МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет прикладной математики, информатики и механики

Кафедра программного обеспечения и администрирования информационных

систем

**Система контроля сахарного диабета I типа «Компенсация»**

Дипломная работа

по специальности 010503 – математическое обеспечение

и администрирование информационных систем,

специализация 351504 – администрирование информационных систем

|  |  |  |
| --- | --- | --- |
| Допущено к защите в ГАК |  | \_\_\_\_.\_\_\_\_.2014 |
| Зав. кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_ | д. ф.-м. н., проф. | Артёмов М. А. |
| Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_ | к. ф.-м.н., доц. | Михайлова Е. Е. |
| Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | 5 курс, 91 группа | Кренёв Н. Ю. |

Воронеж 2014

**Содержание**

[Введение 4](#_Toc390441113)

[1. Постановка задачи 6](#_Toc390441114)

[2. Анализ задачи 8](#_Toc390441115)

[2.1. REST-сервис 8](#_Toc390441116)

[2.2. Общая функциональность клиентских приложений 8](#_Toc390441117)

[2.3. Локальное приложение 16](#_Toc390441118)

[2.4. Web-приложение 20](#_Toc390441119)

[2.5. Мобильное приложение 20](#_Toc390441120)

[3. Средства реализации 21](#_Toc390441121)

[4. Требования к аппаратному и программному обеспечению 23](#_Toc390441122)

[4.1. Требования для REST-сервиса 23](#_Toc390441123)

[4.2. Требования для локального приложения 23](#_Toc390441124)

[4.3. Требования для web-приложения 23](#_Toc390441125)

[4.4. Требования для мобильного приложения 24](#_Toc390441126)

[5. Интерфейс пользователя 25](#_Toc390441127)

[5.1. Интерфейс локального приложения 25](#_Toc390441128)

[5.2. Интерфейс web-приложения 42](#_Toc390441129)

[5.3. Интерфейс мобильного приложения 47](#_Toc390441130)

[6. Реализация 57](#_Toc390441131)

[6.1. Структура информационной системы 57](#_Toc390441132)

[6.2. Структура баз данных 58](#_Toc390441133)

[6.3. Форматы данных 59](#_Toc390441134)

[6.4. Реализация REST-сервиса 59](#_Toc390441135)

[6.5. Реализация общей библиотеки классов 68](#_Toc390441136)

[6.6. Реализация локального приложения 73](#_Toc390441137)

[6.7. Реализация web-приложения 78](#_Toc390441138)

[6.8. Реализация мобильного приложения 80](#_Toc390441139)

[7. План тестирования 83](#_Toc390441140)

[7.1. Тестирование локального приложения 83](#_Toc390441141)

[7.2. Тестирование web-приложения 84](#_Toc390441142)

[7.3. Тестирование мобильного приложения 85](#_Toc390441143)

[Заключение 87](#_Toc390441144)

[Список литературы 89](#_Toc390441145)

[Приложение 1. Описание таблиц БД 91](#_Toc390441146)

[Приложение 2. Форматы данных 93](#_Toc390441147)

[Приложение 3. Листинг модуля AnalyzeCoreImpl.java 98](#_Toc390441148)

[Приложение 4. Листинг модуля SyncService.java 104](#_Toc390441149)

Введение

Сахарный диабет I типа – хроническое заболевание поджелудочной железы, выраженное в недостаточной секреции инсулина или её отсутствии. Инсулин необходим для усвоения углеводов, содержащихся в пище. В организме здорового человека количество выделяемого инсулина строго соответствует количеству употребляемых углеводов, в результате чего уровень сахара крови остаётся на одном уровне.

При сахарном диабете I типа производство инсулина поджелудочной железой нарушается или прекращается вовсе, поэтому пациентам необходимо делать инъекции инсулина перед каждым приёмом пищи. При этом дозировка препарата определяется индивидуально и напрямую зависит от того, что и в каком количестве пациент собирается съесть, а также времени суток, значения сахара крови перед едой, физической активности и многих других факторов [1].

Таким образом, ключевой задачей в компенсации сахарного диабета является учёт рациона пациента и точный расчёт доз, что необходимо для стабильного управления уровнем сахара крови. Для решения этой задачи эндокринологи обычно рекомендуют вести дневник, куда пациент записывает значения сахара крови, а также информацию об инъекциях инсулина и всей съеденной пище.

Традиционно для этих целей применяются бумажные дневники с жёсткой структуризацией записей по времени и некоторым другим формальным параметрам (например, любой приём пищи обязан быть завтраком, обедом или ужином – иные варианты не предполагаются) (рис. 1).

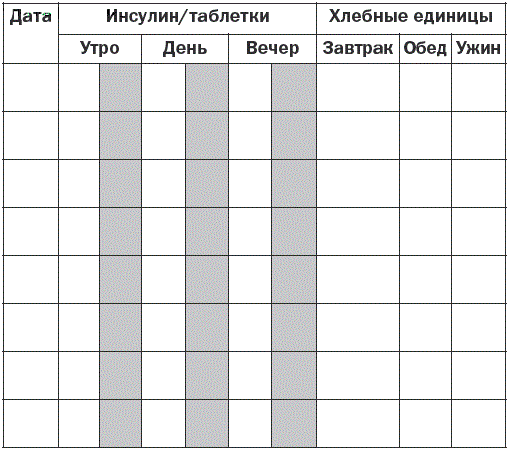


Рис. 1. Классический дневник

При этом расчёт доз обычно производится пациентами «на глаз» либо с помощью так называемых хлебных единиц (ХЕ) – довольно грубой единицы измерения количества углеводов в пище (обычно за одну хлебную единицу принимается 10-12 г углеводов). Как следствие, низкая точность получаемых результатов и «скачущий» сахар крови, приводящий к раннему появлению осложнений [1, 2].

Для технологической поддержки решения данной проблемы необходима разработка информационной системы, позволяющей хранить, анализировать и оперативно записывать информацию о состоянии здоровья пациентов, проводимой инсулиновой терапии, приёмах пищи и уровне сахара крови. Система должна включать приложения для персонального компьютера, мобильной и web-платформ, а также сервер, обеспечивающий централизованное хранение и синхронизацию данных.

1. Постановка задачи

Разработать и реализовать информационную систему, предназначенную для индивидуального мониторинга и контроля инсулинозависимого сахарного диабета первого типа. Чтобы пользователь мог работать с системой независимо от средств доступа, необходимо реализовать набор приложений для различных платформ на базе клиент-серверных технологий. Для этого следует разработать:

1. REST-сервис, который предоставляет следующие функциональные возможности:

* авторизация пользователей;
* чтение и запись дневника пользователя;
* чтение и запись баз продуктов пользователя;
* чтение и запись баз блюд пользователя.

1. Клиентские приложения:

* локальное приложение, которое будет установлено на ПК пользователя; должно использовать локальную базу данных с возможностью синхронизации с БД, хранящейся на сервере, посредством REST-сервиса;
* web-приложение, позволяющее работать с системой с помощью web-браузера;
* мобильное Android-приложение, которое будет установлено на смартфоне (планшете) пользователя; должно использовать локальную базу данных с возможностью синхронизации с БД, хранящейся на сервере, посредством REST-сервиса.

Все клиентские приложения должны предоставлять следующие функции:

* ведение дневника пользователя;
* ведение баз продуктов и блюд;
* статистический анализ дневника пользователя;
* построение адаптивной модели углеводного обмена;
* вычисление необходимой дозировки инсулина.

При реализации клиентских приложений должны быть учтены следующие особенности:

* ограничения платформы реализации;
* характеристики устройств, на которых будет установлено клиентское приложение;
* организация хранения локальных данных;
* интерфейс пользователя.

2. Анализ задачи

2.1. REST-сервис

Необходимо создать интерфейс прикладного программирования (API) для обработки HTTP-запросов клиентских приложений. Запросы можно разделить на несколько категорий, перечисленных ниже:

* Работа с учётными записями пользователей и информацией о сервере:
  + получение версии API сервера;
  + защита от неавторизованного доступа;
  + авторизация пользователей;
  + выход из системы.
* Работа с базами данных:
  + синхронизация дневника, который хранится локально у пользователя, с дневником, хранящимся в БД на сервере и связанным с учётной записью пользователя (это необходимо для повышения надёжности хранения данных пользователя, а также предоставления возможности вести дневник одновременно на нескольких устройствах);
  + синхронизация баз продуктов и блюд (это необходимо для тех же целей, что и синхронизация дневника).

2.2. Общая функциональность клиентских приложений

2.2.1. Общие требования

Дата и время в приложении должны храниться в формате UTC, однако при вводе/выводе пользователь должен видеть информацию в своём текущем часовом поясе, определяемом системой.

Все поля ввода массы должны поддерживать использование простых выражений с четырьмя арифметическими операциями.

2.2.2. Базы продуктов и блюд

Пользователь должен иметь возможность хранить, просматривать, создавать и редактировать продукты и блюда.

Каждый продукт должен содержать:

* название;
* относительное содержание белков (на 100 г);
* относительное содержание жиров (на 100 г);
* относительное содержание углеводов (на 100 г);
* относительная калорийность (на 100 г);
* метка «источник информации», возможные значения:
  + этикетка;
  + таблица.

Каждое блюдо должно содержать следующую информацию:

* название;
* вес готового блюда (опционально);
* состав – список продуктов/блюд, каждый из которых содержит следующую информацию:
  + название;
  + относительное содержание белков (на 100 г);
  + относительное содержание жиров (на 100 г);
  + относительное содержание углеводов (на 100 г);
  + относительная калорийность (на 100 г);
  + масса продукта/блюда.

Дополнительные требования к интерфейсу:

* пользователю должны быть доступны поля быстрого поиска продуктов и блюд: при вводе поисковой строки приложение должно фильтровать и отображать только те продукты (блюда), название которых включает в себя введённую поисковую строку;
* продукты и блюда должны иметь визуально различимые иконки, обеспечивающие цельность опыта взаимодействия пользователя (consistent user experience).

2.2.3. Дневник пользователя

Дневник пользователя представляет собой список записей одного из четырёх типов, перечисленных ниже. Приложение должно позволять пользователю создавать, просматривать и редактировать следующие типы записей:

* замер СК (дата, время, значение СК в ммоль/л, наименование пальца, из которого бралась кровь для анализа);
  + значение СК должно храниться с точностью до одного знака после запятой;
  + при создании новой записи поле для указания пальца должно автоматически заполняться наименованием пальца, следующего за тем, из которого последний раз бралась кровь;
  + все десять пальцев должны чередоваться по кругу: левый большой, левый указательный, левый средний, левый безымянный, левый мизинец, правый мизинец, правый безымянный, правый средний, правый указательный, правый большой – это необходимо для равномерного распределения проколов пальцев и улучшения их заживаемости;
* инъекция болюсного инсулина (дата, время и доза инсулина);
  + доза инсулина должна храниться с точностью до одного знака после запятой;
* приём пищи (дата, время, список продуктов);
  + для каждого продукта в приёме пищи должна храниться информация об относительном (на 100 г) содержании белков, жиров, углеводов и калорийности, а также массе – всё с точностью до двух знаков после запятой;
  + приём пищи должен иметь переключатель «стандартный / сокращённый постпрандиал», значение по умолчанию – «стандартный постпрандиал»;
  + продукты для приёма пищи пользователь может выбирать из выпадающего списка автодополнения;
  + в списке автодополнения должны присутствовать все продукты и блюда пользователя, отсортированные по частоте использования;
* заметка (дата, время, текст заметки).

2.2.4. Релевантная сортировка

Для повышения удобства пользователя продукты и блюда (пункты) в списке автодополнения должны быть отсортированы по убыванию релевантности. Релевантность каждого пункта определяется как

|  |  |  |
| --- | --- | --- |
|  | , | (1) |
|  | , | (2) |

где

|  |  |  |
| --- | --- | --- |
|  | – | релевантность k-го пункта, |
|  | – | количество раз, которое k-й пункт встретился в дневнике за период , |
|  | – | функция взвешивания, |
|  | – | текущее время, |
|  | – | момент времени, в который k-й пункт встретился в дневнике i-й раз, |
|  | – | определяет интервал анализа дневника (по умолчанию – один месяц). |

2.2.5. Математическая модель углеводного обмена

Для моделирования углеводного обмена используется следующее уравнение:

|  |  |  |
| --- | --- | --- |
|  | , | (3) |

где

|  |  |  |
| --- | --- | --- |
|  | – | сахар крови (СК) перед едой (input blood sugar), ммоль/л, |
|  | – | углеводный коэффициент, определяющий гипергликемический эффект 1 г углеводов, (ммоль/л)/г, |
|  | – | количество углеводов (carbohydrates) в приёме пищи, г, |
|  | – | белковый коэффициент, определяющий гипергликемический эффект 1 г белков, (ммоль/л)/г, |
|  | – | количество белков (proteins) в приёме пищи, г, |
|  | – | инсулиновый коэффициент, определяющий гипогликемический эффект 1 единицы инсулина, (ммоль/л)/ед, |
|  | – | введённая доза инсулина (insulin), единиц, |
|  | – | СК через 4-5 часов после еды (output blood sugar), ммоль/л. |

Каждый из коэффициентов является функцией времени суток:

|  |  |  |
| --- | --- | --- |
|  | , , | (4) |

где t – время суток в минутах после полуночи, .

Выделим из дневника пользователя замеры СК, инъекции инсулина и записи о приёмах пищи из интервала и сформируем из них набор данных вида (5):

|  |  |  |
| --- | --- | --- |
|  |  | (5) |

где

|  |  |  |
| --- | --- | --- |
|  | – | текущая дата, |
|  | – | период выборки данных, |
|  | – | время приёма пищи (время суток) в минутах, , |
|  | – | дата приёма пищи, |
|  | – | СК перед едой, ммоль/л, |
|  | – | количество белков в приёме пищи, г, |
|  | – | количество жиров в приёме пищи, г, |
|  | – | количество углеводов в приёме пищи, г, |
|  | – | введённая доза инсулина, единиц, |
|  | – | СК не менее чем через 4-5 часов после еды, ммоль/л. |

Для данных (5) составим переопределённую систему линейных уравнений (6):

|  |  |  |
| --- | --- | --- |
|  | , | (6) |

где

.

Придадим теперь уравнениям (6) весовые коэффициенты , зависящие от времени :

|  |  |  |
| --- | --- | --- |
|  |  | (7) |
|  |  | (8) |

где c – коэффициент адаптации, подбираемый опытным путём.

Получаем переопределённую взвешенную систему линейных уравнений:

|  |  |  |
| --- | --- | --- |
|  |  | (9) |

Определим функции (4) как решения системы (9) методом наименьших квадратов:

|  |  |  |
| --- | --- | --- |
|  |  | (10) |

Введём обозначения (11-17):

|  |  |  |
| --- | --- | --- |
|  |  | (11) |
|  |  | (12) |
|  |  | (13) |
|  |  | (14) |
|  |  | (15) |
|  |  | (16) |
|  |  | (17) |

Тогда для решения задачи (10) необходимо решить линейную систему уравнений (18):

|  |  |  |
| --- | --- | --- |
|  |  | (18) |

Для этого воспользуемся методом Крамера:

|  |  |  |
| --- | --- | --- |
|  |  | (19) |
|  |  | (20) |
|  |  | (21) |

Найдя функции (4) с помощью выражений (19-21), можно определить необходимую дозировку инсулина для любых начальных условий:

|  |  |  |
| --- | --- | --- |
|  | , | (22) |

где

|  |  |  |
| --- | --- | --- |
|  | – | необходимая доза инсулина, единиц, |
|  | – | целевой (нормальный) сахар крови, ммоль/л. |

2.2.6. Синхронизация

Все клиенты должны иметь возможность автоматически синхронизироваться с сервером с помощью REST-сервиса. Для этого все записи дневника, продукты и блюда должны являться стандартными версионированными объектами, т.е. кроме основных данных должны содержать следующие метаданные:

* ID (globally unique identifier – глобально уникальный идентификатор);
* дата и время последней модификации (timestamp);
* номер версии;
* признак удалённости объекта.

Работа с дневником, базами продуктов и блюд должна быть реализована через сервисы, имплементирующие стандартный интерфейс, имеющий следующие методы:

* получить список объектов, изменённых после указанного момента времени;
* получить объект по идентификатору;
* сохранить список объектов.

Синхронизация двух сервисов данных (DAO) должна происходить следующим образом:

1. От каждого сервиса получаются списки объектов, изменённых после последней синхронизации этих сервисов минус сутки (для компенсации возможных ошибок сохранения времени последней синхронизации).
2. Для всех объектов, присутствующих только в списке первого (второго) сервиса, должна осуществиться попытка дополнительно запросить соответствующий объект из второго (первого) сервиса, при его наличии объект добавляется в соответствующий список; сравнение идёт по ID.
3. Все объекты, присутствующие только в первом (втором) списке, сохраняются во втором (первом) сервисе.
4. Все объекты, присутствующие в обоих списках, но имеющие больший номер версии в первом (втором) списке, сохраняются во втором (первом) сервисе.

2.3. Локальное приложение

Приложение для ПК должно иметь основную функциональность, описанную в п. 2.2.

Общие технические требования к локальному приложению:

* все задачи, выполняющиеся продолжительное время (авторизация, синхронизация, сохранение и загрузка данных) должны выполняться в отдельном потоке;
* ключевые моменты в работе приложения (предупреждения, ошибки) должны логироваться (записываться) в текстовый файл.
* загрузка (сохранение) всех данных должна занимать не более одной секунды.

Дополнительные требования к отдельным компонентам локального приложения описаны ниже в этом разделе.

