

ОГЛАВЛЕНИЕ

ОГЛАВЛЕНИЕ	1
ВВЕДЕНИЕ	2
1. Серверная составляющая	4
1.1. Директория БД.....	4
1.2. Настройщик сервера	7
2. IOS-клиент	10
2.1. Основные экраны	10
ДИСКЛЕЙМЕР	17

ВВЕДЕНИЕ

Прототип состоит из 2-х частей: серверная [1] – отвечает за хранение данных и IOS-клиент [2] – отвечает за отображение и обработку серверных данных. Для удобства навигации, предлагается ознакомиться со структурой 2-х частей:

Сервер

{BASE_DIR} = /User/.../portiondemo/backend

[1] {BASE_DIR} (*Директория по умолчанию*)

[1.1] {BASE_DIR}/database (*Директория БД*)

[1.1.1] {BASE_DIR}/database/orders

[1.1.1.1] allpurch.xlsx

[1.1.1.2] mainpurch.xlsx

[1.1.1.3] otherpurch.xlsx

[1.1.1.4] {BASE_DIR}/database/orders/support

[1.1.1.4.1] {BASE_DIR}/database/orders/support/html_new_ids

[1.1.1.4.2] {BASE_DIR}/database/orders/support/orderhtml

[1.1.2] {BASE_DIR}/database /products (*Директория для хранения информации о продуктах, в рамках прототипа*)

[1.1.2.1] appdb2.xlsx

[1.1.2.2] prodlinks.xlsx

[1.1.2.3] {BASE_DIR}/database/products/images

[1.1.3] {BASE_DIR}/database/users (*Отдельная директория для рациона**)

[1.1.3.1] Rationinfo.xlsx

[1.2] {BASE_DIR}/src (*Директория для настройки сервера*)

[1.2.1] main_server.py

[1.2.2] {BASE_DIR}/src /modules

[1.2.2.1] __init__.py

[1.2.2.2] api_routes.py

[1.2.2.3] appdb_updater.py

[1.2.2.4] check_processor.py

[1.2.2.5] database_handler.py

[1.2.2.6] images_handler.py

[1.2.2.7] server_order_creator.py

[1.2.2.8] server_ration_handler.py

IOS-клиент

{BASE_DIR} = /User/.../portiondemo/ios-app

[2] {BASE_DIR}/project-portion-demo (*Директория проекта*)

[2.1] {BASE_DIR}/Project-Portion-demo/Screens (*Директория для основных экранов*) LoadingScreen.swift

[2.1.1] {BASE_DIR}/Project-Portion-demo/Screens/Account
(*Экран «Аккаунт»*)

[2.1.1.1] AccountView.swift

[2.1.2] {BASE_DIR}/Project-Portion-demo/Screens/Fridge
(*Экран «Запасы»*)

[2.1.2.1] FridgeBanner.swift

[2.1.2.2] FridgeView.swift

[2.1.2.3] {BASE_DIR}/Project-Portion-demo/Screens/Fridge/Order
(*Поп-ап элемент*)

[2.1.2.3.1] CheckScanner.swift

[2.1.2.3.2] CreatorView.swift

[2.1.3] {BASE_DIR}/Project-Portion-demo/Screens/Statistic
(*Экран «Статистика»*)

[2.1.3.1] StatisticView.swift

[2.1.4] {BASE_DIR}/Project-Portion-demo/Screens/Ration
(*Экран «Рацион» и pop-up*)

[2.1.4.1] RationPopUp.swift

[2.1.4.2] RationView.swift

[2.2] {BASE_DIR}/Project-Portion-demo/SwiftData (*Настройка работы с данными*)

[2.2.1.1] Models.swift

[2.2.1.2] ProductImages.swift

[2.3] {BASE_DIR}/Project-Portion-demo/Templates
(*Вспомогательные элементы*)

