Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт
Информатики
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
__ ___ А. И. Рубан
подпись
« _____» _____ 2016г.

# ДИПЛОМНЫЙ ПРОЕКТ

230105.65 - «Программное обеспечение вычислительной техники и автоматизированных систем»
код и наименование специальности

«Программный компонент для организации и синхронизации электронных контактов»
тема

Пояснительная записка

Руководитель _____ А.С. Кузнецов
подпись, дата инициалы, фамилия

Нормоконтролер _____ О. А. Антамошкин
подпись, дата инициалы, фамилия

Выпускник ЗКИ 10-05 031016575 _____ В.В. Колосовский
номер группы номер зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2016

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт
Информатики
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ А. И. Рубан
подпись
« _____» _____ 2016г.

**ЗАДАНИЕ**
**НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ**
**в форме дипломного проекта**

Студенту Колосовскому Владиславу Викторовичу.

Группа ЗКИ10-05 Направление (специальность) 230105.65, Программное обеспечение вычислительной техники и автоматизированных систем.

Тема выпускной квалификационной работы: «Программный компонент для организации и синхронизации электронных контактов».

Утверждена приказом по университету № 4202/с от 28.03.16.

Руководитель ВКР А.С. Кузнецов, доцент кафедры «Информатика», канд. техн. наук.

Исходные данные для ВКР: спроектировать и разработать программный компонент для организации и синхронизации электронных контактов.

Перечень разделов ВКР:

- Введение;
- Постановка задачи;
- Архитектура и проектирование компонента;
- Тестирование компонента и устранение проблем;
- Руководство программиста;
- Заключение.

Перечень графического или иллюстративного материала с указанием основных чертежей, плакатов, слайдов: презентационные слайды Impress.

Руководитель ВКР _____ А.С. Кузнецов
(подпись)

Задание принял к исполнению _____ В.В. Колосовский
(подпись)

« ___ » _____ 2016 г.

# АННОТАЦИЯ

Выпускная квалификационная работа по теме «Программный компонент для организации и синхронизации электронных контактов» содержит 61 страницу текстового документа, включая одно приложение, 9 рисунков, 5 таблиц и 18 библиографических источников.

Целью работы являлась разработка компонента для организации и синхронизации электронных контактов. Компонент позволяет хранить и обрабатывать адресные книги пользователей, а также синхронизировать их с внешними источниками.

В дипломный проект входит введение, 4 раздела и заключение.

Во введении определяется необходимость реализации и основные задачи проекта.

В первом разделе описывается анализ предметной области, выявление проблемы и постановка задачи для программного решения.

Во втором разделе описан процесс проектирования и архитектура решения.

В третьем разделе описаны проблемы, выявленные на этапе тестирования системы в условиях, приближенных к производственным.

Четвертый раздел представляет собой руководство для программиста по внедрению компонента.

Заключение посвящено подведению итогов по всей проделанной работе.

## СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Для современного человека важную роль играют его личные и деловые контакты. В информационную эпоху возможности для быстрой связи обширны: телефонные линии, сотовая связь, электронная почта, социальные сети, средства обмена мгновенными сообщениями - все это позволяет поддерживать контакты с множеством людей. Для многих профессий (предприниматели, торговые представители, менеджеры, фрилансеры) количество таких контактов может достигать сотен и тысяч и вопрос об эффективной организации подобной информации встает особенно остро. Личные контакты (семья, родные, друзья) также имеют большое значение и часто очень важно иметь под рукой актуальный номер телефона человека или его домашний адрес.

Для решения этих задач существует множество средств:

- бумажные телефонные и адресные книги;
- электронные адресные книги на мобильных устройствах и персональных компьютерах;
- облачные средства синхронизации адресных книг.

Безусловно, средства на основе интернет-технологий являются самыми перспективными, так как инфраструктура всемирной сети хорошая развита, позволяет передавать большие объемы данных с высокой скоростью и надежностью. Благодаря беспроводным сетям, мобильный интернет доступен практически везде.