2.3.1. Дневник пользователя

Дополнительные требования к пользовательскому интерфейсу дневника:

* редактор для каждого типа записей должен быть представлен в виде модального окна;
* для просмотра записей пользователь должен выбрать дату в календаре; в ответ приложение должно отобразить все записи, относящиеся к выбранному дню (в часовом поясе пользователя), а также дополнительную статистику:
  + суммарное количество белков за день (в граммах) и процент от нормы;
  + суммарное количество жиров за день (в граммах) и процент от нормы;
  + суммарное количество углеводов за день (в граммах) и процент от нормы;
  + суммарная калорийность за день (в килокалориях);
  + суммарная масса всей еды за день (в граммах);
  + суммарная доза инсулина за день (в условных единицах);
  + график СК за день с цветовым выделением зон нормального и повышенного уровня СК; зоны определяются тремя значениями:
* минимальный СК до еды;
* максимальный СК до еды;
* максимальный СК после еды;
* дневные нормы белков, жиров и углеводов, а также граничные значения СК должны настраиваться из пользовательского интерфейса (см. п. 2.3.3);
* пользователю должна быть доступна панель со следующей информацией:
  + прошедшее время после последней инъекции инсулина (с точностью до минуты);
  + прошедшее время после последнего приёма пищи (с точностью до минуты);
  + название очередного пальца для замера СК;
* при выборе записи с приёмом пищи пользователь должен видеть три дополнительные панели:
  + статистика по приёму пищи:
* суммарное количество белков (в граммах);
* суммарное количество жиров (в граммах);
* суммарное количество углеводов (в граммах);
* суммарная калорийность (в килокалориях);
* суммарная масса еды (в граммах);
  + калькулятор дозы инсулина (см. п. 2.2.5):
* расчётная доза инсулина (с учётом поправки на СК перед едой – должен определяться автоматически, при его отсутствии считать СК перед едой нормальным);
* коррекция (разность между нормальным СК и СК перед едой), в зависимости от значения (по модулю) метка должна менять цвет:
  + менее 3.0 ммоль/л – серый;
  + от 3.0 до 6.0 ммоль/л – чёрный;
  + свыше 6.0 ммоль/л – тёмно-красный;
* ожидаемый СК с учётом СК перед едой и введённой дозы, если таковая найдена (в ммоль/л);
  + подбор углеводов – представлен в виде таблицы, состоящей из следующий столбцов:
* доза инсулина, в условных единицах – целое число от 0 до некоторого максимального значения, указанного в настройках (см. п. 2.3.3); при обнаружении доз инсулина, превышающих это значение, оно должно автоматически увеличиваться до максимальной введённой дозы инсулина;
* коррекция, в граммах – показывает, какое количество углеводов нужно добавить в приём пищи (убрать из него), чтобы расчётная доза инсулина совпала с дозой из первого столбца;
* итого, в граммах – сумма количества углеводов в приёме пищи и коррекции;
  + все панели должны иметь в верхнем правом углу кнопку для сворачивания / разворачивания панели;
  + в статусной панели приложения должна отображаться следующая информация:
* текущие выполняемые задачи (если есть);
* статус подключения к серверу (см. п. 2.2.6):
  + «онлайн» – подключение установлено;
  + «ошибка подключения» – подключение установить не удалось;
* статус последней синхронизации (см. п. 2.2.6):
  + «изменено» – дневник был изменён после последней синхронизации;
  + «синхронизировано» – дневник был успешно синхронизирован и с этого момента не был изменён;
  + «ошибка синхронизации» – в ходе последней синхронизации произошла ошибка;

2.3.2. Визуализация модели углеводного обмена

Пользователь должен иметь возможность просматривать суточные графики следующих величин для каждого из применяемых методов анализа, а также для средневзвешенного метода:

* углеводный коэффициент (k);
* белковый коэффициент (p);
* инсулиновый коэффициент (q);
* объединённый компенсационный коэффициент (x) – показывает количество инсулина, необходимого для усвоения 1 грамма углеводов и 0,25 грамма белков:

|  |  |  |
| --- | --- | --- |
|  | . | (23) |

2.3.3. Настройки пользователя

Пользователь должен иметь возможность изменять следующие настройки приложения:

* медицинские параметры (значения СК, максимальная доза инсулина, дневные нормы питательных веществ и пр.):
* параметры анализа;
* параметры интерфейса (включая внешний вид);
* учётные данные.

2.4. Web-приложение

Web-приложение должно иметь основную функциональность, описанную в п. 2.2. Кроме того, оно должно предоставлять пользователю возможность регистрации, авторизации в системе и выхода из неё.

2.5. Мобильное приложение

Мобильное приложение должно иметь основную функциональность, описанную в п. 2.2. Кроме того, оно должно обладать необходимым набором систем кэширования для обеспечения высокой производительности на мобильном устройстве.

3. Средства реализации

Поставленная задача реализована с использованием следующих сред разработки:

* Borland Delphi 7 (язык Delphi) – локальное приложение [4];
* Eclipse IDE (язык Java, Java Script) – мобильное приложение, web-приложение, REST-сервис [6, 7, 8, 9].

Дополнительно использовались следующие технологии:

* ADB Tools – отладка мобильного приложения;
* ADT plugin – плагин для Eclipse для работы с Android SDK;
* Android SDK – основная Android-библиотека [20];
* Apache Maven – управление зависимостями проектов [12];
* Apache Tomcat – сервер Java-приложений [11];
* Apache Wicket – пользовательский интерфейс web-приложения [13];
* Bitbucket – система отслеживания ошибок;
* DUnit – модульное тестирование (Delphi);
* Find Bugs – статический анализ кода;
* Google Code – облачный репозиторий исходного кода;
* GSON – сериализация данных;
* Jenkins Cargo Plugin – развёртывание web-приложения [10];
* Jenkins CI – сборка и тестирование web-приложения [10];
* Jersey – реализация REST-сервиса;
* JQuery – отдельные компоненты пользовательского интерфейса;
* JSP – отображение простых страниц [17, 18];
* JUnit – модульное тестирование (Java);
* MySQL – СУБД REST-сервиса [3, 14];
* MySQL Workbench – разработка архитектуры БД [5, 15, 16];
* PMD – статический анализ кода;
* SLF4J – логирование;
* Spring Security – контроль доступа к web-ресурсам;
* SQLite – СУБД мобильного приложения;
* Subversion Plugin – плагин для работы с SVN в Eclipse [21];
* SVN – система контроля версий исходного кода.

Выбор обусловлен требованиями учебной дисциплины, а также современным состоянием рынка ПО.

4. Требования к аппаратному и программному обеспечению

4.1. Требования для REST-сервиса

Требования к аппаратному обеспечению:

* ПК типа IBM PC;
* процессор с тактовой частотой не менее 2.4 ГГц;
* оперативная память не менее 2048 Мб;
* постоянная память не менее 4 Гб;
* подключение к сети Интернет со скоростью не менее 100 Мбит/с.

Требования к программному обеспечению:

* Java Runtime Environment;
* MySQL server;
* Tomcat Application Server 7.0.

4.2. Требования для локального приложения

Требования к аппаратному обеспечению:

* ПК типа IBM PC;
* процессор с тактовой частотой не менее 1 ГГц;
* оперативная память не менее 128 Мб;
* постоянная память не менее 32 Мб;
* видеокарта 800х600;
* подключение к сети Интернет со скоростью не менее 100 Кбит/с (только для синхронизации).

Требования к программному обеспечению:

* ОС MS Windows XP / Vista / 7 / 8.

4.3. Требования для web-приложения

Требования к аппаратному обеспечению:

* ПК типа IBM PC;
* процессор с тактовой частотой не менее 1 ГГц;
* оперативная память не менее 128 Мб;
* постоянная память не менее 32 Мб;
* видеокарта 800х600;
* подключение к сети Интернет со скоростью не менее 100 Кбит/с;
* клавиатура;
* мышь.

Требования к программному обеспечению: браузер Opera / Google Chrome / Mozilla Firefox / Internet Explorer.

4.4. Требования для мобильного приложения

Требования к аппаратному обеспечению:

* мобильный телефон / смартфон / планшетный компьютер с поддержкой архитектуры ARM;
* постоянная память 20 Мб;
* сенсорный экран с диагональю от 2.8”;
* подключение к сети Интернет (только для синхронизации): 100 Кбит/с.

Требования к программному обеспечению: ОС Android не ниже 3.2.

5. Интерфейс пользователя

Каждое клиентское приложение предоставляет пользователю интерфейс на двух языках: русском и английском. Язык определяется автоматически на основе текущей системной локали пользователя.

5.1. Интерфейс локального приложения

После запуска приложения пользователь видит главное окно, в котором доступны три вкладки: «Дневник», «Базы» и «Анализ» (рис. 5.1-5.3).

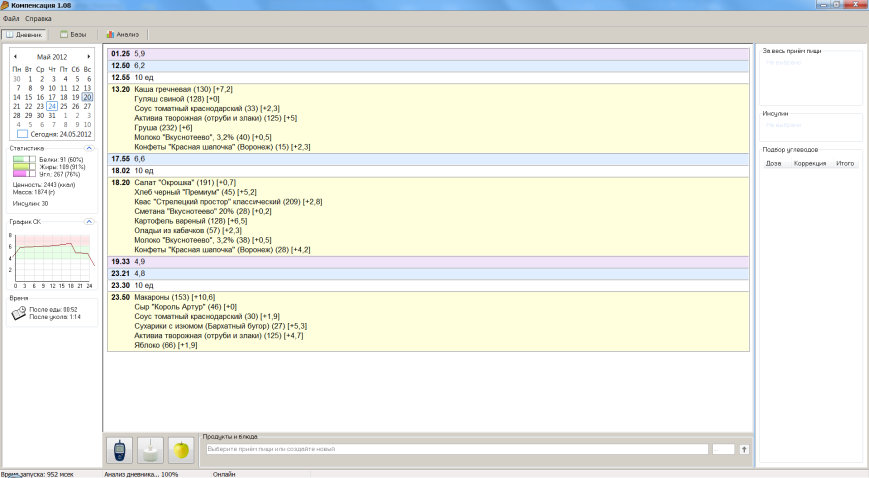


Рис. 5.1. Главное окно, закладка «Дневник»

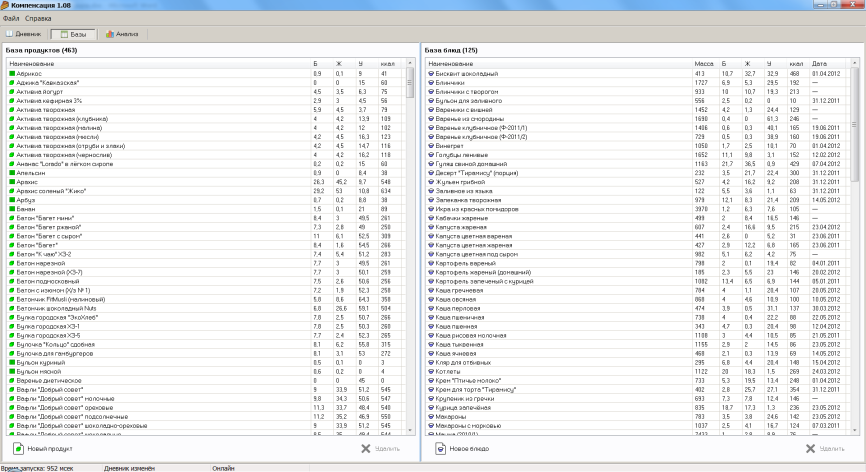


Рис. 5.2. Главное окно, закладка «Базы»

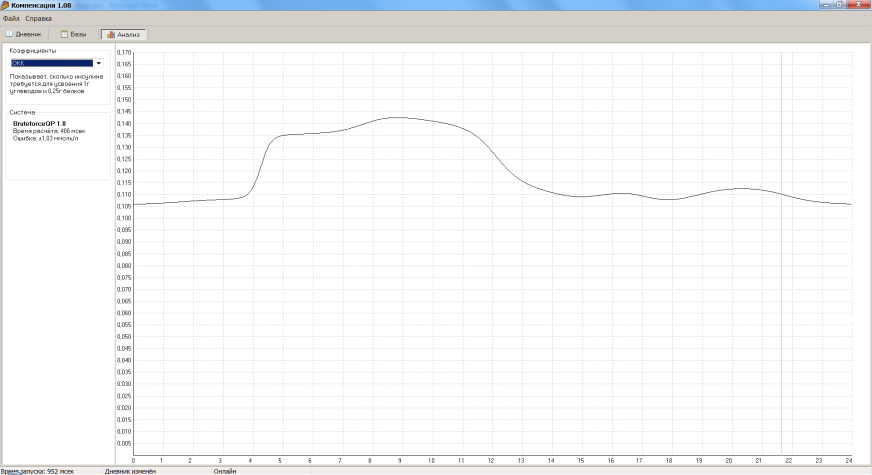


Рис. 5.3. Главное окно, закладка «Анализ»

5.1.1. Дневник

Вкладка «Дневник» имеет множество информационных панелей (блоков). Блоки сгруппированы так, что с левой стороны располагается информация обо всём текущем дне, по центру – текущая страница дневника, а справа – сведения о выбранном приёме пищи. Рассмотрим интерфейс более подробно (табл. 5.1).

Таблица 5.1. Информационные панели

|  |  |
| --- | --- |
| **Элемент интерфейса** | **Описание** |
| Левый столбец («День») | |
|  | Дневник отображается постранично, по одной странице на каждый день. Текущая дата выбирается с помощью календаря. |
|  | Под календарём расположен блок с информацией о статистике за день. Горизонтальные столбчатые диаграммы показывают долю питательных веществ от суточной нормы. |
|  | Далее расположен наглядный график изменения сахара крови за день. Зелёным цветом обозначена зона нормальных значений СК перед едой, красной – после еды (т.н. постпрандиальных значений). |
|  | Внизу расположен блок, сообщающий время после еды и инъекции – эта информация полезна для планирования постпрандиальной компенсации. |
| Центральный столбец («Страница») | |
|  | В центральной части находится сам дневник. Записи дневника расположены в хронологическом порядке. Различные типы записей (замер СК, инъекция, приём пищи, заметка) отображаются разными цветами. |
|  | Кнопки для добавления новых записей: замера СК, инъекции и приёма пищи. |
|  | Поле для ввода продуктов и блюд в дневник |
| Правый столбец («Приём пищи») | |
|  | В правой части экрана располагается информация о текущем (выбранном) приёме пищи. Из первого блока пользователь может узнать энергетический и количественный состав съеденного. |
|  | Во втором блоке отображается рассчитанная необходимая доза инсулина с учётом коррекции предыдущего значения СК. Величина коррекции показана здесь же. |
|  | Третий блок помогает пользователю подобрать необходимое количество углеводов исходя из выбранной дозы инсулина. По сравнению с классическим подходом, когда доза инсулина подбирается под еду, такой подход имеет ряд преимуществ, и в первую очередь, повышенную точность. Для большей гибкости в программе реализована возможность использовать любой из этих способов. |

При нажатии на одну из кнопок добавления записей на экране появляется соответствующее диалоговое окно (редактор), в котором пользователь может ввести необходимые данные (рис. 5.4-5.7).

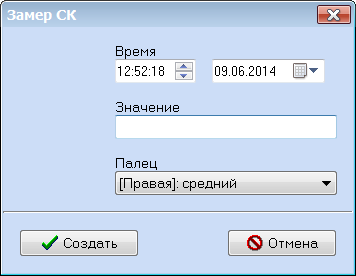


Рис. 5.4. Редактор замера СК

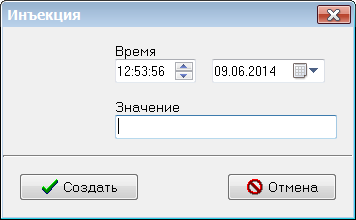


Рис. 5.5. Редактор инъекции

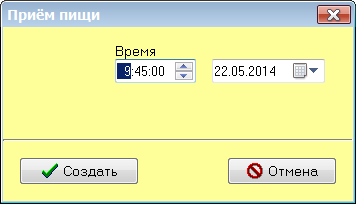


Рис. 5.6. Редактор приёма пищи

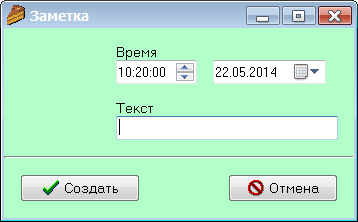


Рис. 5.7. Редактор заметки

Чтобы изменить запись, можно дважды щёлкнуть по ней либо выбрать пункт «Изменить…» из контекстного меню. Удаление записей также производится через контекстное меню (рис. 5.8).

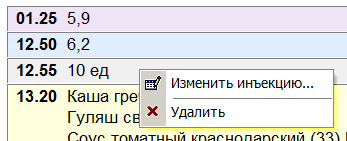


Рис. 5.8. Контекстное меню

Для добавления продуктов и блюд необходимо создать новый приём пищи (или выбрать существующий) и вводить необходимые названия и массу в соответствующее поле, подтверждая ввод нажатием клавиши Enter либо кнопкой «Добавить». Список автозавершения, отображающийся в процессе ввода, отсортирован по релевантности, что существенно ускоряет поиск необходимых продуктов (рис. 5.9). Числа справа означают содержание углеводов в 100 г продукта.

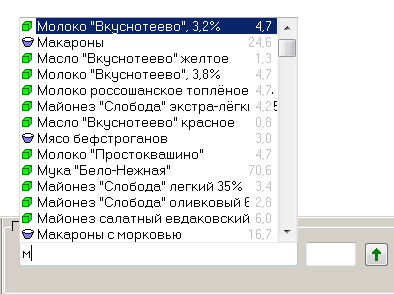


Рис. 5.9. Список автозавершения

Для повышения удобства использования программы добавлены некоторые дополнительные возможности, доступные из контекстного меню:

* Для приёма пищи: сокращённый постпрандиал – позволяет уменьшить срок, в течение которого после приёма пищи все замеры считаются простпрандиальными и, соответственно, не участвующими в анализе. Обычной практикой является установка отметки о сокращённом постпрандиале для приёмов пищи, принятых для купирования гипогликемий (состояний пониженного сахара крови).
* Для продукта из приёма пищи (рис. 5.10):
  + Изменить массу – позволяет установить новую массу продукта.
  + Прибавить/вычесть – позволяет увеличить или уменьшить массу продукта на указанную величину.
  + Подобрать до введённой дозы – вычисляет и устанавливает такую массу указанного продукта, чтобы весь приём пищи соответствовал введённой дозе. Если это невозможно, выдаётся соответствующее предупреждение.

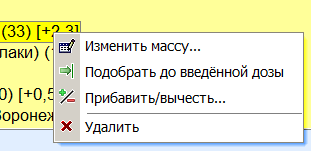


Рис. 5.10. Контекстное меню продукта

5.1.2. Базы

На данной вкладке расположена база продуктов и база блюд. В стандартной поставке имеется несколько десятков демонстрационных «табличных» продуктов и блюд, однако в целом базы предназначены для заполнения и адаптации пользователем под свои нужды.

Создать новый продукт или удалить существующий можно с помощью соответствующих кнопок в нижней части экрана. Для редактирования продукта необходимо дважды кликнуть по нему в списке либо нажать клавишу Enter (рис. 5.11).

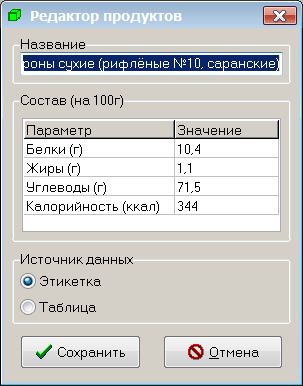


Рис. 5.11. Редактор продуктов

Добавление, удаление и модификация блюд происходит так же, как и в случае с базой продуктов, а работа с компонентами блюда – аналогично работе с приёмом пищи в дневнике (рис. 5.12).