[2.3.1] CpcfBanner.swift

[2.3.2] SearchBlocks.swift

[2.3.3] ProductCards.swift

[2.4] ContentView.swift

[2.5] Styles.swift

[2.6] TestUI2App.swift

1. Серверная составляющая

1.1. Директория БД

[1.1.1]/mainpurch.xlsx

Основной файл текущих запасов продуктов в «холодильнике» (из Основного магазина, StoreID = 1). Записи агрегируются по ключу (ProdID, FamilyID/UserID). Основные атрибуты: ProdID, Name, Volume, Unit, VolumeGr, Kcal100g, Prot100g, Fat100g, Carb100g, ExpireDate, Tag, Cat, Store, StoreID (всегда 1), Date, FamilyID, TotalCost, Address, AddressID, Count, TotalVolume, TotalVolumeGr, TotalCostPerCount, UserID. Используется:

- [1.2.2]/api_routes.py: эндпоинты /get_main_purch, /update_main_purch.
- [1.2.2]/database_handler.py: функции: get_purch_data, _aggregate_main_purch.
- [1.2.2]/server_order_creator.py: функция update_main_purch_with_aggregation.
- [2.4.2]/FridgeView.swift и [2.4.4]/RationPopUp.swift: как источник данных для отображения "запасов".
- [2.4.2]/FridgeBanner.swift: для расчета суммарной пищевой ценности.

[1.1.1]/otherpurch.xlsx

Файл запасов продуктов из других магазинов (StoreID != 1). Записи агрегируются по ключу (ProdID, FamilyID/UserID, StoreID, Date). Атрибуты аналогичны mainpurch. Используется:

- [1.2.2]/api_routes.py: эндпоинты /get_other_purch, /update_other_purch.
- [1.2.2]/database_handler.py: функции get_purch_data, _aggregate_other_purch.
- [1.2.2]/server_order_creator.py: функция _save_to_other_purch.

- [2.4.4]/RationPopUp.swift: вкладка "Заказы" для добавления в рацион.

[1.1.1]/allpurch.xlsx

Детализированный журнал всех покупок. Содержит данные о входящих заказах. Атрибуты включают все поля из mainpurch и otherpurch, а также Count, TotalVolume, TotalVolumeGr, TotalCostPerCount. Используется:

- [1.2.2]/api_routes.py: эндпоинт /get_allpurch_by_daterange.
- [1.2.2]/database_handler.py: функция get_allpurch_by_daterange.
- [1.2.2]/server_order_creator.py: функция _save_to_all_purch.
- [2.4.3]/StatisticView.swift: для анализа финансовых трат за период.

[1.1.2]/appdb2.xlsx

Основная база данных товаров (справочник). Атрибуты: ProdID, plu, Name, Volume, Unit, VolumeGr, Kcal100g, Prot100g, Fat100g, Carb100g, Tag, Cat, Store, StoreID, TotalCost, image (URL). Используется:

- [1.2.2]/api_routes.py: эндпоинты /search_products, /lavka/process_order.
- [1.2.2]/appdb_updater.py: как целевой файл для добавления новых товаров.
- [1.2.2]/database_handler.py: функция search_products.
- [2.4.2]/CreatorView.swift: для поиска и добавления товаров в заказ.
- [2.4.4]/RationPopUp.swift: как источник данных о продуктах.

[1.1.2]/prodlinks.xlsx

Вспомогательный файл сопоставления ProdID с внешними ссылками на продукты. Атрибуты: ProdID, ProductURL. Используется:

- [1.2.2]/api_routes.py: эндпоинт /product_link/<prod_id>.

[1.1.3]/rationinfo.xlsx

Файл дневного рациона пользователей. Атрибуты: ProdID, Name, Volume, Unit, VolumeGr, Kcal100g, Prot100g, Fat100g, Carb100g, ExpireDate, Tag, Cat, MealID, MealName, RationDate, VolumeServ, VolumeServGr, KcalServ, ProtServ, FatServ, CarbServ, UserID, CreatedAt. Используется:

- [1.2.2]/api_routes.py: эндпоинты /add_to_ration, /get_ration_by_date, /get_ration_by_daterange.
- [1.2.2]/database_handler.py: функции save_to_ration_info, get_family_ration.
- [1.2.2]/server_ration_handler.py: как основной файл для всех операций с рационом.
- [2.4.4]/RationView.swift: для отображения и редактирования дневного рациона.
- [2.4.3]/StatisticView.swift: для анализа потребления за период.