Расцвет интернет технологий привел к появлению разнообразия средств для облачного хранения своих контактов, а также и непосредственного общения с другими людьми. Являясь безусловным достижением человечества, данная ситуация привела к тому что люди одновременно используют различные средства (веб-сайты, сервисы и т.п.) для организации своих контактов. В большинстве случаев это популярные почтовые сервисы, такие *Gmail* или *Yahoo*. Это связано с тем что те или иные средства более удобны в разных ситуациях, или для связей с разными людьми.

Данная проблема наиболее актуальна в корпоративной среде, где эффективный доступ к нужным контактам в нужный момент может напрямую влиять на эффективность деятельности - удачная сделка, новый клиент и так далее.

Таким образом, встает вопрос о наиболее эффективном подходе к организации контактов. Возможны следующие варианты решения:

- отказаться от всех средств, кроме одного (обременяет пользователя выбором);
- использовать различные средства организации данных, но осуществлять импорт-экспорт вручную;
- использовать готовые решения с функцией синхронизации контактов.

В данной работе далее будут рассмотрены решения, позволяющие использовать несколько адресных книг в режиме автоматической синхронизации. Будет обоснована необходимость разработки собственного решения данной задачи.

Наличие множества систем организации контактов ставит перед нами задачу об обмене этой информацией между системами. Практически в любых электронных адресных книгах доступна функция импорта или экспорта данных из файла. Однако такой подход требует от пользователя совершать одну и ту же процедуру (экспорт из системы А, импорт в систему Б) каждый раз после появления новых контактов.

На практике это приводит к тому, что пользователь скорее откажется от использования остальных систем в пользу одной, что ограничит мобильность его адресной книги. В случае если пользователь решит продолжить использование нескольких систем, актуальность информации со временем будет падать.

Таким образом, очевидна необходимость автоматизации данного процесса. Кроме простого добавления контактов, процедура переноса должна обрабатывать следующие ситуации:

– контакт уже существует в системе Б, необходимо его актуализировать по определенным правилам;
– контакт был удален из системы А, нужно принять решение о его удалении из системы Б.

То есть на практике внешне простая задача синхронизации данных является не тривиальной и требует более тщательного анализа конкретных вариантов использования и интеллектуального подхода к обработке измененных данных.

Данную задачу может решить программное средство синхронизации, реализованное на том или ином языке программирования, с использованием определенной среды разработки и стека технологий.

Конечной целью данной работы является разработка готового программного компонента, который позволит эффективно решать поставленные задачи. Важными шагами к разработке являются:

– анализ предметной области, обоснование актуальности разработки и постановка задачи;
– выявление бизнес-моделей и процессов;
– обоснование проектных решений и проектирование целостного компонента;
– разработка и тестирование программного решения;
– написание документации, в том числе руководства программиста.

# 1 Постановка задачи

## 1.1 Анализ существующих решений

Существует множество решений для хранения контактов как отдельных записей - на компьютере, в "облаке" (*Gmail, Yahoo, Outlook*), на мобильном устройстве (*iOS, Android, Windows Phone*). Для некоторых из них существует возможность синхронизации с другими источниками данных.

Примеры таких средств:

- Адресная книга *Android* позволяет синхронизировать контакты с учетной записью *Gmail*, а также *WhatsApp* и прочих приложений.
- Адресная книга *iOS* позволяет в качестве источников для синхронизации выбирать *Gmail, Yahoo, Outlook* (*Hotmail*), *iCloud*, а так же любой сервер, реализующий протокол CardDAV (передача и синхронизация контактов).
- Почти во всех системах электронных адресных книг есть функция ручного импорта данных из файла - CSV, VCard.