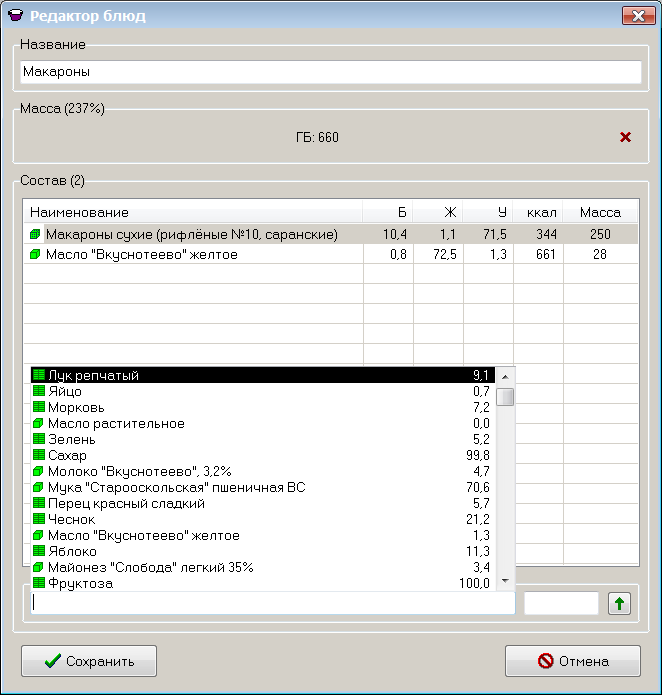


Рис. 5.12. Редактор блюд

Опция «ГБ» (готовое блюдо) необходима для тех блюд, сумма масс компонентов которых не совпадает с массой самого блюда (например, в ходе приготовления оно разварилось или ужарилось). В этом случае относительное содержание БЖУ и калорийность вычисляются с учётом изменения массы блюда.

5.1.3. Анализ

На этой вкладке отображается справочная информация о модели, построенной в ходе анализа дневника пользователя (рис. 5.13).

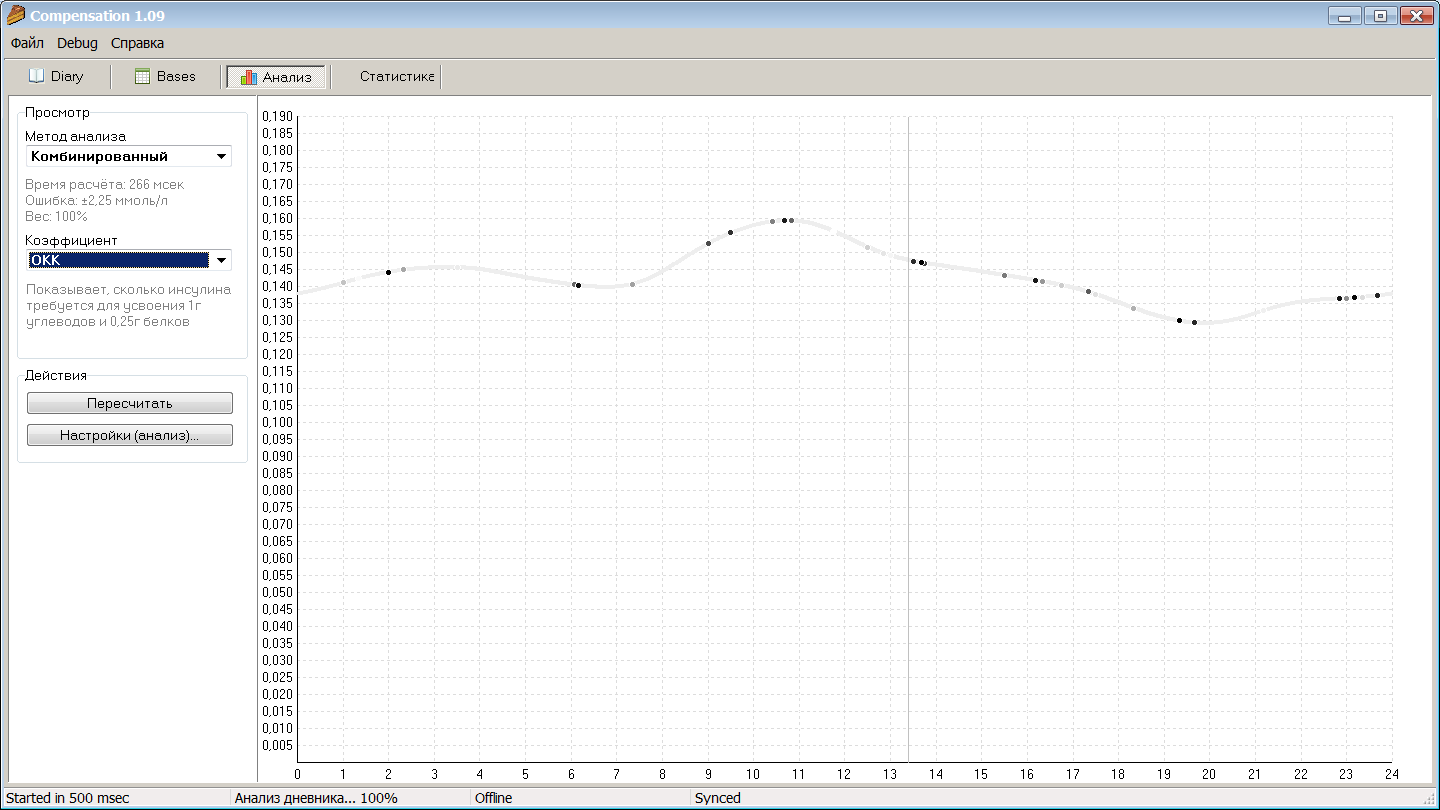


Рис. 5.13. График ОКК

На панели слева можно выбрать суточный график одного из четырёх коэффициентов:

* углеводный коэффициент – показывает, насколько увеличится сахар крови при съедении 1 грамма углеводов;
* белковый коэффициент – показывает, насколько увеличится сахар крови при съедении 1 грамма белков;
* инсулиновый коэффициент – показывает, насколько уменьшится сахар крови при введении и полной отработке 1 единицы пищевого инсулина;
* объединённый компенсационный коэффициент (ОКК) – показывает, сколько инсулина требуется для усвоения 1 грамма углеводов и 0,25 грамма белков.

Сам график отображается в центральной части вкладки и отражает изменение выбранного коэффициента в зависимости от времени суток. Текущее время отмечено вертикальной серой чертой. Кроме того, на этой вкладке можно получить информацию о названии текущего модуля анализа, времени расчёта и значение среднеквадратического отклонения построенной модели (рис. 5.14).

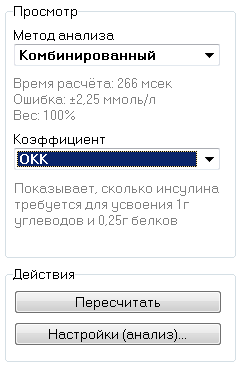


Рис. 5.14. Панель анализа

5.1.4. Настройки

Окно настроек можно открыть, выбрав в меню «Файл» пункт «Настройки…» либо воспользовавшись клавишей F10. Все настройки разделены на категории, описанные ниже.

Категория «Личные» (рис. 5.15):

* целевой сахар крови – желаемое значение сахара крови в условиях отсутствия выраженного действия внешних факторов;
* минимальный сахар крови до еды, минимальный сахар крови после еды, максимальный сахар крови после еды – границы соответствующих зон на графике СК (см. вкладку «Дневник»);
* наибольшая доза в подборе – наибольшая доза инсулина в панели «Подбор углеводов» (см. вкладку «Дневник»).

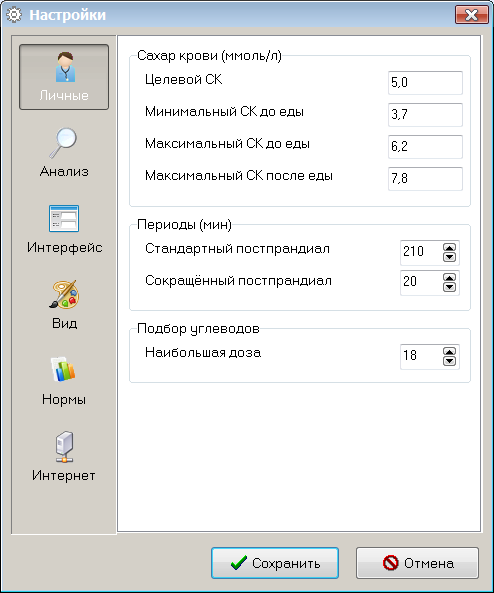


Рис. 5.15. Настройки (личные)

Категория «Анализ» (рис. 5.16):

* период анализа (дней) – при анализе дневника учитываются только записи, сделанные не позже указанного срока;
* стандартный постпрандиал (мин), сокращённый постпрандиал (мин) – продолжительность постпрандиального периода при анализе;
* ускорение адаптации – специальный параметр анализа;
* компрессия – специальный параметр анализа;
* обновление коэффициентов – определяет, когда необходимо обновлять модель.

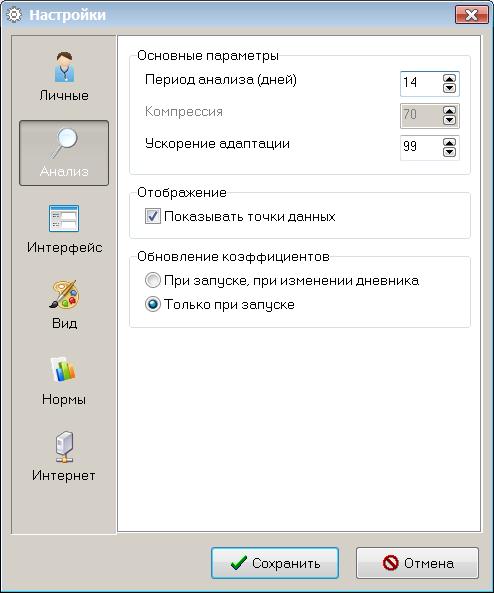


Рис. 5.16. Настройки (анализ)

Категория «Интерфейс» (рис. 5.17):

* анализ частоты использования – срок, в течение которого анализируется частота использования продуктов и блюд; применяется в процессе расчёта релевантности и построения списка автозавершения;
* интервал автосохранения – частота автоматического сохранения базы данных;
* информация об углеводности – необходимо ли отображать в списке автозавершения относительную углеводность продуктов;
* проверять обновления – необходимо ли проверять обновления программы при старте.

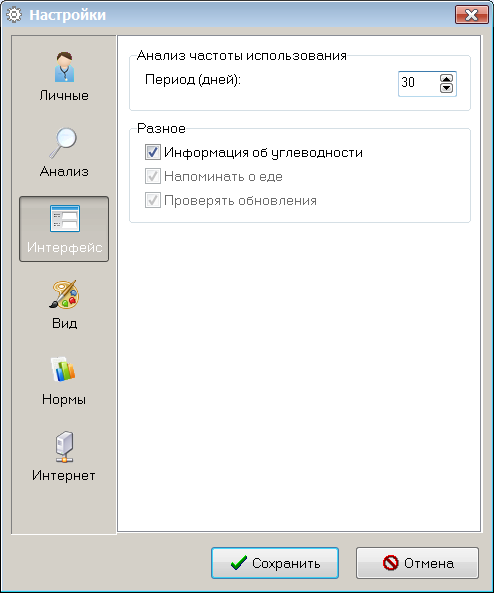


Рис. 5.17. Настройки (интерфейс)

Категория «Вид» (рис. 5.18):

* шрифт – настройки шрифта для дневника;
* цветовая схема – настройка цветовой схемы дневника;
* затенять при отображении редактора – использовать ли эффект «тень» при отображении редакторов;
* цветные редакторы – использовать ли цвета для окон редакторов из цветовой схемы дневника;
* двойная буферизация – использовать ли двойную буферизацию (увеличивает нагрузку на ЦП и улучшает внешний вид);
* анимированные панели – использовать ли анимацию при свёртывании и развёртывании информационных панелей.

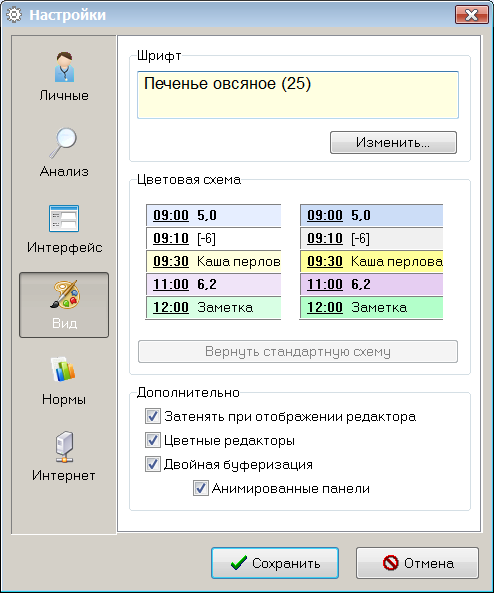


Рис. 5.18. Настройки (вид)

Категория «Нормы» (рис. 5.19):

* балансировать питание – данная опция позволяет включить балансировку питания по БЖУ и отображение суточных норм на вкладке «Дневник»;
* проценты – позволяет настроить режим отображения текущего прогресса относительно суточной нормы.

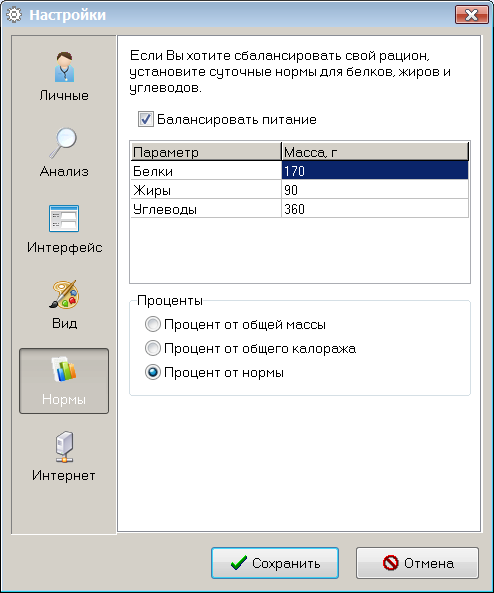


Рис. 5.19. Настройки (нормы)

Категория «Интернет» (рис. 5.20):

* адрес сервера – URL REST-сервиса для синхронизации данных;
* логин (e-mail) –логин пользователя в системе;
* пароль – пароль пользователя в системе;
* автоматически синхронизировать – определяет, включена ли синхронизация данных с REST-сервисом;
* интервал – интервал автоматической синхронизации данных.

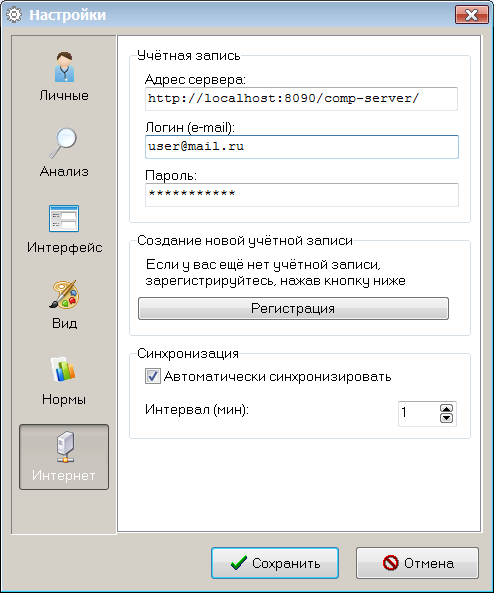


Рис. 5.20. Настройки (Интернет)

5.1.5. Экспорт

В программе также предусмотрена возможность экспортирования данных дневника в текстовый файл. Для этого пользователь должен выбрать в меню «Файл» пункт «Экспорт» либо воспользоваться сочетанием клавиш «Ctrl+E» (рис. 5.21).

В появившемся окне доступны следующие опции:

* выбор период экспорта;
* выбор данных, которые необходимо экспортировать;
* добавление разрывов (пустой строки) при разнице между двумя событиями больше трёх часов.

После определения настроек экспорта пользователь может сохранить данные в формате RTF или TXT либо просто скопировать текст из окна вывода.

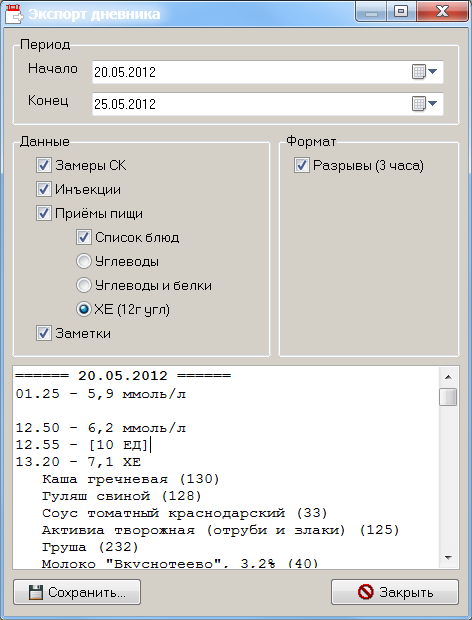


Рис. 5.21. Экспорт дневника

5.2. Интерфейс web-приложения

При открытии сайта пользователь видит страницу с кратким описанием проекта и возможностью авторизации (рис. 5.22).

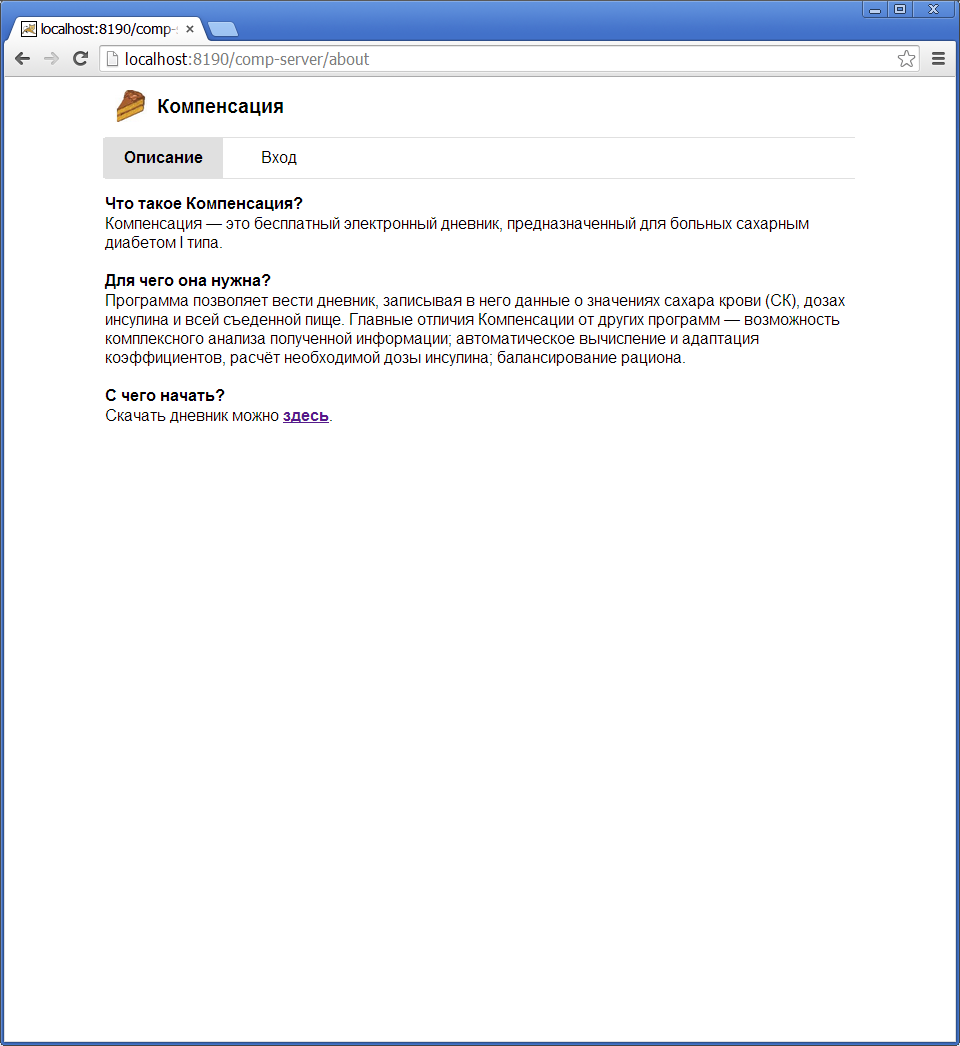


Рис. 5.22. Главная страница

После успешной авторизации пользователь переходит на страницу просмотра дневника (рис. 5.23), с помощью которой он может просматривать и редактировать свой дневник за любой день, выбрав необходимую дату из выпадающего списка.