[1.1.1.4]/orderhtml

Директория для хранения исходных HTML-страниц заказов из магазина, скачанных модулем [1.2.2]/check_processor.py.

[1.1.1.4]/html_new_ids

Директория для хранения HTML-страниц товаров, скачанных модулем [1.2.2]/appdb_updater.py для последующего парсинга.

[1.1.2]/images

Директория для локального хранения изображений продуктов. Имена файлов соответствуют ProdID.

1.2. Настройщик сервера

[1.2]/main_server.py

Главный скрипт Flask-сервера. Определяет базовые пути к директориям данных ([1.1]), инициализирует все модули обработки ([1.2.2]) и регистрирует API маршруты через [1.2.2]/api_routes.py. Запускает сервер на всех интерфейсах (0.0.0.0) порт 8000. Взаимодействует напрямую только с другими файлами из папки [1.2]/src.

[1.2.2]/api_routes.py

Модуль, определяющий все эндпоинты REST API сервера. Регистрирует логику обработки запросов от iOS-приложения, делегируя вызовы соответствующим модулям:

- Работа с покупками (/get_main_purch, /get_other_purch, /update_main_purch) [1.2.2]/database_handler.py.
- Создание заказа (/create_order) [1.2.2]/server_order_creator.py.
- Обработка чека Лавки (/lavka/process_order) [1.2.2]/check_processor.py и [1.2.2]/appdb_updater.py.
- Работа с рационом (/add_to_ration, /get_ration_by_date) [1.2.2]/server_ration_handler.py.
- Поиск товаров (/search_products) [1.2.2]/database_handler.py.
- Отдача изображений (/image/<prod_id>) [1.2.2]/images_handler.py.

Все эндпоинты возвращают данные в формате JSON, который парсится соответствующими ViewModel в iOS-приложении ([2.4.2], [2.4.4] и др.).

[1.2.2]/database_handler.py

Универсальный обработчик для чтения и записи Excel файлов ([1.1.1], [1.1.2], [1.1.3]). Содержит основную логику фильтрации данных по FamilyID/UserID (с учетом типа аккаунта UserAccType) и агрегации записей для mainpurch и otherpurch. Не содержит бизнес-логики создания сущностей, только операции с таблицами. Вызывается из:

- [1.2.2]/api_routes.py
- [1.2.2]/server_order_creator.py
- [1.2.2]/server_ration_handler.py.

[1.2.2]/server_order_creator.py

Серверная имитация логики создания заказа. Принимает данные корзины из iOS, ищет продукты в [1.1.2]/appdb2.xlsx и создает детализированные записи в [1.1.1]/allpurch.xlsx, одновременно обновляя агрегированные данные в [1.1.1]/mainpurch.xlsx или [1.1.1]/otherpurch.xlsx. Ключ агрегации включает UserID. Используется эндпоинтом /create_order.

[1.2.2]/server_ration_handler.py

Обработчик операций с дневным рационом. Добавляет записи в [1.1.3]/rationinfo.xlsx и извлекает из него данные, фильтруя по дате и UserID. Вызывается из [1.2.2]/api_routes.py для соответствующих эндпоинтов.

[1.2.2]/check_processor.py

Модуль для первичной обработки ссылки на заказ из Основного магазина. Скачивает HTML страницу заказа в [1.1.1.4]/orderhtml,

извлекает Plu товаров и их количество. Проверяет наличие Plu в [1.1.2]/appdb2.xlsx. Новые PLU передает в [1.2.2]/appdb_updater.py.

[1.2.2]/appdb_updater.py

Модуль для обновления базы товаров. Скачивает HTML-страницы новых товаров в [1.1.1.4]/html_new_ids, парсит данные (название, категорию, питательность, цену, URL изображения) и добавляет их в [1.1.2]/appdb2.xlsx. При необходимости скачивает изображения в [1.1.2]/images. Работает как вторая ступень после [1.2.2]/check_processor.py в рамках /lavka/process_order.

