(Long ownerId) – получение продуктов пользователя.
- ProductResponse getProduct(Long productId) – получение подробностей по одному продукту.

### **ProductServiceImpl**

Реализует бизнес-логику работы с продуктами.

- addProduct(...)  
Выполняет базовую валидацию и вызывает ProductFunctionRepository.addProduct, который обращается к pl/pgsql-функции fn\_add\_product. Статус продукта задаётся как ACTIVE, а поле placed\_at рассчитывается на стороне БД.
- getUserProducts(Long ownerId)  
Делегирует получение списка продуктов текущего пользователя функции fn\_get\_user\_products.
- getProduct(Long productId)  
Загружает сущность Product через ProductRepository и маппит её в DTO ProductResponse.

### **FridgeService**

Описывает операции с холодильниками и участниками.

- FridgeResponse createFridge(...) – создание холодильника.
- List<FridgeResponse> getAllFridges() – список холодильников.
- void addMember(Long fridgeId, Long userId, boolean moderator) – добавление участника.
- void removeMember(Long fridgeId, Long userId) – удаление участника.

### **FridgeServiceImpl**

Реализует бизнес-логику для работы с холодильниками.

- createFridge(...)  
Формирует и сохраняет сущность Fridge, устанавливая дату создания и настройки доступа. Возвращает DTO с основными полями.

- `getAllFridges()`  
Получает список всех холодильников через `FridgeRepository` и преобразует их в `FridgeResponse`.
- `addMember(Long fridgeId, Long userId, boolean moderator)`  
Добавляет пользователя в холодильник, вызывая `pl/pgsql`-функцию `fn_add_fridge_member` через `FridgeMembershipFunctionRepository`.
- `removeMember(Long fridgeId, Long userId)`  
Логически завершает участие пользователя в холодильнике через функцию `fn_remove_fridge_member`.

### **UserService**

Описывает операции с пользователями.

- `UserResponse createUser(CreateUserRequest request)` – создание нового пользователя.
- `UserResponse getUser(Long userId)` – получение одного пользователя.
- `List<UserResponse> getAllUsers()` – список всех пользователей.

### **UserServiceImpl**

Реализует бизнес-логику работы с пользователями.

- `createUser(CreateUserRequest request)`  
Создаёт сущность `User` из входных данных, задаёт дату создания и настройки уведомлений, сохраняет в БД и возвращает DTO `UserResponse`.
- `getUser(Long userId)`  
Находит пользователя по идентификатору и возвращает его как DTO.
- `getAllUsers()`  
Возвращает список всех пользователей, преобразованный в представление `UserResponse`.

### **CategoryService / CategoryServiceImpl**

Интерфейс и реализация для справочника категорий.

- `List<CategoryResponse> getAllCategories()`  
Загружает все записи из таблицы `categories` и возвращает их в виде DTO. Используется для заполнения справочников в интерфейсе.

### **ZoneService/ZoneServiceImpl**

Интерфейс и реализация для зон холодильника.

- `List<ZoneResponse> getZonesByFridge(Long fridgeId)`  
Находит все зоны, принадлежащие заданному холодильнику, и возвращает их как удобные DTO для фронтенда.

## **NotificationService/NotificationServiceImpl**

Сервис для работы с уведомлениями.

- `void sendExpiryNotificationsForFridge(Long fridgeId, int daysBefore)`  
Получает список продуктов с истекающим сроком годности через pl/pgsql-функцию `fn_get_products_expiring_within`, подбирает тип шаблона уведомления и для каждого продукта создаёт запись в таблице `notifications` с помощью функции `fn_create_notification`.

## **3. Репозитории**

JPA-репозитории

- `UserRepository`  
Наследуется от `JpaRepository<User, Long>`, предоставляет стандартные CRUD-операции с пользователями. Дополнительно содержит метод `findByEmail` для поиска пользователя по e-mail.
- `FridgeRepository`  
`JpaRepository<Fridge, Long>` – CRUD для холодильников.
- `ZoneRepository`  
`JpaRepository<Zone, Long>` с методом `findByFridge_FridgeId`, возвращающим все зоны конкретного холодильника.
- `CategoryRepository`  
`JpaRepository<Category, Long>` – доступ к справочнику категорий.
- `ProductRepository`  
`JpaRepository<Product, Long>` – базовые операции над сущностями продуктов.
- `NotificationRepository`  
`JpaRepository<Notification, Long>` – сохранение и чтение уведомлений (используется при необходимости анализа или отладки).

Function-репозитории (вызовы pl/pgsql-функций)

- `ProductFunctionRepository / ProductFunctionRepositoryImpl`  
**Отвечает за вызовы БД-функций, связанных с продуктами:**
  - `addProduct(...)` – вызывает `fn_add_product` для создания продукта с учётом триггеров и бизнес-правил на стороне БД;
  - `getProductsExpiringWithin(...)` – обращается к `fn_get_products_expiring_within` для поиска продуктов с истекающим сроком годности;
  - `getUserProducts(...)` – вызывает `fn_get_user_products` и возвращает проекции `UserProductDto`.
- `FridgeMembershipFunctionRepository / FridgeMembershipFunctionRepositoryImpl`

Инкапсулирует вызовы функций, управляющих участием пользователей в холодильнике:

- `addFridgeMember(...)` – вызывает `fn_add_fridge_member` для добавления или восстановления участника;
  - `removeFridgeMember(...)` – вызывает `fn_remove_fridge_member` для отметки выхода пользователя.
  - **NotificationFunctionRepository / NotificationFunctionRepositoryImpl**  
Отвечает за запись сведений об отправленных уведомлениях:
    - `createNotification(...)` – вызывает `fn_create_notification` и создаёт запись в таблице `notifications` с типом канала, шаблоном и статусом отправки.
- 

## 4. DTO-классы

- DTO уровня API (`CreateUserRequest`, `UserResponse`, `CreateFridgeRequest`, `FridgeResponse`, `AddProductRequest`, `ProductResponse`, `CategoryResponse`, `ZoneResponse`, `AddFridgeMemberRequest`, `NotificationResponse`)  
Используются для обмена данными между клиентом и сервером: описывают структуру входных запросов и выходных ответов REST-контроллеров.
- DTO уровня БД (`UserProductDto`, `ExpiringProductDto`)  
Представляют собой проекции, соответствующие результатам вызовов `pl/pgsql`-функций. Используются сервисным слоем для реализации бизнес-логики без необходимости маппить каждую строку результата вручную.