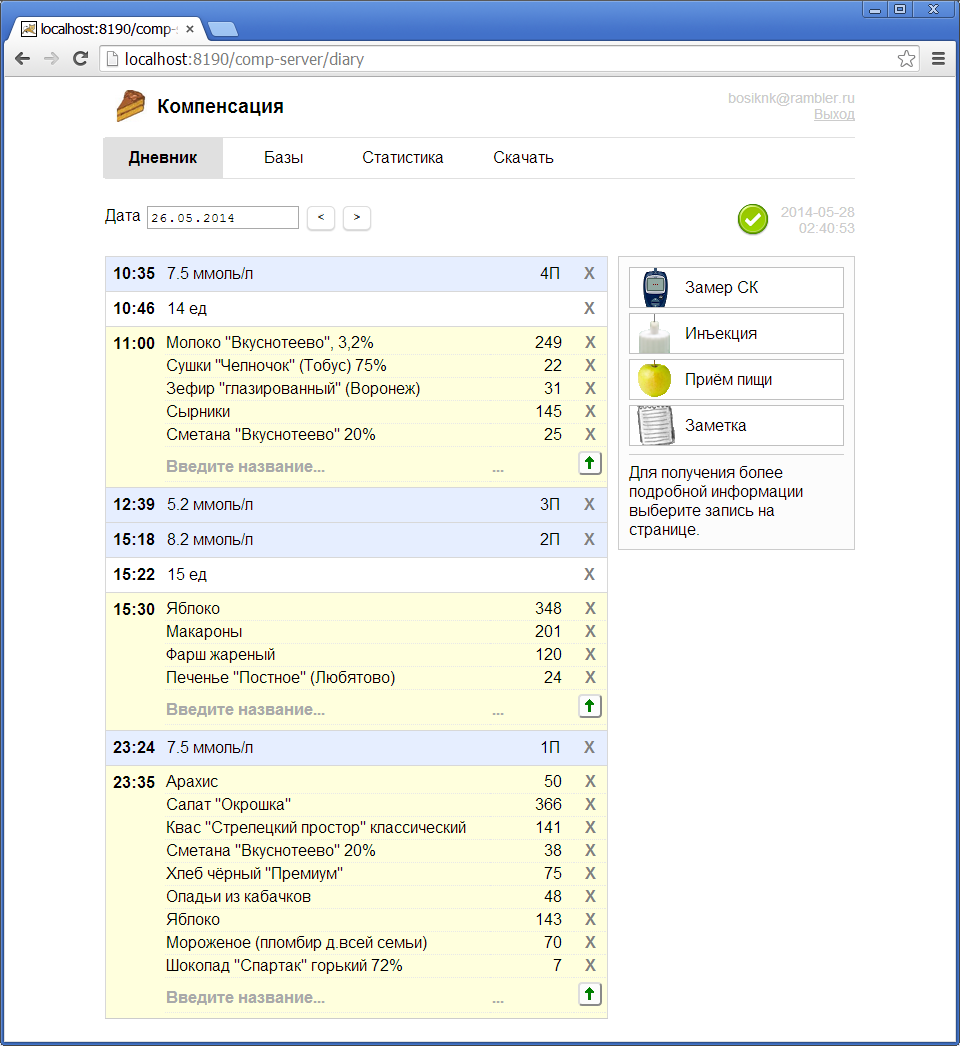


Рис. 5.23. Дневник пользователя

Выбрав приём пищи, пользователь видит окно с указанием дозировки инсулина, коррекцией углеводов и другой справочной информацией (рис. 5.24).

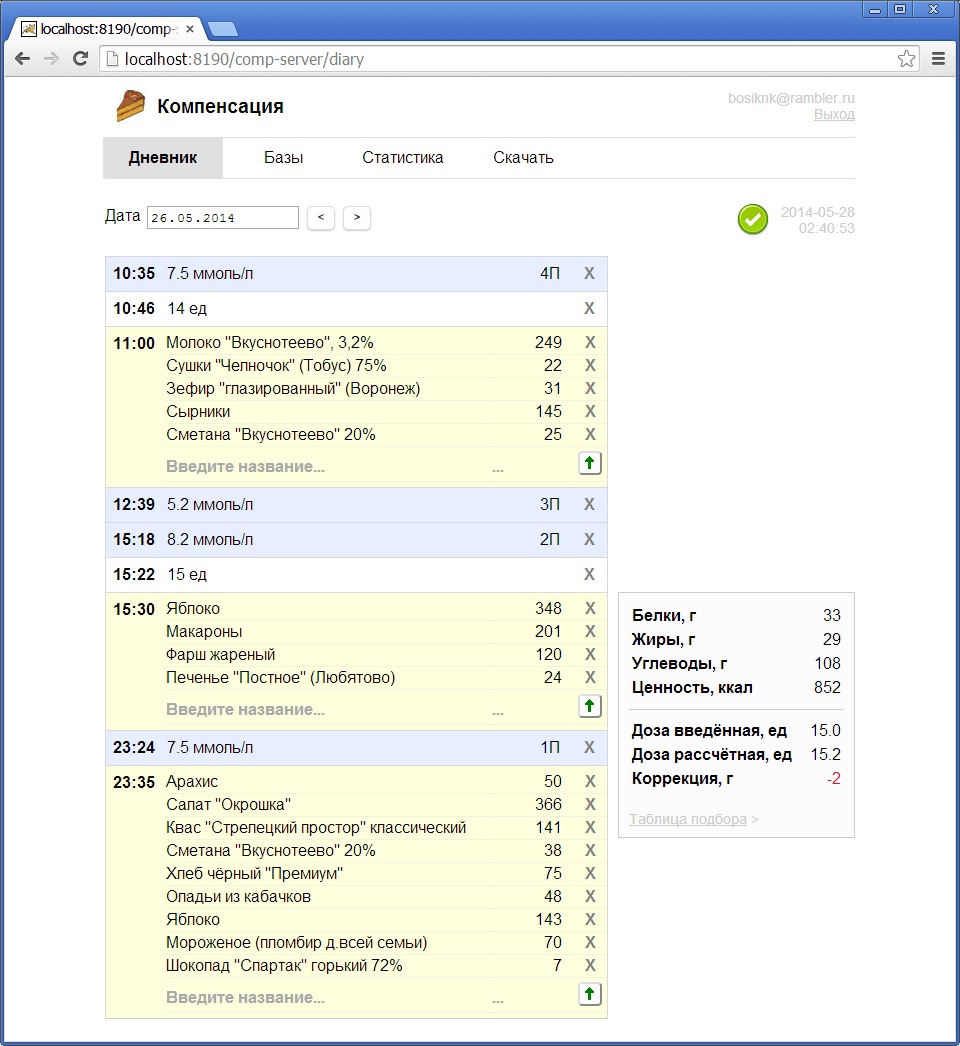


Рис. 5.24. Справочное окно с дозировкой инсулина

Перейдя на страницу «Базы», пользователь может просматривать и редактировать продукты и блюда (рис. 5.25).

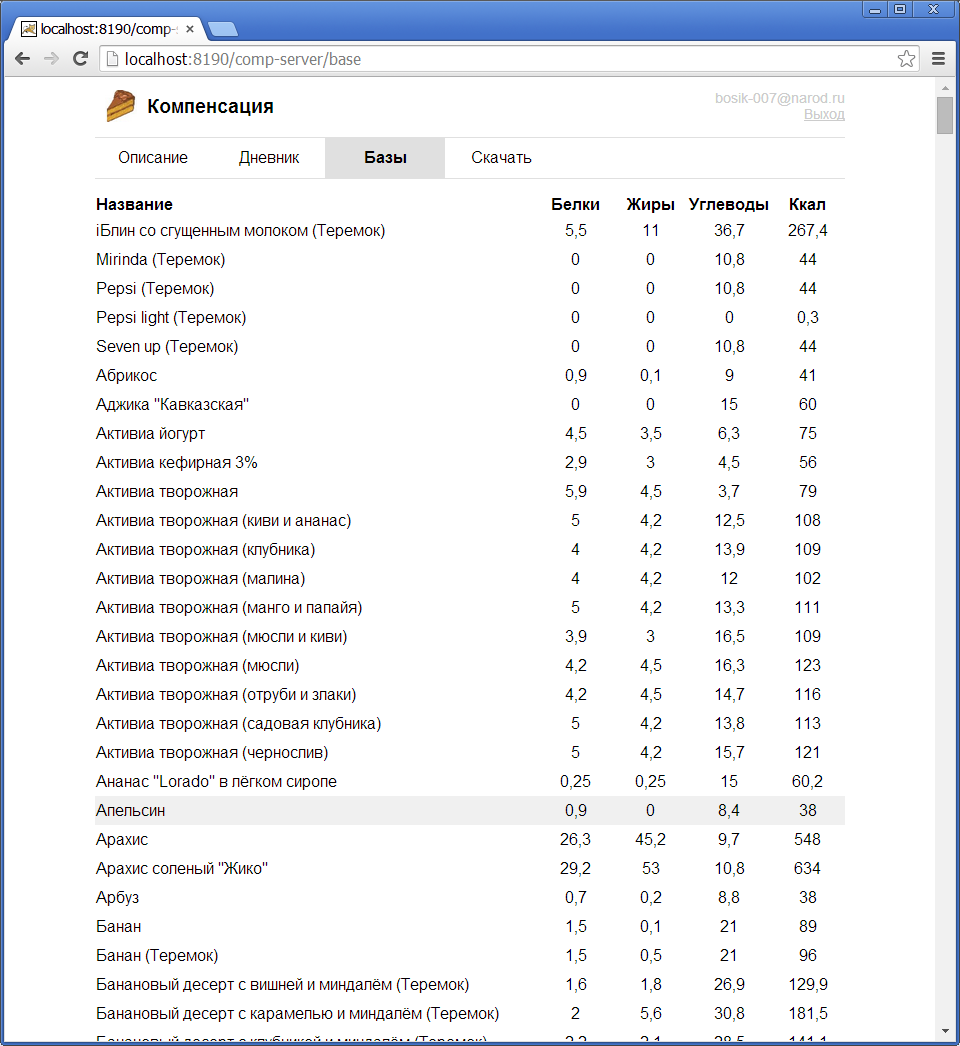


Рис. 5.25. База продуктов и блюд

На странице «Скачать» пользователь может загрузить локальное приложение, а также мобильное приложение для платформы Android (рис. 5.26).

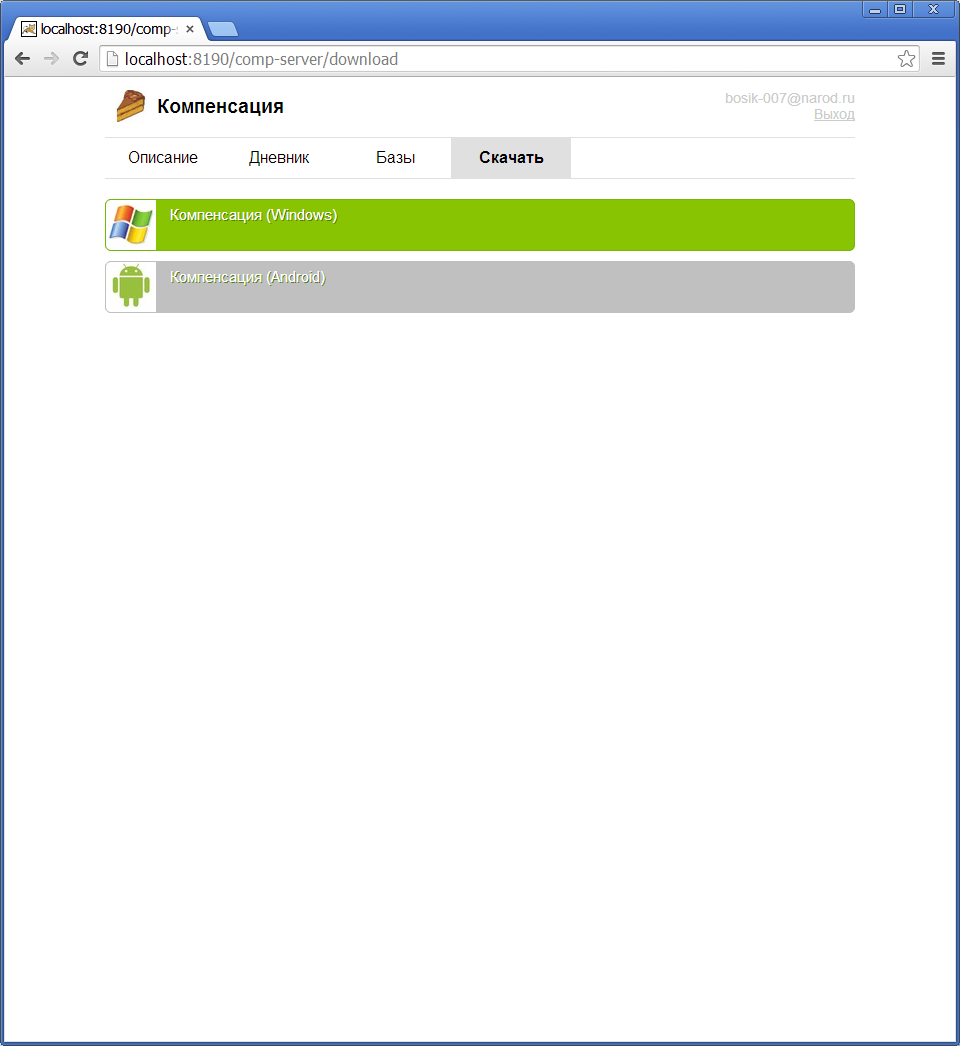


Рис. 5.26. Страница загрузок

5.3. Интерфейс мобильного приложения

Главное меню приложения представлено на рис. 5.27. Из него пользователь может перейти в активности «Дневник», «Базы» и «Настройки», а также запустить синхронизацию данных.

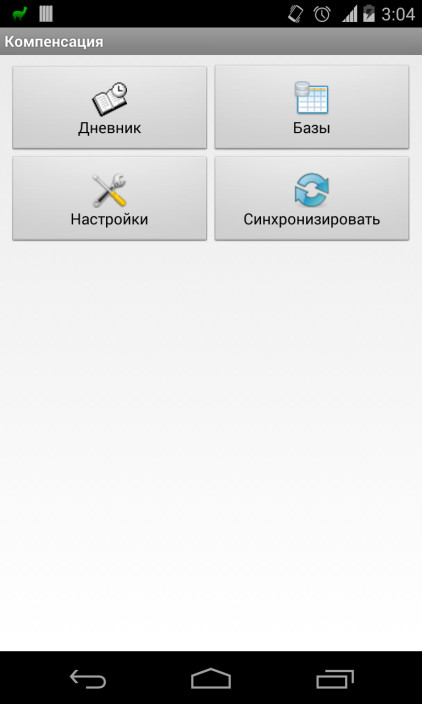


Рис. 5.27. Активность «Главное меню»

5.3.1. Дневник пользователя

При запуске приложения пользователь сразу попадает в активность «Дневник» (рис. 5.28) – это позволяет быстрее начинать вводить и просматривать данные.

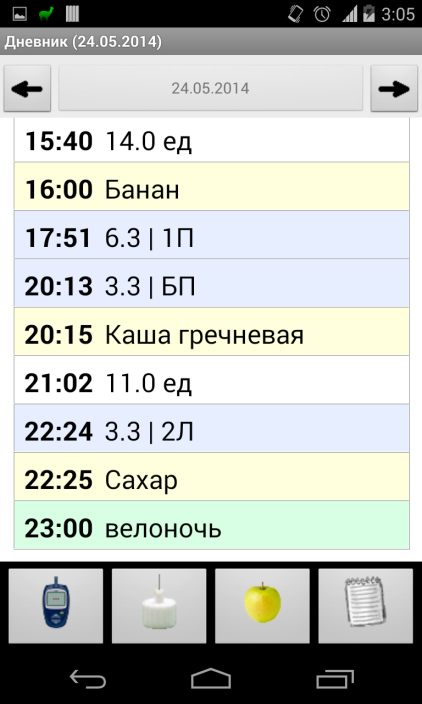


Рис. 5.28. Активность «Дневник»

Там пользователь видит страницу дневника с данными за последние сутки, при необходимости пользователь может выбрать необходимую дату с помощью стрелок. В нижней части активности доступны кнопки для создания новых записей: замеров сахара крови, инъекций инсулина, приёмов пищи и заметок. Любую созданную запись дневника пользователь может просмотреть и отредактировать, нажав на неё.

При создании / редактировании записи о замере сахара крови пользователь находится в активности «Замер сахара крови» (рис. 5.29).

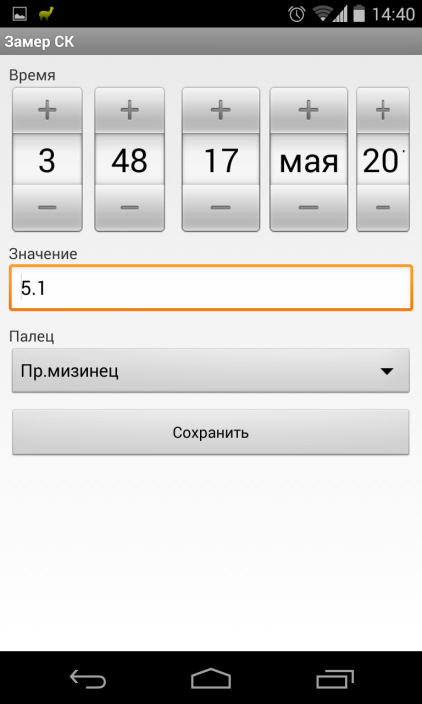


Рис. 5.29. Активность «Замер сахара крови»

Здесь он может указать дату и время замера, значение сахара крови в ммоль/л, а также палец, из которого бралась кровь – по умолчанию в этом поле подставляется очередной палец, следующий за тем, из которого последний раз бралась кровь. Для сохранения записи пользователь должен нажать кнопку «Сохранить».

При создании / редактировании записи об инъекции болюсного инсулина пользователь находится в активности «Инсулин» (рис. 5.30).