Рассмотрим несколько решений для облачного хранения контактов с функцией синхронизации данных. Сравнительный анализ таких решений представлен в таблице 1. Для получения сведений о представленных решениях были использованы ресурсы [1-4], а также отзывы пользователей в сети.

Таблица 1 - Сравнительный анализ облачных решений

| Название | Преимущества | Недостатки |
|---|---|---|
| Outlook (Hotmail) | Интеграция с почтовым сервисом;<br>Интеграция с одноименным приложением для ПК или мобильного устройства;<br>Бесплатность. | Ограниченный набор хранимых данных контактов;<br>Плохо поддерживаемый API с ограниченным функционалом. |
| Yahoo | Интеграция с почтовым сервисом;<br>Более обширный набор возможных данных о контакте;<br>Эффективный API для синхронизации с широким функционалом: REST и YQL;<br>Бесплатность. | Устаревшая документация API;<br>Изменения в API приводят к неработоспособности клиентских приложений;<br>Периодические сбои в работе сервиса. |
| Gmail | Интеграция с почтовым сервисом;<br>Расширенные возможности по хранению данных: пользовательские категории данных, неограниченное число элементов, формы для заполнения структурированных элементов (адреса);<br>Эффективный API для синхронизации на основе XML с поддержкой пакетной обработки данных. | Ненадежная работа сервиса;<br>Недостаточно интуитивный интерфейс;<br>Недостаточная "интеллектуальность" (автоматически могут создаваться множество "лишних" контактов, которые затем необходимо объединять вручную). |
| FullContact | Двунаправленная синхронизация данных о контактах одновременно с множеством сервисов: почтовые сервисы, системы обмена мгновенными сообщениями, социальные сети;<br>Интеллектуальное объединение дублирующих данных;<br>Сопутствующие мобильные приложения синхронизируют адресную книгу приложения с облаком, а также имеют расширенные возможности, такие как сканирование и обработка бумажных визитных карточек. | Проприетарная платформа;<br>Платна для полноценного использования;<br>Данные хранятся в неконтролируемой инфраструктуре. |

Анализ и тестирование данных средств показал ряд особенностей, нежелательных при использовании их в качестве основных рабочих сервисов в корпоративной среде:

– ограниченные структуры данных (поддержка пользовательских полей или категорий, ограничение на максимальное количество контактных данных);
– неполная и неточная синхронизация контактной информации (потеря или неточности при переносе категорий контактных данных, невозможность обновления изменившихся контактов);
– хранение потенциально конфиденциальных данных на неконтролируемой инфраструктуре, что может быть актуально при наличии высоких требований к информационной безопасности (защита сведений от несанкционированного доступа, обеспечение бесперебойного доступа к данным);
– интегрированные решения, которые не позволяют организовать тесное взаимодействие с существующими или разрабатываемыми корпоративными информационными системами.

### 1.2 Цель и задачи разрабатываемого решения

Цель данной работы - спроектировать, разработать и внедрить программный компонент для организации и синхронизации электронных контактов, который будет отвечать требованиям необходимой эффективности и функциональности в рамках использования его как части корпоративной информационной системы.

Выделим основные требования к задачам, решаемым компонентом:
– эффективная подсистема хранения и обработки данных, способная обрабатывать миллионы записей, хранить адресные книги множества пользователей;
– функциональность по синхронизации данных на основе массивов контактов, полученных из любых внешних источников;

– модуль активной синхронизации данных с несколькими популярными почтовыми сервисами, который должен быть выполнен с учетом требований к отказоустойчивости (обработка непредвиденных изменений в работе сторонних API, сбоев, получения данных с нарушенной внутренней согласованностью);
– решение должно быть реализовано в виде независимого программного компонента, который можно будет встроить в существующую корпоративную информационную систему;
– программный код должен быть покрыт автоматическими тестами, отлажен и устойчив к различным условиям использования (высокие нагрузки на запись и чтение, различные информационные системы и т.д.);
– компонент должен работать эффективно и надежно, при условии полного и точного соблюдения руководства программиста по внедрению.