[1.2.2]/images_handler.py

Простой модуль для отдачи статических изображений из директории [1.1.2]/images по запросу к эндпоинту /image/<prod_id>.

[1.2.2]/init.py

Файл инициализации пакета модулей, экспортирующий все основные классы для импорта в [1.2]/main_server.py.

2. IOS-клиент

2.1. Основные экраны

[2.1.1]/AccountView.swift

Экран служит для управления профилем пользователя в приложении. Основная функция – сбор и хранение персональных данных, необходимых для персонализации: пол, возраст, рост, вес, уровень активности и цель (дефицит, баланс, профицит калорий). На основе этих данных в реальном времени рассчитывается оптимальный суточный рацион (калории, белки, жиры, углеводы) по формулам, заложенным в ViewModel. Эти значения сохраняются в модели [UserInfo] и [RationOptimum] из [2.2.1]/Models.swift.

Экран также управляет типом аккаунта: личный или семейный (UserAccType). Для семейного аккаунта требуется ввод идентификатора семьи (FamilyID). При изменении типа аккаунта или данных пользователя значения оптимального рациона пересчитываются и сохраняются в локальное хранилище (SwiftData) через ModelContext.

Экран связан с другими через общую модель пользователя: данные, введенные здесь, используются в [2.1.2]/FridgeView для баннера и в [2.1.3]/StatisticView и [2.1.4]/RationView для расчетов отклонения от нормы.

[2.1.2]/FridgeView.swift

FridgeView, центральный экран для работы с продуктами, находящимися в «запасах» (основные покупки). Он реализует паттерн MVVM с ViewModel [FridgeViewModel], которая инкапсулирует всю бизнес-логику загрузки и обработки данных.

Экран инициирует загрузку данных с сервера при появлении и по запросу пользователя (pull-to-refresh). Запрос к endpoint `/get_main_purch` в `[1.2.2]/api_routes.py` передает параметры `family_id` и `user_id` в зависимости от типа аккаунта пользователя `[UserInfo]`. Функция `get_purch_data` в `[1.2.2]/database_handler.py` читает файл `mainpurch.xlsx` из `[1.1.1]/mainpurch.xlsx`, фильтрует записи и возвращает их в виде кортежа, содержащего 20 полей. Полученные данные кэшируются в `ViewModel`.

Логика группировки продуктов происходит в несколько этапов: сначала выделяются "скоро истекающие" продукты, затем остальные группируются по категории (`Cat`) или предпочтению (`PrefMeal`). Для семейных аккаунтов дополнительно выполняется группировка по `UserID`. Отображение продуктов использует компоненты из `[2.3]/ProductCards.swift`.

Из меню экрана доступны переходы к `[2.1.2]/CreatorView` (создание заказа) и `[2.1.2]/CheckScanner` (обработка чеков Основного магазина). Экран связан с серверными модулями `[1.2.2]/database_handler.py` и `[1.2.2]/server_order_creator.py`.

[2.1.2]/FridgeBanner.swift

`FridgeBanner` является верхним баннером на экране `[2.1.2]/FridgeView`. Его задача – агрегировать данные о всех продуктах в разделе "Запасы" (`mainpurch`) и представлять суммарную пищевую ценность запасов в холодильнике.

Баннер получает массив продуктов типа `[MainPurch]` и вычисляет общую сумму калорий, белков, жиров и углеводов. Затем, используя оптимальную суточную норму калорий пользователя (передаваемую параметром `userKcalOpt` из модели `[UserInfo]`), рассчитывает, на сколько

полноценных рационов хватит текущих запасов. Этот расчет отображается в правой части баннера, левая часть показывает цифры БЖУ.

Данные для баннера готовятся на уровне [2.1.2]/FridgeView, который запрашивает их с сервера через endpoint /get_main_purch. Логика обработки этого запроса находится в [1.2.2]/api_routes.py и использует [1.2.2]/database_handler.py для чтения файла mainpurch.xlsx.

[2.1.2]/CheckScanner.swift

CheckScanner предоставляет интерфейс для автоматизации добавления покупок из Основного магазина в систему. Пользователь копирует ссылку из истории заказов и вставляет ее в текстовое поле на экране.