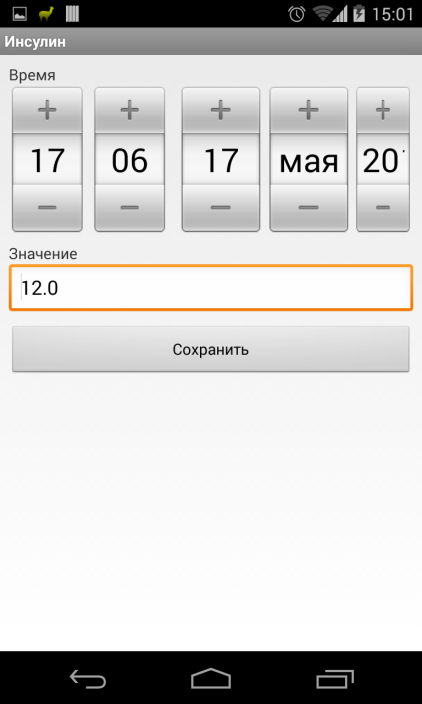


Рис. 5.30. Активность «Инсулин»

Здесь он может указать дату и время инъекции, а также введённую дозу. Для сохранения записи пользователь должен нажать кнопку «Сохранить».

При создании / редактировании приёма пищи пользователь находится в активности «Приём пищи» (рис. 5.31).

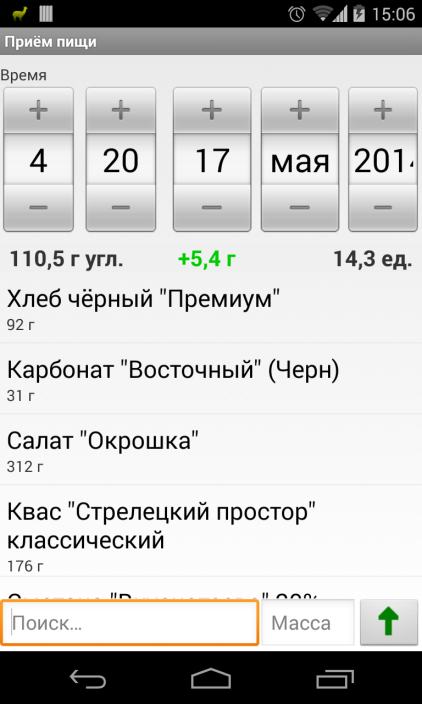


Рис. 5.31. Активность «Приём пищи»

Здесь он может указать дату и время записи, а также отредактировать список продуктов и блюд в приёме пищи. На информационной панели отображается информация о суммарном количестве углеводов в приёме пищи, коррекции углеводов до введённой дозы инсулина и расчётной дозе инсулина. Данные о сахаре крови перед приёмом пищи и введённой дозе инсулина определяются автоматически.

Поле с коррекцией углеводов удобно использовать для подбора еды под введённую дозу инсулина, поскольку на 1 единицу инсулина обычно приходится 5-10 г углеводов, такая техника позволяет существенно повысить точность компенсации. Для улучшения восприятия поле коррекции меняет цвет: оно становится зелёным, если коррекция положительна (т.е. необходимо ещё что-то съесть), и красным, если коррекция отрицательна (т.е. пользователь съел больше, чем необходимо).

Найденные результаты отображаются пользователю в списке автодополнения отсортированными по релевантности относительно дневника, что позволяет легко находить нужные продукты и блюда даже при использовании больших баз (порядка 1-2 тысяч записей). Для облегчения восприятия продукты визуально отличаются от блюд иконками в начале каждого пункта (рис. 5.32).

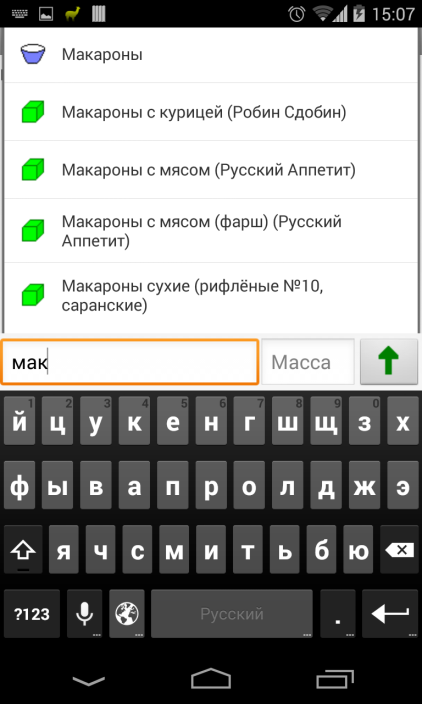


Рис. 5.32. Поиск продуктов и блюд

После выбора нужного продукта пользователь должен ввести положительную массу и нажать на кнопку с зелёной стрелкой. Поле ввода массы допускает использование простых выражений с четырьмя арифметическими действиями. При наличии ошибок пользователь получает всплывающую подсказку. Запись о приёме пищи сохраняется автоматически при выходе из активности.

При создании / редактировании заметок пользователь находится в активности «Заметка» (рис. 5.33).

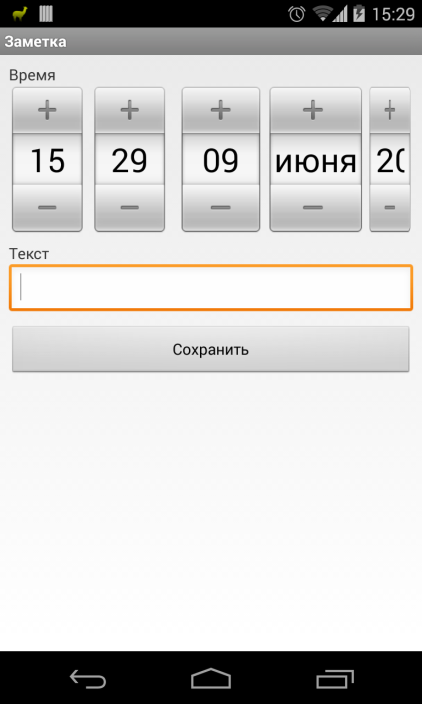


Рис. 5.33. Активность «Заметка»

Здесь он может указать дату, время и текст заметки. Для сохранения записи пользователь должен нажать кнопку «Сохранить».

5.3.2. Базы продуктов и блюд

Нажав в главном меню кнопку «Базы», пользователь попадает к активность «Базы продуктов и блюд» (рис. 5.34).

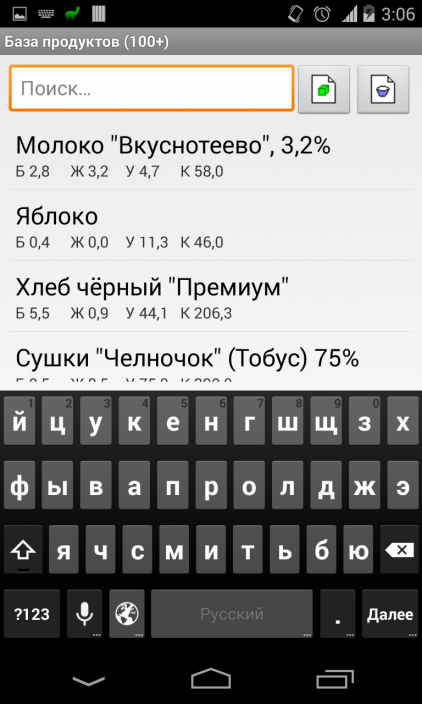


Рис. 5.34. Активность «Базы продуктов и блюд»

Там пользователь может быстро находить необходимые продукты и блюда, начиная вводить их название в поле ввода. Результаты поиска сортируются по релевантности. Количество отображаемых записей ограничено 100. В заголовке активности отображается количество найденных записей. При превышении лимита в заголовке пользователь видит пометку «100+». Под каждым пунктом имеется дополнительная информация об относительном содержании белков, жиров, углеводов и калорийности.

Для редактирования любого найденного продукта / блюда пользователь должен нажать на него. Для создания новых продуктов (блюд) пользователь должен нажать на одну из кнопок в правой части верхней панели.

В редакторе продукта пользователь может ввести всю необходимую информацию о продукте (рис. 5.35): название и содержание в 100 граммах продукта белков, жиров и углеводов, а также калорийности.

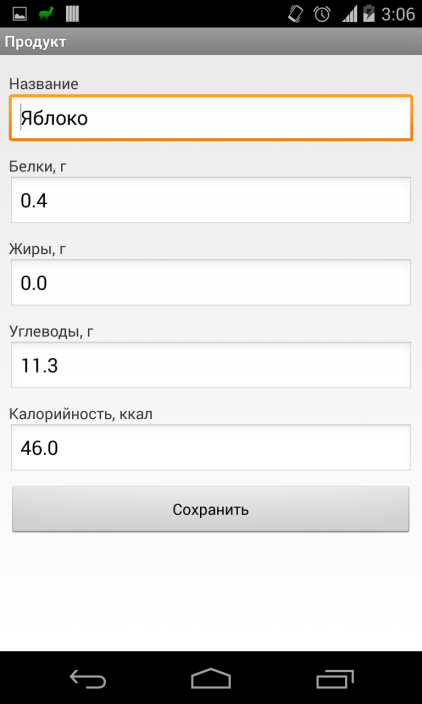


Рис. 5.35. Редактор продуктов

Для сохранения изменений пользователь должен нажать кнопку «Сохранить».

В редакторе блюд пользователь может ввести название блюда, состав и массу готового блюда (опционально) (рис. 5.36).

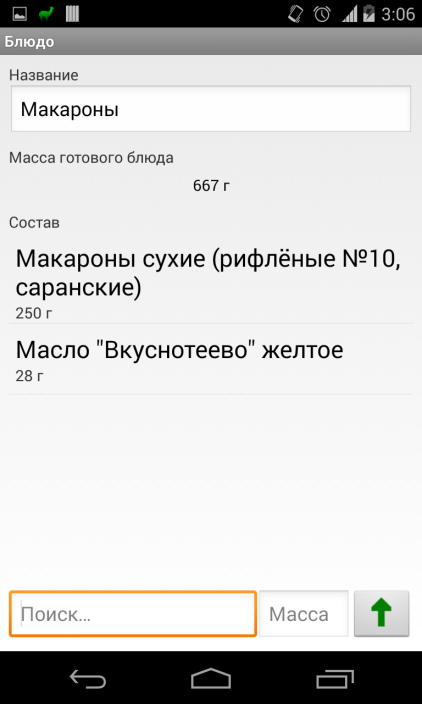


Рис. 5.36. Редактор блюд

Для редактирования состава блюда используется тот же компонент, что и для редактирования приёма пищи. Пункты в списке автодополнения сортируются по релевантности относительно базы блюд.

6. Реализация

6.1. Структура информационной системы

Общая структура информационной системы представлена на рис. 6.1. Все автономные клиентские приложения (локальное и мобильное) работают с собственными базами данных посредством функциональных сервисов, представленных набором Java-классов (для локального приложения – классов Delphi). Кроме того, функциональные сервисы web-приложения доступны извне по сети Интернет через REST-сервис, что позволяет синхронизировать БД всех клиентов между собой.

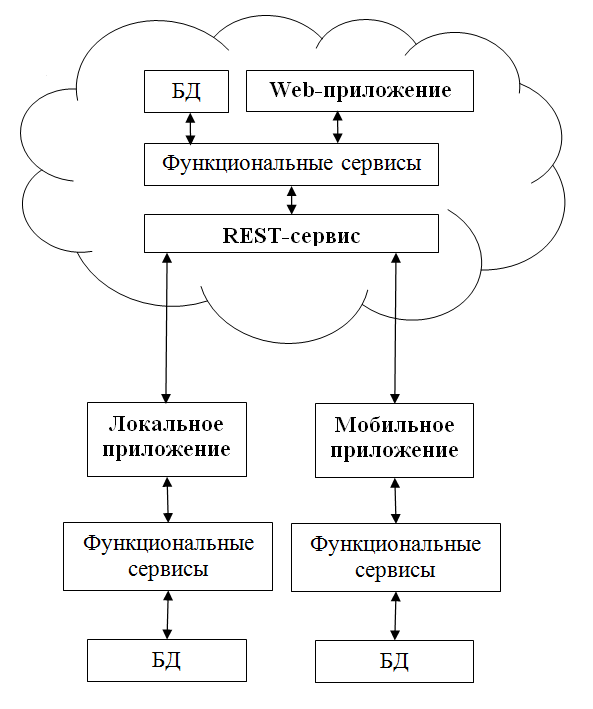


Рис. 6.1. Общая структура информационной системы

6.2. Структура баз данных

6.2.1. Структура базы данных web-приложения

База данных web-приложения отличается от БД автономных клиентов, так как кроме основной информации она должна хранить сведения о пользователях, которым принадлежит эта информация, а также об их учётных записях (рис. 6.2). Описание полей всех таблиц дано в приложении 1.

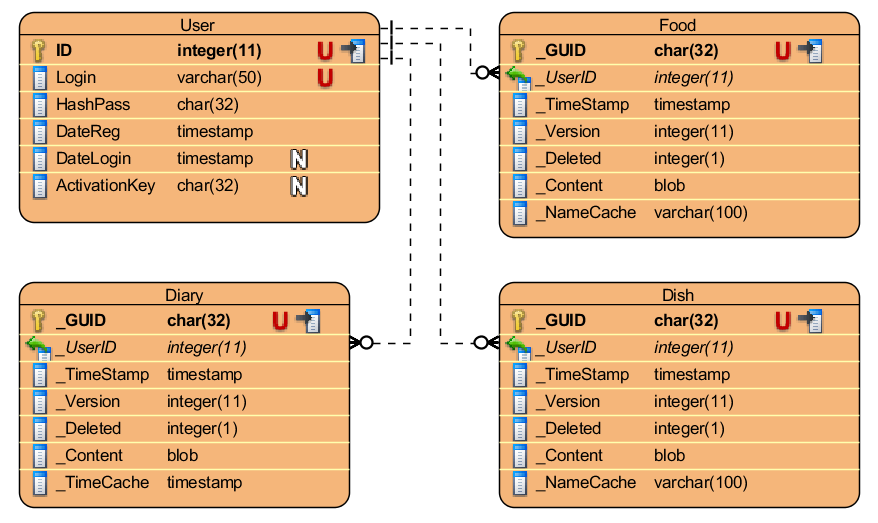


Рис. 6.2. ER-диаграмма БД web-приложения (физическая модель)

6.2.2.Структура баз данных автономных клиентских приложений

Автономные клиентские приложения имеют структуру БД, представленную на рис. 6.3. Поля таблиц имеют тот же смысл, что и в таблицах базы данных web-приложения.

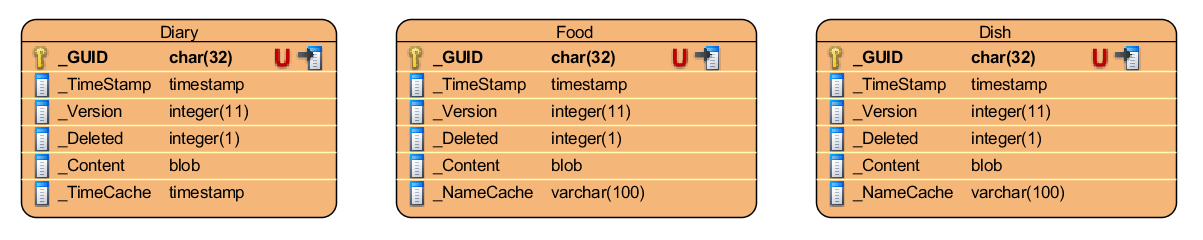


Рис. 6.3. ER-диаграмма БД клиентских приложений (физическая модель)

6.3. Форматы данных

Для облегчения взаимодействия все части системы обмениваются данными (бизнес-объектами), сериализованными с использованием унифицированного JSON-формата. Подробное описание использованного формата дано в приложении 2.

6.4. Реализация REST-сервиса

Для решения поставленной задачи был спроектирован и реализован REST API. В целях унификации обработки ответов сервиса была разработана и реализована следующая стандартная JSON-форма ответа:

{"code":<code>, "resp":<response>}

Здесь <code> – целое число, означающее код ответа, <response> – тело ответа, которое может быть представлено строкой или JSON-объектом. Кроме того, при необходимости ответ может содержать дополнительные поля.

При передаче даты / времени используется часовой пояс UTC и стандартный формат:

гггг-мм-дд чч:мм:сс

Для большей наглядности все методы разбиты на категории. URL методов указаны относительно домашнего адреса web-приложения, дополненного суффиксом /api.

6.4.1. Методы категории «Авторизация»

Метод Auth.login

Описание: позволяет пользователю авторизоваться в системе.

URL: /auth/login/

Тип: POST

Параметры:

* login – логин пользователя,
* pass – пароль пользователя,
* api – версия API клиента.

Результаты:

* {"code":0, "resp":"...", ...} – метод выполнен успешно;
* {"code":4050, "resp":"...", ...} – авторизация прошла успешно; версия API клиента устарела, но ещё поддерживается;
* {"code":4051, "resp":"...", ...} – ошибка: версия API клиента устарела и больше не поддерживается;
* {"code":4010, "resp":"...", ...} – ошибка: неверный логин или пароль;
* {"code":500, "resp":"...", ...} – внутренняя ошибка сервера.

Метод Auth.logout

Описание: позволяет пользователю выйти из системы.

URL: /auth/logout/

Тип: GET

Параметры: отсутствуют.

Результаты:

* {"code":0, "resp":"...", ...} – метод выполнен успешно;
* {"code":500, "resp":"...", ...} – внутренняя ошибка сервера.

Метод System.info

Описание: позволяет получить информацию о текущей и поддерживаемой версиях API сервиса.

URL: /info/

Тип: GET

Параметры: отсутствуют.

Результаты:

* {"code":0, "resp":{"current":<номер\_текущей\_версии>, "support":<номер\_поддерживаемой\_версии>}, ...} – метод выполнен успешно;
* {"code":500, "resp":"...", ...} – внутренняя ошибка сервера.

6.4.2. Методы категории «Дневник»

Метод Diary.findByPeriod

Описание: получение записей дневника для указанного интервала времени.

URL: /diary/period/

Тип: GET

Параметры:

* start\_time – время начала выборки (UTC) в стандартном формате;
* end\_time – время конца выборки (UTC) в стандартном формате;
* show\_rem – флаг, определяющий, требуется ли включать в выборку удалённые записи (возможные значения: 0 – не включать, 1 – включать).

Результаты:

* {"code":0, "resp":[(item), ...} – метод выполнен успешно, записи отсортированы по возрастанию времени;
* {"code":4011, "resp":"...", ...} – для выполнения действия необходима авторизация;
* {"code":500, "resp":"...", ...} – внутренняя ошибка сервера.

Метод Diary.findChanged

Описание: получение всех записей дневника (как удалённых, так и нет), изменённых после указанного времени.

URL: /diary/changes/

Тип: GET

Параметры:

* since – время (UTC) в стандартном формате.

Результаты:

* {"code":0, "resp":[(item), ...} – метод выполнен успешно, записи отсортированы по возрастанию времени;
* {"code":4011, "resp":"...", ...} – для выполнения действия необходима авторизация;
* {"code":500, "resp":"...", ...} – внутренняя ошибка сервера.