Первым шагом к проектированию такой системы является определение тех структур данных, которые подлежат обработке.

### 1.3 Анализ моделей данных

Необходимо спроектировать такую модель данных о контакте, которая позволит хранить максимальное количество информации о каждом человеке или организации в четко структурированном виде.

Объектом автоматизации является электронная адресная книга пользователя, которая представляет собой множество записей с информацией о личных или деловых контактах.

Структура записи о контакте:

– ID пользователя
– Категория контакта (Личный, Деловой, Общий)
– ФИО

- Название организации
- Должность
- Номера телефонов (массив объектов):
  - Категория (домашний, личный, факс и т.п)
  - Номер
- Электронные почтовые адреса:
  - Категория
  - E-Mail Адрес
- Физические адреса:
  - Категория
  - Улица, дом, квартира/офис
  - Город
  - Штат, Регион
  - Почтовый индекс
  - Страна
- Заметки

Номера телефонов, электронные почтовые адреса и физические адреса впредь будем называть элементами контактных данных. Эти элементы имеют общее поле (категорию) и предполагают схожую логику их обработки (в виде массива объектов).

# 2 Архитектура и проектирование компонента

## 2.1 Общие архитектурные решения

### 2.1.1 Выбор стека технологий

В качестве основной технологии при разработке компонента был выбран язык PHP, являющийся слабо-типизированным динамическим интерпретируемым языком высокого уровня [5]. PHP обладает следующими преимуществами:

- широкая поддержка объектно-ориентированного программирования позволяет применять лучшие практики в области программной инженерии;
- высокая скорость разработки, которая достигается благодаря динамической природе технологии;
- огромное количество готовых библиотек позволяет существенно сократить время разработки программного прототипа;
- популярная библиотека *PHPUnit* позволяет с легкостью создавать юнит-тесты, а её широкая поддержка в интегрированных средах разработки (*NetBeans, PHPStorm*) - быстро выполнять их;
- удобный и эффективный менеджер пакетов *Composer*;
- больше половины веб-сайтов в интернете используют PHP в качестве основного языка; как следствие, практически любой сервер, хостинг или облачная платформа имеют поддержку PHP-приложений.

Было решено создавать компонент с учетом всевозможных условий функционирования приложения, поэтому выбор того или иного веб-сервера (*Apache, nginx, IIS*) или режима его интеграции не играет роли.

### 2.1.2 Структура модулей компонента

Для создания компонента была выбрана архитектура, при которой обработка данных выделена в отдельный модуль (*DAL*). Этот модуль используется в модулях:

– Активной синхронизации (*Sync*), при которой с определенной периодичностью и по определенным правилам опрашивается сторонний источник данных (веб-сервис);
– Пассивной интеграции (*API*), при которой клиентские приложения сами обращаются к компоненту для синхронизации данных.

Общая структура компонента представлена на рисунке 1. При проектировании компонента были использованы рекомендации из [6-7]. В качестве подхода и языка моделирования был выбран UML [8-9].