При нажатии кнопки "Обработать" экран формирует POST-запрос на endpoint /lavka/process_order в [1.2.2]/api_routes.py. В запрос включаются: ссылка на заказ, family_id, user_id и address_id. Endpoint запускает последовательную цепочку обработки:

[1.2.2]/check_processor.py: Скачивает HTML-страницу заказа, извлекает ID товаров (plu) и их количество, проверяет наличие этих ID в базе товаров [1.1.2]/appdb2.xlsx.

[1.2.2]/appdb_updater.py: Для новых ID загружает страницы товаров, парсит данные (название, категорию, цену, изображение) и добавляет их в appdb2.xlsx. При необходимости скачивает изображения в [1.1.2]/images.

На основе обработанных товаров и их количества формируется запрос на внутренний endpoint /create_order (обрабатывается [1.2.2]/server_order_creator.py), что приводит к обновлению файлов [1.1.1]/allpurch.xlsx, [1.1.1]/mainpurch.xlsx и [1.1.1]/otherpurch.xlsx.

[2.1.2]/CreatorView.swift

Позволяет пользователю вручную собрать заказ из товаров, имеющихся в общей базе [1.1.2]/appdb2.xlsx. Экран имеет два режима: "Поиск" для добавления товаров в корзину и "Корзина" для просмотра и подтверждения заказа.

В режиме поиска при вводе текста отправляется запрос на endpoint /search_products в [1.2.2]/api_routes.py. Модуль [1.2.2]/database_handler.py выполняет поиск по файлу appdb2.xlsx и возвращает результаты, которые группируются по StoreID. Товары с StoreID = 1 отображаются во вкладке "Основной", остальные – в "Прочее". Пользователь регулирует количество каждого товара с помощью стэппера.

При подтверждении заказа экран формирует структура данных, содержащую family_id, address_id, order_date, user_id и массив выбранных товаров с их количествами. Эти данные отправляются на endpoint /create_order.

Модуль [1.2.2]/server_order_creator.py обрабатывает этот запрос: добавляет детализированные записи в [1.1.1]/allpurch.xlsx, а также агрегирует и обновляет данные в [1.1.1]/mainpurch.xlsx (для StoreID = 1) или [1.1.1]/otherpurch.xlsx (для других магазинов) с учетом UserID и FamilyID. Таким образом, экран является основным клиентским инструментом для пополнения запасов через серверную логику создания заказов.

[2.1.4]/RationView.swift

«Рацион» – главный экран для работы с дневным рационом. Пользователь выбирает дату, для которой нужно просмотреть рацион.

При смене даты экран отправляет запрос на endpoint `/get_ration_by_date` в `[1.2.2]/api_routes.py`, передавая `ration_date` и `user_id`. Модуль `[1.2.2]/server_ration_handler.py` читает файл `[1.1.3]/rationinfo.xlsx`, фильтрует записи по дате (`selectedDate`) и `UserID` и возвращает данные. Полученные записи группируются по приемам пищи (`MealID`), которые управляются локальной моделью `[MealList]` из `[2.2.1]/Models.swift`.

Верхняя часть экрана показывает сводку "Итого рацион": суммарные калории и БЖУ по всем приемам пищи за день. Эти значения сравниваются с оптимальным рационом пользователя, который берется из локальной модели `[RationOptimum]` (создается автоматически при первом обращении к дате) или из `[UserInfo]`. На основе отклонений по нутриентам формируется текстовая рекомендация.

Для каждого приема пищи отображается блок с списком добавленных продуктов (используются карточки из `[2.3]/ProductCards.swift`) и кнопкой «+». Нажатие на «+» открывает pop-up `[2.1.4]/RationPopUp.swift`, передавая ему `mealID` и `selectedDate`.

[2.1.4]/RationPopUp.swift

`RationPopUp` открывается из экрана `[2.1.4]/RationView` для конкретного приема пищи (`mealID`) и выбранной даты (`selectedDate`). Окно имеет три вкладки: "Запасы", "Заказы", "Вручную".