Метод Diary.findById

Описание: получение записи с указанным ID.

URL: /diary/guid/<guid>

Тип: GET

Параметры: отсутствуют, идентификатор передаётся как часть адреса.

Результаты:

* {"code":0, "resp":(item), ...} – метод выполнен успешно, запись найдена;
* {"code":404, "resp":"...", ...} – запись не найдена;
* {"code":4011, "resp":"...", ...} – для выполнения действия необходима авторизация;
* {"code":500, "resp":"...", ...} – внутренняя ошибка сервера.

Метод Diary.save

Описание: создание / сохранение одной или нескольких записей дневника.

URL: /diary/

Тип: PUT

Параметры:

* items – массив записей дневника в формате JSON.

Результаты:

* {"code":0, "resp":"...", } – метод выполнен успешно;
* {"code":4011, "resp":"...", ...} – для выполнения действия необходима авторизация;
* {"code":500, "resp":"...", ...} – внутренняя ошибка сервера.

6.4.3. Методы категории «База продуктов»

Метод Food.findAll

Описание: получение всех продуктов.

URL: /food/all/

Тип: GET

Параметры:

* show\_rem – флаг, определяющий, требуется ли включать в выборку удалённые записи (возможные значения: 0 – не включать, 1 – включать).

Результаты:

* {"code":0, "resp":[(item), ...} – метод выполнен успешно, записи отсортированы по алфавиту;
* {"code":4011, "resp":"...", ...} – для выполнения действия необходима авторизация;
* {"code":500, "resp":"...", ...} – внутренняя ошибка сервера.

Метод Food.findByName

Описание: поиск продуктов по названию, включаются только неудалённые записи.

URL: /food/search/

Тип: GET

Параметры:

* q – подстрока для поиска.

Результаты:

* {"code":0, "resp":[(item), ...} – метод выполнен успешно, записи отсортированы по алфавиту;
* {"code":4011, "resp":"...", ...} – для выполнения действия необходима авторизация;
* {"code":500, "resp":"...", ...} – внутренняя ошибка сервера.

Метод Food.findChanged

Описание: получение всех продуктов (как удалённых, так и нет), изменённых после указанного времени.

URL: /food/changes/

Тип: GET

Параметры:

* since – время (UTC) в стандартном формате.

Результаты:

* {"code":0, "resp":[(item), ...} – метод выполнен успешно, записи отсортированы по алфавиту;
* {"code":4011, "resp":"...", ...} – для выполнения действия необходима авторизация;
* {"code":500, "resp":"...", ...} – внутренняя ошибка сервера.

Метод Food.findById

Описание: получение продукта с указанным ID.

URL: /food/guid/<guid>

Тип: GET

Параметры: отсутствуют, идентификатор передаётся как часть адреса.

Результаты:

* {"code":0, "resp":(item), ...} – метод выполнен успешно, запись найдена;
* {"code":404, "resp":"...", ...} – запись не найдена;
* {"code":4011, "resp":"...", ...} – для выполнения действия необходима авторизация;
* {"code":500, "resp":"...", ...} – внутренняя ошибка сервера.

Метод Food.save

Описание: создание / сохранение одного или нескольких продуктов.

URL: /food/

Тип: PUT

Параметры:

* items – массив продуктов в формате JSON.

Результаты:

* {"code":0, "resp":"...", } – метод выполнен успешно;
* {"code":4011, "resp":"...", ...} – для выполнения действия необходима авторизация;
* {"code":500, "resp":"...", ...} – внутренняя ошибка сервера.

6.4.4. Методы категории «База блюд»

Метод Dish.findAll

Описание: получение всех блюд.

URL: /dish/all/

Тип: GET

Параметры:

* show\_rem – флаг, определяющий, требуется ли включать в выборку удалённые записи (возможные значения: 0 – не включать, 1 – включать).

Результаты:

* {"code":0, "resp":[(item), ...} – метод выполнен успешно, записи отсортированы по алфавиту;
* {"code":4011, "resp":"...", ...} – для выполнения действия необходима авторизация;
* {"code":500, "resp":"...", ...} – внутренняя ошибка сервера.

Метод Dish.findByName

Описание: поиск блюд по названию, включаются только неудалённые записи.

URL: /dish/search/

Тип: GET

Параметры:

* q – подстрока для поиска.

Результаты:

* {"code":0, "resp":[(item), ...} – метод выполнен успешно, записи отсортированы по алфавиту;
* {"code":4011, "resp":"...", ...} – для выполнения действия необходима авторизация;
* {"code":500, "resp":"...", ...} – внутренняя ошибка сервера.

Метод Dish.findChanged

Описание: получение всех блюд (как удалённых, так и нет), изменённых после указанного времени.

URL: /dish/changes/

Тип: GET

Параметры:

* since – время (UTC) в стандартном формате.

Результаты:

* {"code":0, "resp":[(item), ...} – метод выполнен успешно, записи отсортированы по алфавиту;
* {"code":4011, "resp":"...", ...} – для выполнения действия необходима авторизация;
* {"code":500, "resp":"...", ...} – внутренняя ошибка сервера.

Метод Dish.findById

Описание: получение блюда с указанным ID.

URL: /dish/guid/<guid>

Тип: GET

Параметры: отсутствуют, идентификатор передаётся как часть адреса.

Результаты:

* {"code":0, "resp":(item), ...} – метод выполнен успешно, запись найдена;
* {"code":404, "resp":"...", ...} – запись не найдена;
* {"code":4011, "resp":"...", ...} – для выполнения действия необходима авторизация;
* {"code":500, "resp":"...", ...} – внутренняя ошибка сервера.

Метод Dish.save

Описание: создание / сохранение одного или нескольких блюд.

URL: /dish/

Тип: PUT

Параметры:

* items – массив блюд в формате JSON.

Результаты:

* {"code":0, "resp":"...", } – метод выполнен успешно;
* {"code":4011, "resp":"...", ...} – для выполнения действия необходима авторизация;
* {"code":500, "resp":"...", ...} – внутренняя ошибка сервера.

6.5. Реализация общей библиотеки классов

REST-сервис, web-приложение и мобильное приложение реализованы на языке Java, и значительная их часть вынесена в виде общего набора библиотек и бизнес-объектов.

6.5.1. Основные классы

В общий набор, реализованный в виде отдельного Maven-модуля, входят следующие классы:

AlreadyDeletedException.java – исключение, выбрасываемое версионированными DAO при попытке удалить уже удалённый объект.

AnalyzeCore.java – интерфейс модуля анализа дневника.

AnalyzeCoreImpl.java – реализация модуля анализа дневника.

AnalyzeExtracter.java – содержит методы извлечения и формализации данных дневника.

AnalyzeRec.java – класс, содержащий минимальную единицу анализируемых данных.

AuthException.java – общее исключение авторизации.

BaseService.java – интерфейс DAO баз продуктов и блюд.

BloodRecord.java – бизнес-объект «Замер сахара крови».

CommonServiceException.java – общее исключение, выбрасываемое версионированным DAO.

DeprecatedAPIException.java – исключение, выбрасываемое сервисом авторизации при попытке войти в систему с клиентом, имеющим устаревшую и не поддерживаемую сервисом версию API.

DiaryRecord.java – общий класс записи дневника.

DiaryService.java – интерфейс DAO дневника.

DishBaseService.java – интерфейс DAO базы блюд.

DishItem.java – бизнес-объект «Блюдо».

DuplicateException.java – исключение, выбрасываемое версионированным DAO при попытке создать объект с уже существующим идентификатором.

FakeDiaryService.java – тестовая реализация DAO дневника для интеграционного тестирования.

FakeDishBaseService.java – тестовая реализация DAO базы блюд для интеграционного тестирования.

FakeFoodBaseService.java – тестовая реализация DAO базы продуктов для интеграционного тестирования.

FakeObjectService.java – тестовая реализация общего DAO для интеграционного тестирования.

Food.java – бизнес-объект «Продукт».

FoodBaseService.java – интерфейс DAO базы продуктов.

FoodItem.java – бизнес-объект «Запись в базе продуктов».

FoodMassed.java – бизнес-объект «Продукт с массой».

InsRecord.java – бизнес-объект «Инъекция».

Koof.java – тройка базовых коэффициентов, является минимальной единицей результатов анализа.

KoofList.java – суточный список коэффициентов, является результатом анализа.

KoofService.java – интерфейса сервиса получения коэффициентов.

KoofServiceImpl.java – реализация интерфейса получения коэффициентов.

MealRecord.java – бизнес-объект «Приём пищи».

Mock.java – интерфейс мок-генератора, содержащий методы для получения новых записей и сравнения двух записей между собой.

MockDiaryRecord.java – мок-генератор записей дневника.

MockDishItem.java – мок-генератор записей базы блюд.

MockFood.java – мок-генератор продуктов.

MockFoodItem.java – мок-генератор записей базы продуктов.

MockFoodMassed.java – мок-генератор взвешенных продуктов.

MockVersionedConverter.java – конвертер мок-генератора в версионированный мок-генератор.

Named.java – интерфейс объекта, имеющего имя.

NamedRelativeTagged.java – интерфейс объекта, имеющего название и числовой тег, а также хранящего информацию об относительном содержании белков, жиров, углеводов и калорийности.

NotAuthorizedException.java – исключение отсутствия авторизации.

NoteRecord.java – бизнес-объект «Заметка».

NotFoundException.java – исключение, выбрасываемое при отсутствии запрошенного объекта.

ObjectService.java – интерфейс версионированного DAO.

Parser.java – интерфейс сериализатора / десериализатора в формат JSON.

ParserDiaryRecord.java – сериализатор / десериализатор записей дневника в формат JSON.

ParserDishItem.java – сериализатор / десериализатор записей базы блюд в формат JSON.

ParserFoodItem.java – сериализатор / десериализатор записей базы продуктов в формат JSON.

ParserDiaryRecord.java – сериализатор / десериализатор взвешенных продуктов в формат JSON.

ParserVersioned.java – конвертер сериализаторов в сериализатор версионированных объектов.

PersistenceException.java – общая ошибка сохранения данных.

PrimeRec.java – первичная запись при извлечении из дневника данных для анализа.

Profiler.java – вспомогательный класс для профилирования приложений.

Relative.java – интерфейс объекта, хранящего информацию об относительном содержании белков, жиров, углеводов и калорийности.

RelevantIndexator.java – класс для релевантной сортировки продуктов и блюд.

ResponseBuilder.java – класс для формирования и чтения запросов при общении между REST-сервисом и клиентскими приложениями.

SearchResult.java – класс сокращённых результатов поиска в компоненте автодополнения.

Serializer.java – интерфейс сериализатора / десериализатора объектов в строку.

SerializerAdapter.java – конвертер сериализатора / десериализатора в формате JSON в сериализатор / десериализатор в простую строку.

SerializerDiaryRecord.java – сериализатор / десериализатор записей дневника в текстовом формате.

SerializerDishItem.java – сериализатор / десериализатор записей базы блюд в текстовом формате.

SerializerFoodItem.java – сериализатор / десериализатор записей базы продуктов в текстовом формате.

SerializerFoodMassedPlain.java – сериализатор / десериализатор взвешенных продуктов в текстовом формате.

Sorter.java – содержит компараторы для версионированных объектов.

StdResponse.java – хранит стандартный ответ REST-сервиса.

SyncService.java – сервис синхронизации версионированных DAO.

Tagged.java – интерфейс объекта, имеющего числовой тег.

TagInfo.java – объект, содержащий числовой тег (релевантность) и универсальный идентификатор (GUID).

TagService.java – интерфейс сервиса, предоставляющего информацию о релевантности объектов.

TestDiaryService.java – интерфейс модуля unit-тестов для DAO дневника.

TestDiaryServiceCommon.java – набор unit-тестов для DAO дневника.

TestDishbaseService.java – интерфейс модуля unit-тестов для DAO базы блюд.

TestDishbaseServiceCommon.java – набор unit-тестов для DAO базы блюд.

TestFoodbaseService.java – интерфейс модуля unit-тестов для DAO базы продуктов.

TestFoodbaseServiceCommon.java – набор unit-тестов для DAO базы продуктов.

TimePoint.java – объект, хранящий пару (время, значение).

UnsupportedAPIException.java – исключение, выбрасываемое сервисом авторизации при отсутствии поддержки запрошенной версии API.

Utils.java – модуль вспомогательных утилит общего назначения.

Versioned.java – стандартный версионированный объект.

VersionedUtils.java – набор утилит для работы с версионированными объектами.

WeightedTimePoint.java – тройка (время, вес, значение).

6.5.1. Модульные тесты

Для проверки корректности работы основных классов были реализованы следующие модульные тесты:

TestAnalyzeService.java – тест сервиса анализа.

TestBloodRecord.java – тест бизнес-объекта «замер сахара крови».

TestDishItem.java – тест бизнес-объекта «элемент базы блюд».

TestFakeDiaryService.java – тест макета сервиса дневника.

TestFakeFoodBaseService.java – тест макета сервиса базы продуктов.

TestFood.java – тест бизнес-объекта «продукт».

TestFoodItem.java – тест бизнес-объекта «элемент базы продуктов».

TestFoodMassed.java – тест бизнес-объекта «продукт с массой».

TestInsRecord.java – тест бизнес-объекта «инъекция».

TestMealRecord.java – тест бизнес-объекта «приём пищи».

TestNoteRecord.java – тест бизнес-объекта «заметка».

TestSerializer.java – тест обобщённого (generics) сериализатора.

TestSerializerDiaryRecord.java – тест сериализатора записей дневника.

TestSerializerDishItem.java – тест сериализатора записей базы блюд.

TestSerializerFoodItem.java – тест сериализатора записей базы продуктов.

TestSerializerFoodMassedPlain.java – тест сериализатора продуктовой с массой.

TestSync.java – тест сервиса синхронизации.

TestUtils.java – тест общих утилит.

6.6. Реализация локального приложения

6.6.1. Список классов

Локальное приложение состоит из следующих модулей:

ACCombo.pas – компонент автодополнения с выпадающим списком.

AnalyzeGraphic.pas – модуль для визуализации работы системы анализа.

AnalyzeInterface.pas – интерфейс взаимодействия системы анализа с главным приложением.

AutoLog.pas – модуль логирования.

Bases.pas – модуль классов, необходимых для реализации баз продуктов и блюд.

BasesTest.pas –unit-тесты для модуля Bases.pas.

BusinessObjects.pas – ключевые бизнес-объекты.

BusinessObjectsTest.pas – unit-тесты для модуля BusinessObjects.pas.

DiaryAnalyze.pas – модуль, хранящий методы для извлечения данных дневника и взаимодействия с системой анализа.

DiaryCore.pas – хранит все данные приложения в виде набора singleton-объектов.

DiaryDAO.pas – интерфейс DAO дневника.

DiaryDAOTest.pas – unit-тесты для модуля DiaryDAO.pas.

DiaryGraphicInterface.pas – модуль работы с графикой для компонента TDiary.

DiaryInterface.pas – вспомогательный модуль, содержащий утилиты для пользовательского интерфейса приложения.

DiaryLocalSource.pas – реализация локального DAO дневника.

DiaryLocalSourceTest.pas – unit-тесты для модуля DiaryLocalSource.pas.

DiaryPageSerializer.pas – хранит методы для сериализации и десериализации записей дневника.

DiaryPageSerializerTest.pas – unit-тесты для модуля DiaryPageSerializer.pas.

DiaryRecords.pas – хранит классы, представляющие записи дневника.

DiaryRoutines.pas – модуль вспомогательных утилит широкого назначения.

DiaryRoutinesTest.pas – unit-тесты для модуля DiaryRoutines.pas.

DiarySync.pas – содержит методы для синхронизации любых версионированных DAO.

DiaryView.pas – компонент для визуального представления дневника.

DiaryWeb.pas – модуль с общими методами работы с REST-сервисом.

DiaryWebSource.pas – реализация сетевого DAO дневника.

DiaryWebSourceTest.pas – unit-тесты для модуля DiaryWebSource.pas.

DishbaseDAO.pas – интерфейс DAO базы блюд.

DishbaseLocalDAO.pas – реализация локального DAO базы блюд.

DishbaseWebDAO.pas – реализация сетевого DAO базы блюд.

FoodbaseDAO.pas – интерфейс DAO базы продуктов.

FoodbaseLocalDAO.pas – реализация локального DAO базы продуктов.

FoodbaseWebDAO.pas – реализация сетевого DAO базы продуктов.

JsonSerializer.pas – модуль содержит общие методы работы с JSON.

JsonVersionedSerializer.pas – модуль содержит методы для сериализации / десериализации версионированных объектов в формате JSON.

MainUnit.pas – хранит главную форму приложения и реализует базовую логику его работы.

ObjectService.pas – интерфейс версионированного DAO.

OldStorage.pas – хранилище для альфа-версий алгоритмов анализа.

SettingsINI.pas – модуль для работы с настройками приложения.

TextInterface.pas – модуль для локализации интерфейса приложеиия.

ThreadTask.pas – модуль для создания и управления задачами в многопоточном режиме.

UnitAbout.pas – модуль формы «О программе».

UnitDataInterface.pas – хранилище интерфейсных данных приложения.

UnitEditDish.pas – модуль формы «Редактор блюда».

UnitEditFood.pas – модуль формы «Редактор продукта».

UnitEditorBlood.pas – модуль формы «Редактор замера сахара крови».

UnitEditorIns.pas – модуль формы «Редактор инъекции».

UnitEditorMeal.pas – модуль формы «Редактор приёма пищи».

UnitEditorNote.pas – модуль формы «Редактор заметки».

UnitExportText.pas – модуль формы «Экспорт дневника».

UnitFirstMan.pas – модуль формы «Первоначальная установка».

UnitLogViewer.pas – модуль формы «Просмотр лог-файлов».

UnitMisc.pas – модуль формы «Экспериментальные возможности».

UnitSettings.pas – модуль формы «Настройки».

UnitShadow.pas – модуль формы «Эффект затенения».

UnitStartup.pas – модуль формы «Прогресс задачи».

6.6.2. Особенности реализации локального приложения

В процессе разработки локального приложения был создан ряд компонентов, предназначенных для выбора и редактирования данных пользователем, а также для вывода графической информации.

Компонент TDiaryView позволяет пользователю просматривать и редактировать дневник. Он обладает следующими особенностями:

* позволяет отображать список записей дневника;
* позволяет обрабатывать одинарные и двойные клики;
* позволяет полностью менять внешний вид (цветовое оформление, отступы, шрифт, текст сообщения об отсутствии данных);
* позволяет добавлять контекстные меню для каждого отдельного типа записей;
* позволяет выводить рядом с продуктами произвольную пользовательскую информацию (используется для отображения гликемического эффекта продукта);
* использует двойную буферизацию для улучшения внешнего вида при перерисовке;
* поддерживает автоматический выбор кодировки для отображения данных;
* поддерживает перемещение продуктов в блюдах методом drag-and-drop;
* поддерживает подсветку выделенных продуктов и блюд и её сохранение при модификации данных на основе GUID записи.