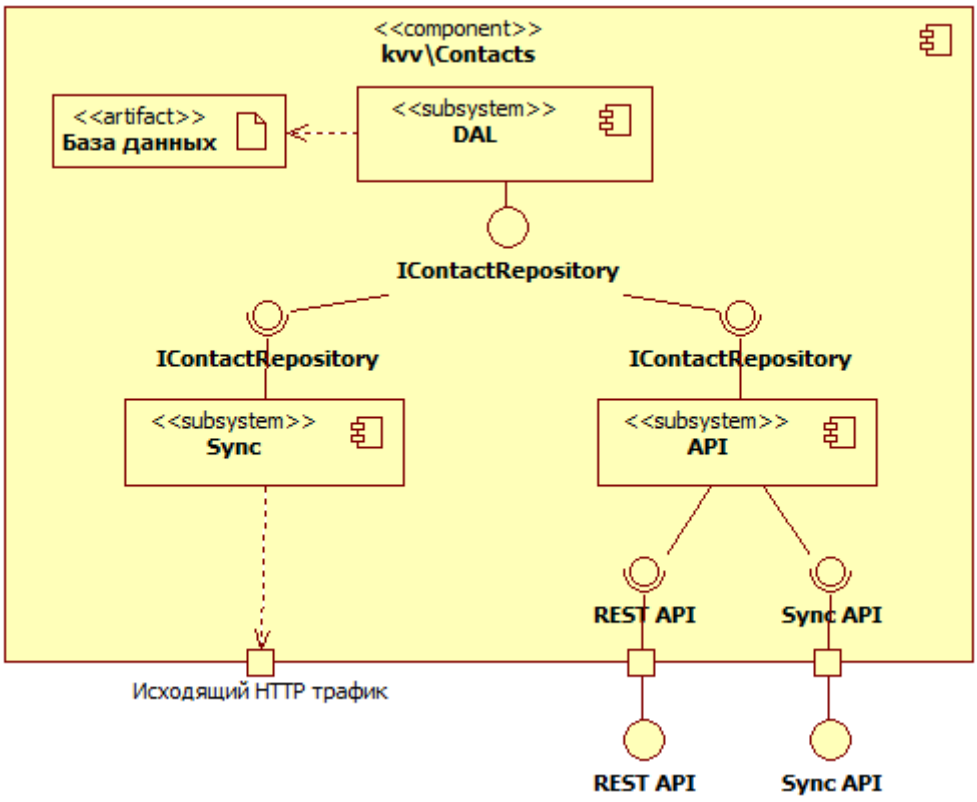



Рисунок 1 - Диаграмма внутренней структуры компонента

Кратко рассмотрим DAL - слой доступа к данным, как наиболее важный модуль - "опору" всего компонента. Логическая внутренняя архитектура данного модуля представлена на рисунке 2Рисунок 2.