Вкладка "Запасы" получает данные через тот же механизм, что и `[2.1.2]/FridgeView` (endpoint `/get_main_purch`), группируя продукты по категориям и выделяя скоро истекающие. Вкладка "Заказы" получает данные через endpoint `/get_other_purch`, группируя записи по дате заказа и магазину.

При выборе продукта пользователь вводит объем порции. Нажатие кнопки добавления формирует запрос к endpoint `/add_to_ration` в `[1.2.2]/api_routes.py`. Запрос включает полные данные о продукте, указанный объем, `mealID`, дату рациона и `user_id`. Модуль `[1.2.2]/server_ration_handler.py` обрабатывает этот запрос, сохраняя запись в файл `[1.1.3]/rationinfo.xlsx`.

Одновременно, для продуктов из запасов (`StoreID = 1`) отправляется запрос на endpoint `/update_main_purch`, а для продуктов из заказов (`StoreID != 1`) – на `/update_other_purch`. Эти endpoints, реализованные в `[1.2.2]/api_routes.py`, уменьшают количество (объем) доступного продукта в соответствующих файлах `[1.1.1]/mainpurch.xlsx` или `[1.1.1]/otherpurch.xlsx` через `[1.2.2]/database_handler.py`. Таким образом, окно связывает сервисы рациона `[1.2.2]/server_ration_handler.py` и управления запасами `[1.2.2]/database_handler.py`.

[2.1.3]/StatisticView.swift

StatisticView предоставляет аналитику в двух разрезах: "Здоровье" и "Финансы". Пользователь выбирает дату, и настраивает период для отображения (Неделя, Месяц или Год).

Для анализа "Здоровье" экран загружает данные о рационе за период, отправляя `start_date`, `end_date` и `user_id` на endpoint `/get_ration_by_daterange` в `[1.2.2]/api_routes.py`. Этот endpoint использует `[1.2.2]/server_ration_handler.py` для чтения и фильтрации данных из файла `[1.1.3]/rationinfo.xlsx`. Рассчитывается среднее суточное потребление калорий и БЖУ, которое сравнивается с оптимальными значениями пользователя (из локальной модели `[RationOptimum]` или `[UserInfo]`). На основе отклонения формируется текстовая рекомендация.

Для анализа "Финансы" экран загружает данные о всех покупках за период через endpoint `/get_allpurch_by_daterange`. Этот endpoint использует функцию `get_allpurch_by_daterange` из `[1.2.2]/database_handler.py` для чтения и фильтрации файла `[1.1.1]/allpurch.xlsx` с учетом типа аккаунта (`user_acc_type`). Рассчитываются общие затраты, распределение расходов по категориям товаров (Cat), средняя стоимость 100 ккал и прогнозная стоимость оптимального суточного рациона.

Экран только запрашивает и анализирует информацию с сервера при помощи модулей `[1.2.2]/server_ration_handler.py` и `[1.2.2]/database_handler.py` через соответствующие запросы.

ДИСКЛЕЙМЕР

К проекту прилагаются:

- Презентация проекта (файл: Презентация2.pdf)
- Видеодемонстрация пользовательского процесса (файл Демонстрация.mp4)
- Ссылка на репозиторий исходного кода:
<https://github.com/sukhoves/portiondemo>

!!!ВАЖНО!!!

Полная версия прототипа, продемонстрированная в материалах «Презентация2.pdf» и «Демонстрация.mp4», использует изображения продуктов и логотипы исключительно в качестве визуального наполнения интерфейса для демонстрации логики работы приложения.

Все связанные с ними товарные знаки принадлежат их правообладателям. Использование этих элементов не носит коммерческого характера. Демонстрационные материалы созданы на основе информации, находящейся в открытом доступе.

Исходный код, публикуемый в репозитории github, представляет собой упрощенный публичный прототип. В этой версии функционал, связанный с обработкой и использованием сторонних изображений и данных, был удален. Абсолютно все изображения, логотипы и визуальное наполнение интерфейса заменены на шаблонные элементы. В частности, в публикуемом прототипе отсутствуют серверные модули [1.2.2]/appdb_updater.py, [1.2.2]/check_processor.py и связанные с ними endpoint в [1.2.2]/api_routes.py.