Компонент TACComboBox предназначен для отображения выпадающего списка автодополнения. Он используется для выбора продуктов и блюд в дневнике и редакторе блюд. Компонент обладает следующими особенностями:

* при отображении данных используется виртуальная модель (т.е. перерисовываются только элементы, видимые в настоящий момент пользователю), что позволяет эффективно работать со списками из нескольких тысяч элементов;
* каждый элемент списка может быть дополнен иконкой;
* каждый элемент списка может иметь собственный цвет шрифта (это используется для улучшения визуального восприятия отсортированного по релевантности списка: самые актуальные элементы отображаются чёрным шрифтом, по мере уменьшения релевантности шрифт становится серым);
* список автодополнения строится на основе двух списков, следующих друг за другом в указанном порядке:
  + список элементов, в которых присутствуют слова, начинающиеся с введённой строки;
  + список элементов, в который присутствует введённая строка;
* каждый элемент списка может быть дополнен справа двумя текстовыми подсказками (это используется для вывода относительной углеводности продуктов и блюд и даты модификации последних).

Компонент TNumberEdit предназначен для ввода вещественных числовых значений. Он используется для ввода масс, а также некоторых настроек. Компонент обладает следующими особенностями:

* разрешён ввод только цифр, десятичного разделителя и символов арифметических операций;
* поддерживается автоматический выбор верного десятичного разделителя (например, если в системе таковым установлена запятая, то при вводе точки она будет заменена на запятую);
* поддерживается обработка простых арифметических выражений;
* при вводе некорректного выражения компонент меняет цвет (по умолчанию – на светло-красный, возможна настройка);
* поддерживается проверка введённого значения (в том числе выражения) на неотрицательность.

Компонент TStatProgress предназначен для отображения индикатора прогресса. Используется для отображения статистики по белкам, жирам и углеводам за день относительно суточных норм. Компонент обладает следующими особенностями:

* имеет два основных параметра: значение и лимит;
* компонент состоит из двух частей:
  + первая (визуально составляет 70% компонента) заполняется линейно, пока значение не превосходит лимит;
  + вторая (визуально составляет 30% компонента) заполняется после заполнения первой части и использует логарифмическую модель;
* цвет каждой из частей компонента настраивается отдельно;
* компонент имеет объёмную стилизацию.

6.7. Реализация web-приложения

6.7.1. Основные классы

Web-приложение включает в себя следующие классы:

AboutPage.java – web-страница «Описание».

AuthDAO.java – интерфейс DAO сервиса авторизации.

AuthorizedRestClient.java – авторизационный REST-клиент с возможностью сохранения учётных данных.

AuthRestService.java – REST-сервис авторизации.

AuthService.java – интерфейс авторизационного сервиса.

BaseDAO.java – обобщённый (generics) интерфейс базы продуктов / блюд.

Config.java – хранилище параметров приложения.

DetachableFoodModel.java – модель продукта (wicket).

DiaryDAO.java – интерфейс DAO дневника.

DiaryPage.java – web-страница «Дневник».

DiaryRestService.java – REST-сервис дневника.

DishbaseDAO.java – интерфейс DAO базы блюд.

DishBaseRestService.java – REST-сервис базы блюд.

DownloadPage.java – web-страница «Скачать».

FakeAuthDAO.java – макет авторизационного DAO.

FakeDiaryDAO.java – макет DAO дневника.

FakeFoodbaseDAO.java – макет DAO базы продуктов.

FoodbaseDAO.java – интерфейс DAO базы продуктов.

FoodBasePage.java – web-страница «Базы».

FoodBaseRestService.java – REST-сервис базы продуктов.

FoodDataProvider.java – поставщик данных для модели продуктов.

FrontendAuthService.java – авторизационный frontend-сервис.

FrontendDiaryService.java – frontend-сервис дневника.

FrontendDishbaseService .java – frontend-сервис базы блюд.

FrontendFoodbaseService.java – frontend-сервис базы продуктов.

HeaderPanel.java – компонент заголовка web-страниц.

LoginPage.java – web-страница авторизации.

MasterPage.java – родительская web-страница.

Menu.java – компонент меню (wicket).

MenuContent.java – содержимое меню (wicket).

MenuItem.java – элемент меню (wicket).

MySQLAccess.java – класс для доступа к базе данных MySQL.

MySQLAuthDAO.java – реализация авторизационного DAO.

MySQLDiaryDAO.java – реализация DAO дневника.

MySQLDishbaseDAO.java – реализация DAO базы блюд.

MySQLFoodbaseDAO.java – реализация DAO базы продуктов.

RestClient.java – базовый REST-клиент.

SystemRestService.java – служебный REST-сервис.

UserSessionUtils.java – вспомогательный класс для работы с web-сессией.

Utils.java – вспомогательные методы.

WicketApplication.java – главный класс wicket-приложения.

6.7.2. Модульные тесты

Для проверки корректности работы ключевых систем были разработаны следующие модульные тесты:

TestAuthWebService.java – тест web-реализации авторизационного сервиса.

TestDiaryWebService.java – тест web-реализации сервиса дневника.

TestDishbaseWebService.java – тест web-реализации сервиса базы блюд.

TestFoodbaseWebService.java – тест web-реализации сервиса базы продуктов.

TestMySQLAuthDAO.java – тест авторизационного DAO.

6.7.3. Среда выполнения

В качестве среды выполнения web-приложения и REST-сервиса используется Apache Tomcat 7.0 в связке с сервером непрерывной интеграции (CI, continuous integration) Jenkins. CI-сервер имеет две активные задачи (job):

* comp-server;
* comp-server-test.

Задача comp-server каждые десять минут проверяет наличие изменений в репозитории исходного кода SVN, и при их обнаружении запускает обновление внутреннего локального хранилища, затем полную чистую сборку приложения. Тесты на этой стадии пропускаются. Если компиляция и сборка прошли успешно, готовое приложение загружается на тестовый сервер с выделенной тестовой средой MySQL, после чего вызывается задача comp-server-test.

Задача comp-server-test проводит полное тестирование развёрнутого приложения, включая модульное и интеграционное тестирование. В случае успеха приложение загружается на внутренний сервер разработчика (development server).

Затем следует фаза ручного функционального тестирования приложения с использованием клиентских приложений. Тестирование проводится на реальных задачах (test cases) разработчика в течение 1-2 недель, после чего приложение развёртывается на рабочий сервер (production server).

6.8. Реализация мобильного приложения

6.8.1. Основные классы

Мобильное приложение включает в себя следующие классы:

ActivityBase.java – активность «База продуктов и блюд».

ActivityDiary.java – активность «Дневник».

ActivityEditor.java – абстрактная активность «Редактор».

ActivityEditorBlood.java – активность «Редактор замера сахара крови».

ActivityEditorDish.java – активность «Редактор блюда».

ActivityEditorFood.java – активность «Редактор продукта».

ActivityEditorIns.java – активность «Редактор инъекции».

ActivityEditorMeal.java – активность «Редактор приёма пищи».

ActivityEditorNote.java – активность «Редактор заметки».

ActivityMain.java – активность «Меню».

ActivityPreferences.java – активность «Настройки».

AndroidUtils.java – Android-специфичные утилиты.

AuthException.java – исключение авторизации.

ConnectionException.java – исключение соединения.

DeprecatedAPIException.java – исключение устаревшего API.

DiaryContentProvider.java – главный контент-провайдер.

DiaryLocalService.java – локальная реализация сервиса дневника.

DiaryView.java – компонент для просмотра и редактирования дневника.

DiaryWebService.java – web-реализация сервиса дневника.

DishBaseLocalService.java – локальная реализация сервиса базы блюд.

DishBaseWebService.java – web-реализация сервиса базы блюд.

ErrorHandler.java – набор утилит для логирования и обработки ошибок.

FileWorker.java – набор утилит для работы с файловой системой.

FoodBaseLocalService.java – локальная реализация сервиса базы продуктов.

FoodBaseWebService.java – web-реализация сервиса базы продуктов.

FoodDishPicker.java – компонент для выбора продуктов и блюд.

FoodDishTextView.java – элемент выпадающего списка автодополнения.

HardcodedAnalyzeService.java – макет сервиса анализа.

HardcodedFoodbase.java – макет сервиса базы продуктов.

MealEditorView.java – компонент для редактирования списка продуктов и блюд.

MealFormatter.java – набор утилит для отображения приёма пищи.

RecordClickListener.java – интерфейс подписчика события нажатия на запись дневника.

ResponseFormatException.java – исключение неверного формата ответа сервера.

Storage.java – хранилище данных (singleton).

TagLocalService.java – локальная реализация сервиса тегирования версионированных объектов по их релевантности.

TaskExecutionException.java – исключение выполнения асинхронной задачи.

UIUtils.java – утилиты пользовательского интерфейса.

UndefinedFieldException.java – исключение отсутствия необходимого аргумента.

WebClient.java – базовый класс для доступа к REST-сервисам.

WebClientException.java – общее исключение REST-клиента.

6.8.2. Модульные тесты

Для проверки корректности работы ключевых систем были разработаны следующие модульные тесты:

TestAndroidUtils.java – тест утилит Android.

TestDiaryLocalService.java – тест локальной реализации сервиса дневника.

TestDiaryWebService.java – тест web-реализации сервиса дневника.

TestDishBaseLocalService.java – тест локальной реализации сервиса базы блюд.

TestDishBaseWebService.java – тест web-реализации сервиса базы блюд.

TestFoodBaseLocalService.java – тест локальной реализации сервиса базы продуктов.

TestFoodBaseWebService.java – тест web-реализации сервиса базы продуктов.

TestTagLocalService.java – тест реализации сервиса тегов.

TestWebClient.java – тест базового класса доступа к REST-сервисам.

7. План тестирования

Основная функциональность реализованных приложений была покрыта автоматическими модульными и интеграционными тестами, описанными в предыдущем разделе. Дополнительно был проведён ряд ручных функциональных тестов, представленных ниже.

7.1. Тестирование локального приложения

**Тест 1**. Добавление записей в базу продуктов.

Цель: проверка работы с базой продуктов.

Порядок проведения: перейти на вкладку «Базы», создать новый продукт, заполнив все необходимые поля.

Результат: продукт успешно создан и отобразился в таблице продуктов.

**Тест 2**. Добавление записей в базу блюд.

Цель: проверка работы с базой блюд.

Порядок проведения: перейти на вкладку «Базы», создать новое блюдо, добавив в него несколько продуктов и указав корректную массу готового блюда.

Результат: блюдо успешно создано и отобразилось в таблице блюд.

**Тест 3**. Добавление записей в дневник.

Цель: проверка работы с дневником пользователя.

Порядок проведения: перейти на вкладку «Дневник», создать новую запись с замером сахара крови (инъекцией, приёмом пищи, заметкой).

Результат: запись успешно создана и отображена в дневнике на соответствующей странице.

**Тест 4**. Вычисление доз инсулина.

Цель: проверка системы анализа.

Порядок проведения: записать в дневник тестовый набор данных (включая замеры сахара крови, инъекции и приёмы пищи), вручную запустить систему анализа, проверить рекомендуемую дозу инсулина для всех записанных приёмов пищи.

Результат: анализ и построение модели прошли успешно, рекомендуемые приложением дозировки инсулина адекватны введённым данным.

**Тест 5**. Синхронизация посредством REST-сервиса.

Цель: проверка системы синхронизации.

Порядок проведения: записать в дневник, базу продуктов и базу блюд тестовый набор данных, настроить и выполнить синхронизацию; установить второй экземпляр приложения, настроить и выполнить синхронизацию в нём.

Результат: все данные успешно переданы и отображены во втором экземпляре приложения.

7.2. Тестирование web-приложения

**Тест 1**. Защита от неавторизованного доступа.

Цель: проверка системы авторизации.

Порядок проведения: попытаться открыть страницу /diary/ без предварительной авторизации.

Результат: web-приложение перенаправило пользователя на страницу авторизации.

**Тест 2**. Авторизация (неверные данные).

Цель: проверка системы авторизации.

Порядок проведения: открыть страницу авторизации и ввести неверный логин / пароль.

Результат: полученное сообщение «Неверный логин / пароль» соответствует требованиям.

**Тест 3**. Авторизация (верные данные).

Цель: проверка системы авторизации.

Порядок проведения: открыть страницу авторизации и ввести верный логин и пароль.

Результат: web-приложение перенаправило пользователя на страницу с дневником, что соответствует требованиям.

**Тест 4**. Работа с дневником.

Цель: проверка работы с дневником пользователя.

Порядок проведения: перейти на вкладку «Дневник», создать новую запись с замером сахара крови (инъекцией, приёмом пищи, заметкой).

Результат: запись успешно создана и отображена в дневнике на соответствующей странице.

**Тест 5**. Синхронизация посредством REST-сервиса.

Цель: проверка системы синхронизации.

Порядок проведения: записать в дневник, базу продуктов и базу блюд тестовый набор данных; установить экземпляр локального приложения, настроить и выполнить синхронизацию в нём.

Результат: все данные успешно переданы и отображены в локальном приложении.

7.3. Тестирование мобильного приложения

**Тест 1**. Добавление записей в базу продуктов.

Цель: проверка работы с базой продуктов.

Порядок проведения: перейти в активность «Базы», создать новый продукт, заполнив все необходимые поля.

Результат: продукт успешно создан и отобразился в списке продуктов и блюд.

**Тест 2**. Добавление записей в базу блюд.

Цель: проверка работы с базой блюд.

Порядок проведения: перейти в активность «Базы», создать новое блюдо, добавив в него несколько продуктов и указав корректную массу готового блюда.

Результат: блюдо успешно создано и отобразилось в списке продуктов и блюд.

**Тест 3**. Добавление записей в дневник.

Цель: проверка работы с дневником пользователя.

Порядок проведения: перейти в активность «Дневник», создать новую запись с замером сахара крови (инъекцией, приёмом пищи, заметкой).

Результат: запись успешно создана и отображена в дневнике на соответствующей странице.

**Тест 4**. Вычисление доз инсулина.

Цель: проверка системы анализа.

Порядок проведения: записать в дневник тестовый набор данных (включая замеры сахара крови, инъекции и приёмы пищи), вручную запустить систему анализа, проверить рекомендуемую дозу инсулина для всех записанных приёмов пищи.

Результат: анализ и построение модели прошли успешно, рекомендуемые приложением дозировки инсулина адекватны введённым данным.

**Тест 5**. Синхронизация посредством REST-сервиса.

Цель: проверка системы синхронизации.

Порядок проведения: записать в дневник, базу продуктов и базу блюд тестовый набор данных, настроить и выполнить синхронизацию; установить экземпляр локального приложения, настроить и выполнить синхронизацию в нём.

Результат: все данные успешно переданы и отображены в локальном экземпляре приложения.

Заключение

Разработана и реализована информационная система, предназначенная для индивидуального мониторинга и контроля инсулинозависимого сахарного диабета первого типа. Разработан набор приложений для различных платформ на базе клиент-серверных технологий:

1. REST-сервис, который предоставляет следующие функциональные возможности:

* авторизация пользователей;
* чтение и запись дневника пользователя;
* чтение и запись баз продуктов пользователя;
* чтение и запись баз блюд пользователя.

1. Клиентские приложения:

* локальное приложение, которое будет установлено на ПК пользователя; должно использовать локальную базу данных с возможностью синхронизации с БД, хранящейся на сервере, посредством REST-сервиса;
* web-приложение, позволяющее работать с системой с помощью web-браузера;
* мобильное Android-приложение, которое будет установлено на смартфоне (планшете) пользователя; должно использовать локальную базу данных с возможностью синхронизации с БД, хранящейся на сервере, посредством REST-сервиса.

Все клиентские приложения предоставляют следующие функции:

* ведение дневника пользователя;
* ведение баз продуктов и блюд;
* статистический анализ дневника пользователя;
* построение адаптивной модели углеводного обмена;
* вычисление необходимой дозировки инсулина.

При реализации клиентских приложений учтены следующие особенности:

* ограничения платформы реализации;
* характеристики устройств, на которых будет установлено клиентское приложение;
* организация хранения локальных данных;
* интерфейс пользователя.

Список литературы

1. Кадомский Ю. Точная инсулинотерапия сахарного диабета I типа / Ю. Кадомский. – Рига: Библиотека Диа-клуба. – 99 с.
2. Каналес Х. Виртуозная инсулинотерапия / Хорхе Каналес. – Москва: Автономная некоммерческая благотворительная организация «Диабетическая газета», 2002. – 209 с.
3. Харрис Э. PHP/MySQL для начинающих : Пер. с англ. / Э. Харрис. – КУДИЦ-Образ, 2005. – 384 с.
4. Стивенс, Род. Delphi. Готовые алгоритмы : Пер. с англ. / Р. Стивенс. – Москва: ДМК Пресс, 2001. – 378 с.
5. Дейт, К. Дж. Введение в системы баз данных : Пер. с англ. / К. Дейт. – 6-е изд. – Москва; Санкт-Петербург; Киев : Вильямс, 2000. – 846 с.
6. JAVA2. Библиотека профессионала, том II. Тонкости программирования / К. Хорстман, Г. Корнелл; 7-е изд., - Москва: Издательский дом «Вильямс», 2006. – 1168 с.
7. Java Server Pages: The complete Reference. / Ed Burns, Chris Shalk, Neil Griffin; 1-е изд., - USA, New York City: McGraw-Hill Osborne Media, 2009. – 752 с.
8. Practical Richfaces / Max Katz, Ilya Shaikovsky; 2-изд., - USA, New York City: Apress, 2008. – 690 с.
9. Eclipse: [сайт]. URL: <http://www.eclipse.org/> (дата обращения: 06.12.2013).
10. Jenkins: [сайт]. URL: <http://jenkins-ci.org/> (дата обращения: 26.01.2014).
11. Apache Tomcat: [сайт]. URL: <http://tomcat.apache.org/> (дата обращения: 25.01.2014).
12. Apache Maven: [сайт]. URL: <http://maven.apache.org/> (дата обращения: 23.01.2014).
13. Apache Wicket [сайт]. URL: http://wicket.apache.org/ (дата обращения: 02.04.2014).
14. MySQL: [сайт]. URL: <http://www.mysql.com/> (дата обращения: 28.01.2014).
15. MySQL Workbench: [сайт]. URL: <http://www.mysql.com/products/workbench/> (дата обращения: 29.01.2014).
16. Visual Paradigm: [сайт]. URL: <http://www.visual-paradigm.com/> (дата обращения: 12.01.2014).
17. QueryDSL: [сайт]. URL: <http://www.querydsl.com/> (дата обращения: 04.05.2014).
18. JSP Technology: [сайт]. URL: <http://www.oracle.com/technetwork/java/tech/persistence-jsp-140049.html> (дата обращения: 19.03.2014).
19. JSP Technology: [сайт]. URL: <http://www.jsptut.com/> (дата обращения: 16.02.2014).
20. Stack Overflow: [сайт]. URL: http://stackoverflow.com/ (дата обращения: 28.04.2014).
21. Android Developers: [сайт]. URL: http://developer.android.com/index.html (дата обращения: 18.12.2013).
22. Subversion: [сайт]. URL: <http://subversion.apache.org/> (дата обращения: 02.03.2014).

Приложение 1. Описание таблиц БД

Таблица П1.1. Описание полей таблицы «User»

|  |  |  |
| --- | --- | --- |
| **Имя поля** | **Тип поля** | **Описание** |
| ID | integer(11) | Первичный ключ |
| Login | char(50) | Логин (адрес электронной почты) пользователя |
| HashPass | char(32) | Хэш пароля пользователя |
| DateLogin | timestamp | Дата последней авторизации |
| DateReg | timestamp | Дата регистрации пользователя |
| ActivationKey | char(32) | Ключ активации, необходимой после регистрации |

Таблица П1.2. Описание полей таблицы «Diary»

|  |  |  |
| --- | --- | --- |
| **Имя поля** | **Тип поля** | **Описание** |
| \_GUID | char(32) | Первичный ключ |
| \_UserID | integer(11) | Идентификатор пользователя, которому принадлежит запись (внешний ключ) |
| \_TimeStamp | timestamp | Дата и время последней модификации записи |
| \_Version | integer(11) | Номер версии записи; используется при синхронизации для определения направления передачи данных |
| \_Deleted | integer(1) | Метка удалённости записи |
| \_Content | blob | Данные записи в текстовом виде (JSON) |
| \_TimeCache | timestamp | Время записи (дублируется из \_Content) для быстрого доступа |

Таблица П1.3. Описание полей таблицы «Food»

|  |  |  |
| --- | --- | --- |
| **Имя поля** | **Тип поля** | **Описание** |
| \_GUID | char(32) | Первичный ключ |
| \_UserID | integer(11) | Идентификатор пользователя, которому принадлежит продукт (внешний ключ) |
| \_TimeStamp | timestamp | Дата и время последней модификации продукта |
| \_Version | integer(11) | Номер версии продукта; используется при синхронизации для определения направления передачи данных |
| \_Deleted | integer(1) | Метка удалённости продукта |
| \_Content | blob | Данные продукта в текстовом виде (JSON) |
| \_NameCache | varchar(100) | Название продукта (дублируется из \_Content) для быстрого доступа |

Таблица П1.4. Описание полей таблицы «Dish»

|  |  |  |
| --- | --- | --- |
| **Имя поля** | **Тип поля** | **Описание** |
| \_GUID | char(32) | Первичный ключ |
| \_UserID | integer(11) | Идентификатор пользователя, которому принадлежит блюдо (внешний ключ) |
| \_TimeStamp | timestamp | Дата и время последней модификации блюда |
| \_Version | integer(11) | Номер версии блюда; используется при синхронизации для определения направления передачи данных |
| \_Deleted | integer(1) | Метка удалённости блюда |
| \_Content | blob | Данные блюда в текстовом виде (JSON) |
| \_NameCache | varchar(100) | Название блюда (дублируется из \_Content) для быстрого доступа |

Приложение 2. Форматы данных

Все версионированные объекты (записи дневника, продукты и блюда) имеют следующий формат (пример):

{

"id": "8b0c99babe6f48988bfaf47eacd1dcd4",

"stamp": "2014-05-01 20:40:00",

"version": 1,

"deleted": false,

"data": <данные>

}

Поля имеют следующий смысл:

* id – уникальный идентификатор записи (GUID).
* stamp – дата и время последней модификации записи.
* version – номер версии записи.
* deleted – метка удалённости записи.
* data – данные объекта.

В зависимости от типа объекта, поле data может иметь различное наполнение.

Для замера сахара крови данные выглядят следующим образом (пример):

{

"time": "2014-05-01 20:40:00",

"type": "blood",

"value": 4.6,

"finger": 1

}

Поля имеют следующий смысл:

* time – дата и время записи.
* type – тип записи, всегда «blood».
* value – значение сахара крови (ммоль/л).
* finger – номер пальца, из которого бралась кровь.

Для инъекции инсулина данные выглядят следующим образом (пример):

{

"time": "2014-05-01 20:58:00",

"type": "ins",

"value": 15

}

Поля имеют следующий смысл:

* time – дата и время записи.
* type – тип записи, всегда «ins».
* value – дозировка инсулина.

Для приёма пищи данные выглядят следующим образом (пример):

{

"time": "2014-05-01 21:20:00",

"type": "meal",

"short": false,

"content": [

{

"mass": 46,

"name": "Молоко \"Вкуснотеево\", 3,2%",

"fats": 3.2,

"value": 58,

"prots": 2.8,

"carbs": 4.7

},

{

"mass": 103,

"name": "Запеканка творожная",

"fats": 8,

"value": 233.6,

"prots": 11.8,

"carbs": 30.3

}

]

}

Поля имеют следующий смысл:

* time – дата и время записи.
* type – тип записи, всегда «meal».
* short – метка сокращённого постпрандиального периода.
* content – состав приёма пищи.
* name – название продукта.
* prots – содержание белков в 100 граммах продукта.
* fats – содержание жиров в 100 граммах продукта.
* carbs – содержание углеводов в 100 граммах продукта.
* value – калорийность 100 грамм продукта.
* mass – масса продукта.

Для текстовой заметки данные выглядят следующим образом (пример):

{

"time": "2014-05-09 17:30:00",

"type": "note",

"text": "велопробег"

}

Поля имеют следующий смысл:

* time – дата и время записи.
* type – тип записи, всегда «note».
* text – текст заметки.

Для записи базы продуктов данные выглядят следующим образом (пример):

{

"name": "iБлин со сгущенным молоком (Теремок)",

"prots": 5.5,

"fats": 11,

"carbs": 36.7,

"value": 267.4,

"table": true,

"tag": 0

}

Поля имеют следующий смысл:

* name – название продукта.
* prots – содержание белков в 100 граммах продукта.
* fats – содержание жиров в 100 граммах продукта.
* carbs – содержание углеводов в 100 граммах продукта.
* value – калорийность 100 грамм продукта.
* table – признак того, что данные взяты из таблицы, а не с этикетки продукта.
* tag – релевантность продукта.

Для записи базы блюд данные выглядят следующим образом (пример):

{

"name": "Макароны",

"mass": 545,

"tag": 62290,

"content": [

{

"mass": 218,

"name": "Макароны рифлёные №10 саранские",

"fats": 1.1,

"value": 344,

"prots": 10.4,

"carbs": 71.5

},

{

"mass": 33,

"name": "Масло \"Вкуснотеево\" жёлтое",

"fats": 72.5,

"value": 661,

"prots": 0.8,

"carbs": 1.3

}

]

}

Поля имеют следующий смысл:

* name – название блюда.
* mass – масса готового блюда.
* tag – релевантность блюда.
* content – состав блюда; имеет тот же формат, что и состав приёма пищи.

Приложение 3. Листинг модуля AnalyzeCoreImpl.java

package org.bosik.diacomp.core.services.analyze;

import java.util.ArrayList;

import java.util.List;

import org.bosik.diacomp.core.services.analyze.entities.AnalyzeRec;

import org.bosik.diacomp.core.services.analyze.entities.KoofList;

import org.bosik.diacomp.core.services.analyze.entities.WeightedTimePoint;

import org.bosik.diacomp.core.utils.Utils;

public class AnalyzeCoreImpl implements AnalyzeCore

{

public AnalyzeCoreImpl(double approxFactor)

{

// pre-calculation

for (int i = 0; i < TIME\_WEIGHTS.length; i++)

{

TIME\_WEIGHTS[i] = Math.exp(-approxFactor \* Math.pow(i / Utils.HalfMinPerDay, 2));

}

}

private class Bean

{

public double p;

public double q;

public double g[] = new double[3];

public Bean()

{

}

}

private interface DevFunction

{

double calculate(double x, double y);

}

class AbsDev implements DevFunction

{

@Override

public double calculate(double x, double y)

{

return Math.abs(x - y);

}

}

class SqrDev implements DevFunction

{

@Override

public double calculate(double x, double y)

{

return Math.pow(x - y, 2);

}

}

class RelDev implements DevFunction

{

@Override

public double calculate(double x, double y)

{

return Math.abs(1 - (x / y));

}

}

private static final double TIME\_WEIGHTS[] = new double[Utils.HalfMinPerDay + 1];

private static double calculateK(AnalyzeRec rec, double q, double p)

{

if (Math.abs(rec.getCarbs()) > Utils.EPS)

{

return (((rec.getBsOut() - rec.getBsIn()) + (rec.getIns() \* q)) - (rec.getProts() \* p)) / rec.getCarbs();

}

else

{

return Double.NaN;

}

}

private static Double calculateW(AnalyzeRec rec)

{

return rec.getWeight();

}

private static WeightedTimePoint[] calculateKW(List<AnalyzeRec> recs, double q, double p)

{

WeightedTimePoint[] result = new WeightedTimePoint[recs.size()];

for (int i = 0; i < result.length; i++)

{

result[i] = new WeightedTimePoint();

result[i].setTime(recs.get(i).getTime());

result[i].setValue(calculateK(recs.get(i), q, p));

result[i].setWeight(calculateW(recs.get(i)));

}

return result;

}

private static int timeDistance(int time1, int time2)

{

int result = Math.abs(time1 - time2);

if (result > Utils.HalfMinPerDay)

{

result = Utils.MinPerDay - result;

}

return result;

}

private static double timeWeight(int time1, int time2)

{

int dist = timeDistance(time1, time2);

return TIME\_WEIGHTS[dist];

}

private static double approximatePoint(WeightedTimePoint[] points, int time)

{

double summ = 0.0;

double summWeight = 0.0;

for (int i = 0; i < points.length; i++)

{

if (!Double.isNaN(points[i].getValue()))

{

// TODO: optimize

double curWeight = points[i].getWeight() \* timeWeight(time, points[i].getTime());

summWeight += curWeight;

summ += points[i].getValue() \* curWeight;

}

}

if (Math.abs(summWeight) < Utils.EPS)

{

return Double.NaN;

}

else

{

return summ / summWeight;

}

}

private static double[] approximate(WeightedTimePoint[] points)

{

double[] result = new double[Utils.MinPerDay];

// TODO: improve handling

if (points.length == 0)

{

for (int i = 0; i < Utils.MinPerDay; i++)

{

result[i] = Double.NaN;

}

}

else

{

for (int i = 0; i < Utils.MinPerDay; i++)

{

result[i] = approximatePoint(points, i);

}

}

return result;

}

private static double getRand(double[] recs, WeightedTimePoint[] points, DevFunction func)

{

double result = 0.0;

int n = 0;

for (int i = 0; i < points.length; i++)

if (!Double.isNaN(points[i].getValue()))

{

result += func.calculate(recs[points[i].getTime()], points[i].getValue());

n++;

}

if (n > 0)

{

result /= n;

}

return result;

}

private static double getDev(List<AnalyzeRec> recs, KoofList koofs, DevFunction func)

{

double result = 0.0;

int n = 0;

for (AnalyzeRec rec : recs)

{

result += func.calculate(

((rec.getBsIn() + (koofs.getKoof(rec.getTime()).getK() \* rec.getCarbs())) - (koofs.getKoof(

rec.getTime()).getQ() \* rec.getIns()))

+ (koofs.getKoof(rec.getTime()).getP() \* rec.getProts()), rec.getBsOut());

n++;

}

if (n > 0)

{

result /= n;

}

return result;

}

private static KoofList copyKQP(double[] ks, double q, double p)

{

KoofList result = new KoofList();

for (int i = 0; i < Utils.MinPerDay; i++)

{

result.getKoof(i).setK(ks[i]);

result.getKoof(i).setQ(q);

result.getKoof(i).setP(p);

}

return result;

}

@Override

public KoofList analyze(List<AnalyzeRec> recs)

{

/\*\*

\* This method assumes the Q and P koofs are fixed and K is floating within the day

\*/

if (recs.isEmpty())

{

//throw new IllegalArgumentException("Recs list is empty");

return null;

}

final double DISC\_Q = 0.0125;

final double MIN\_Q = 1.50;

final double MAX\_Q = 5.00 + (DISC\_Q / 2);

final double DISC\_P = 0.05;

final double MIN\_P = 0.00;

final double MAX\_P = 0.00 + (DISC\_P / 2);

KoofList koofs = new KoofList();

WeightedTimePoint[] points;

List<Bean> V = new ArrayList<Bean>();

double k[];

DevFunction funcRelative = new RelDev();

DevFunction funcSqr = new SqrDev();

for (double q = MIN\_Q; q < MAX\_Q; q += DISC\_Q)

{

for (double p = MIN\_P; p < MAX\_P; p += DISC\_P)

{

Bean bean = new Bean();

bean.q = q;

bean.p = p;

points = calculateKW(recs, q, p);

k = approximate(points);

koofs = copyKQP(k, q, p);

bean.g[0] = getRand(k, points, funcRelative);

bean.g[1] = 0.0;

bean.g[2] = getDev(recs, koofs, funcSqr);

V.add(bean);

}

}

// normalizing weights

for (int n = 0; n < 3; n++)

{

double min = Double.MAX\_VALUE;

double max = Double.MIN\_VALUE;

for (Bean bean : V)

{

if (!Double.isNaN(bean.g[n]))

{

min = Math.min(min, bean.g[n]);

max = Math.max(max, bean.g[n]);

}

}

if (Math.abs(min - max) > Utils.EPS)

{

for (Bean bean : V)

{

if (!Double.isNaN(bean.g[n]))

{

bean.g[n] = (bean.g[n] - min) / (max - min);

}

}

}

}

// search for best solution

double bestDev = Double.MAX\_VALUE;

double bestQ = Double.NaN;

double bestP = Double.NaN;

for (Bean bean : V)

{

double curDev = Math.pow(bean.g[0], 2) + Math.pow(bean.g[1], 2) + Math.pow(bean.g[2], 2);

if (curDev < bestDev)

{

bestDev = curDev;

bestQ = bean.q;

bestP = bean.p;

}

}

// restore

points = calculateKW(recs, bestQ, bestP);

k = approximate(points);

koofs = copyKQP(k, bestQ, bestP);

return koofs;

}

}

Приложение 4. Листинг модуля SyncService.java

package org.bosik.diacomp.core.services.sync;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Date;

import java.util.List;

import org.bosik.diacomp.core.entities.tech.Versioned;

import org.bosik.diacomp.core.services.ObjectService;

public class SyncService

{

// private static final String TAG = "SyncService";

/\* ============================ HELPER CLASSES ============================ \*/

public static interface Callback

{

public void update\_max(int max);

public void update\_progress(int progress);

}

/\* ============================ METHODS ============================ \*/

/\*\*

\* Calculates lists for synchronization

\*

\* @param items1

\* First list

\* @param items2

\* Second list

\* @param newer1

\* Items which has greater version in the first list

\* @param newer2

\* Items which has greater version in the second list

\* @param only1

\* Items which are presented only in the first list

\* @param only2

\* Items which are presented only in the second list

\*/

private static <T> void getOverLists(List<Versioned<T>> items1, List<Versioned<T>> items2,

List<Versioned<T>> newer1, List<Versioned<T>> newer2, List<Versioned<T>> only1, List<Versioned<T>> only2)

{

// null checks

if (null == items1)

{

throw new NullPointerException("items1 can't be null");

}

if (null == items2)

{

throw new NullPointerException("items2 can't be null");

}

if (null == newer1)

{

throw new NullPointerException("newer1 can't be null");

}

if (null == newer2)

{

throw new NullPointerException("newer2 can't be null");

}

if (null == only1)

{

throw new NullPointerException("only1 can't be null");

}

if (null == only2)

{

throw new NullPointerException("only2 can't be null");

}

// preparation

Collections.sort(items1, Versioned.COMPARATOR\_GUID);

Collections.sort(items2, Versioned.COMPARATOR\_GUID);

newer1.clear();

newer2.clear();

only1.clear();

only2.clear();

int i = 0;

int j = 0;

// parallel processing

while ((i < items1.size()) && (j < items2.size()))

{

Versioned<T> p1 = items1.get(i);

Versioned<T> p2 = items2.get(j);

int c = Versioned.COMPARATOR\_GUID.compare(p1, p2);

if (c < 0)

{

only1.add(p1);

i++;

}

else if (c > 0)

{

only2.add(p2);

j++;

}

else

{

if (p1.getVersion() > p2.getVersion())

{

newer1.add(p1);

}

else if (p1.getVersion() < p2.getVersion())

{

newer2.add(p2);

}

i++;

j++;

}

}

// finish first list

while (i < items1.size())

{

only1.add(items1.get(i));

i++;

}

// finish second list

while (j < items2.size())

{

only2.add(items2.get(j));

j++;

}

}

/\*\*

\* Synchronizes two object services

\*

\* @param <T>

\*

\* @param service1

\* First service

\* @param service2

\* Second service

\* @param since

\* Modification time limiter: items modified after this time is taking in account

\* only

\* @return Total number of transferred items

\*/

public static <T> int synchronize(ObjectService<T> service1, ObjectService<T> service2, Date since)

{

// null checks

if (null == service1)

{

throw new NullPointerException("service1 can't be null");

}

if (null == service2)

{

throw new NullPointerException("service2 can't be null");

}

if (null == since)

{

throw new NullPointerException("since date can't be null");

}

// requesting items

List<Versioned<T>> items1 = service1.findChanged(since);

List<Versioned<T>> items2 = service2.findChanged(since);

// null checks again

if (null == items1)

{

throw new NullPointerException("modList1 is null");

}

if (null == items2)

{

throw new NullPointerException("modList2 is null");

}

// calculating transferring lists

List<Versioned<T>> newer1 = new ArrayList<Versioned<T>>();

List<Versioned<T>> newer2 = new ArrayList<Versioned<T>>();

List<Versioned<T>> only1 = new ArrayList<Versioned<T>>();

List<Versioned<T>> only2 = new ArrayList<Versioned<T>>();

getOverLists(items1, items2, newer1, newer2, only1, only2);

// debug

/\*

\* if (BuildConfig.DEBUG) { Log.v(TAG, "1 --> 2 : total count: " +

\* String.valueOf(over1.size())); for (int i = 0; i < over1.size(); i++) { Log.v(TAG,

\* "1 --> 2 : " + String.valueOf(over1.get(i))); } Log.v(TAG, "2 --> 1 : total count: " +

\* String.valueOf(over2.size())); for (int i = 0; i < over2.size(); i++) { Log.v(TAG,

\* "2 --> 1 : " + String.valueOf(over2.get(i))); } }

\*/

// checking items with are only partially presented

for (Versioned<T> item1 : only1)

{

Versioned<T> item2 = service2.findById(item1.getId());

if ((item2 == null) || (item2.getVersion() < item1.getVersion()))

{

newer1.add(item1);

}

else if (item2.getVersion() > item1.getVersion())

{

newer2.add(item2);

}

}

for (Versioned<T> item2 : only2)

{

Versioned<T> item1 = service1.findById(item2.getId());

if ((item1 == null) || (item1.getVersion() < item2.getVersion()))

{

newer2.add(item2);

}

else if (item1.getVersion() > item2.getVersion())

{

newer1.add(item1);

}

}

// transfer

// THINK: divide into small groups?

service1.save(newer2);

service2.save(newer1);

// Result is number of transferred records

return newer1.size() + newer2.size();

}

}