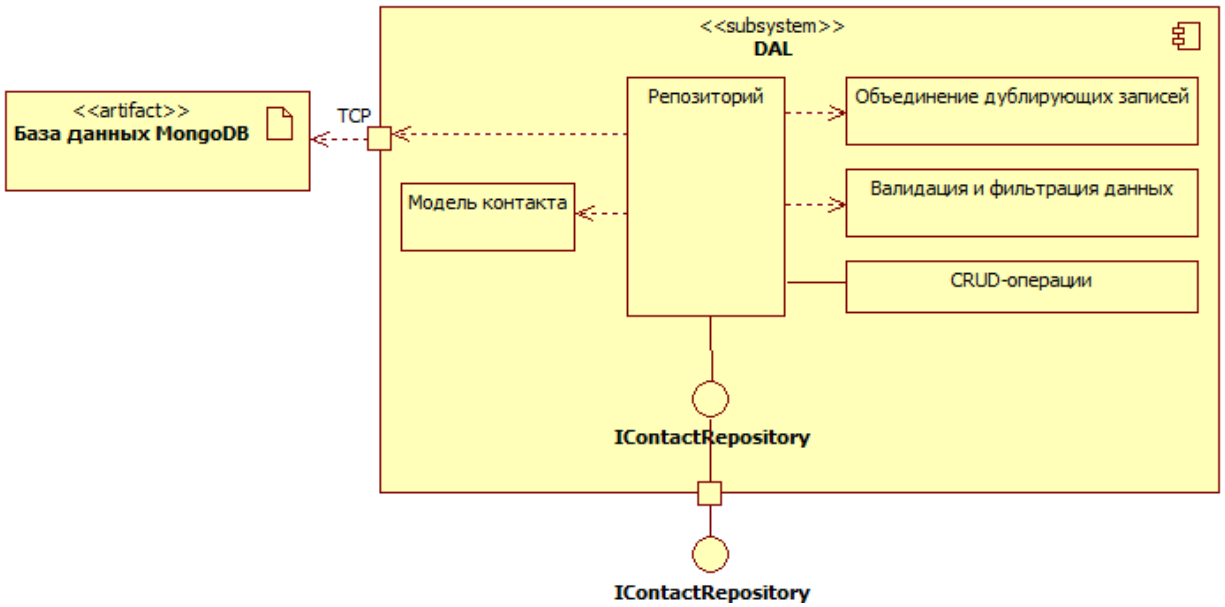



Рисунок 2 - Диаграмма внутренней структуры модуля DAL

Все контакты для различных адресных книг хранятся в базе данных MongoDB. За доступ к данным отвечает репозиторий - набор классов, задачей которых является выборка и модификация данных. При этом используется *Модель контакта* - класс, представляющий собой объектно-ориентированное отображение хранимой информации о контакте.

Представленная архитектура дает возможность разработать универсальный компонент, который может быть использован как относительно самостоятельно, так и в составе монолитной системы.

Для более детального проектирования каждого модуля были использованы некоторые шаблоны из [10]. Результаты проектирования представлены в подразделах 2.2-2.4.

### 2.1.3 Место компонента в архитектуре приложения

На рисунке 3Рисунок 3 представлен пример архитектуры готового приложения по организации и синхронизации электронных контактов в случае тесной интеграции компонента как части системы.

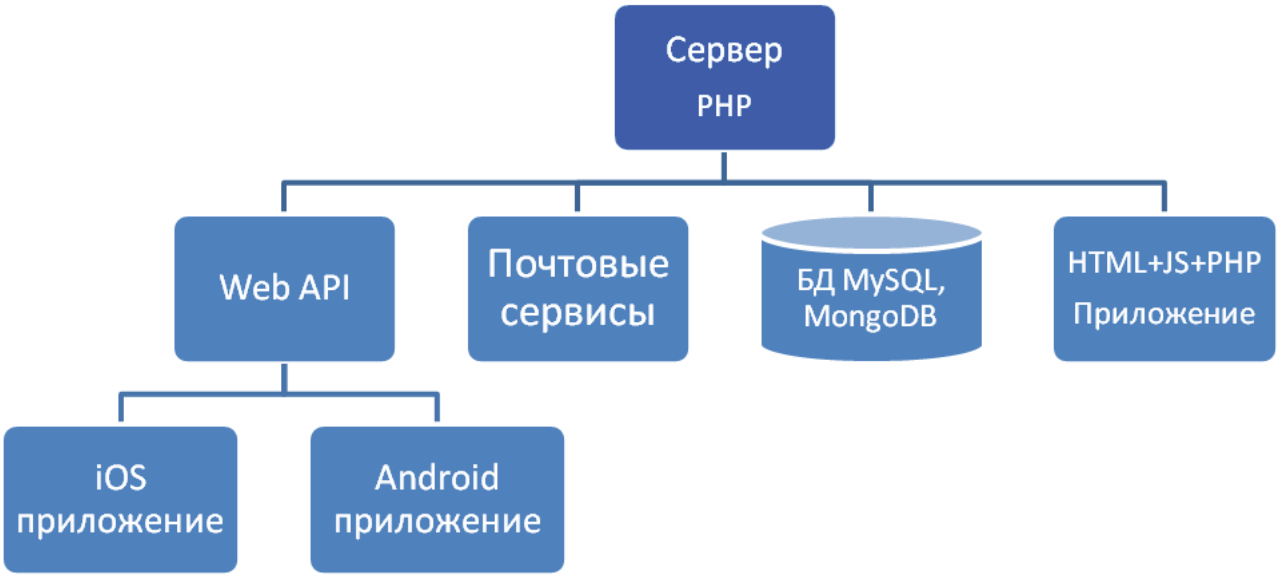



Рисунок 3 - Пример архитектуры приложения

Данное приложение использует четырехзвенную архитектуру:

1. За хранение и обработку данных учетных записей пользователей, а также данных о контактах и связях между ними, отвечают две СУБД: реляционная MySQL и нереляционная MongoDB.
2. Описание моделей данных, логика по поддержанию их целостности, логика всех возможных транзакций представлены в слое доменной модели. Данный слой связан только с СУБД и в качестве интерфейса с внешней средой использует публичные классы и методы, которые принимают или отдают данные и обрабатывают команды.
3. Для взаимодействия с внешней средой существует слой представления, реализованный в виде Web API на основе технологии JSON-REST. Данная технология является наиболее популярной и простой для и по-

зволяет использовать приложение в качестве "бекенда" (источника данных) для других приложений.

4. Различные клиентские приложения используют удобный пользовательский интерфейс и предназначены для непосредственного взаимодействия с конечным пользователем. Они используют Web API в качестве источника данных.

В данной архитектуре разрабатываемый компонент функционирует на первом (база данных в MongoDB) и втором (классы и сервисы PHP) уровнях. Взаимодействие с другими компонентами осуществляется посредством публичных интерфейсов классов. При этом, для использования компонента нет необходимости в модификации его кода. Это достигается благодаря соблюдению принципов SOLID [11]:

- *Single Responsibility* (принцип единственной ответственности) - каждый класс или модуль выполняет только одну, четко сформулированную задачу;
- *Open/Closed Principle* (принцип открытости/закрытости) - сущности должны быть открыты для расширения, но закрыты для модификации;
- *Liskov Substitution* (принцип подстановки Лисков) - экземпляры класса могут быть заменены экземплярами подкласса, при этом гарантируется корректность работы программы;
- *Interface Segregation* (принцип сегрегации интерфейсов) - для каждого варианта использования сущности должен быть отдельный интерфейс (другими словами, лучше много минимальных интерфейсов, чем один громоздкий);
- *Dependency Inversion* (принцип инверсии зависимостей) - классы должны зависеть от абстракций, а не от конкретных классов.

### 2.1.4 Выбор СУБД

Контакты представляют собой структурированные документы (содержат массивы объектов - телефонные номера с категориями, адреса, состоящие из компонентов и так далее). Ожидаемое количество контактов на одну адресную книгу - от 100 до 5000 записей. Разрабатываемая система должна поддерживать информацию большого количества пользователей - от тысяч до десятков тысяч.

Так как каждый контакт может содержать до десятка и более агрегированных записей, несложно подсчитать что количество таких записей в системе может достигает сотен миллионов. В случае использования реляционной модели и реляционных СУБД, это означает хранение всех этих записей в отдельных таблицах. Для выборки данных о контакте, потребуется либо производить операцию соединения, которая очень требовательна к ресурсам, либо получать каждый компонент документа отдельными SQL-запросами.

Безусловно, при грамотно спроектированной схеме реляционной БД и эффективно построенных SQL-запросах можно добиться приемлемой производительности такой системы. Однако более подходит для данной задачи популярный в последнее время, так называемый "*No-SQL*" подход.

Преимущества NoSQL:

1) Высокая скорость выборки одной записей по первичному ключу, достигается благодаря отказу от использования "человеческого" языка SQL. Для выполнения каждого такого запроса необходимо осуществить лексический и синтаксический разбор выражения, затем происходит оптимизация запроса. Всех этих шагов можно избежать, отказавшись от SQL.

2) Отказ от реляционной модели данных позволяет хранить структурированные сущности в виде готовых к использованию документов, в "дружественной" форме для объектно-ориентированных языков. Например, при выборке одного контакта, он уже содержит все принадлежащие ему номера телефонов, адреса (массивы или списки объектов) и дополнительную информацию.

3) Отказ от использования жесткой схемы позволяет гибко расширять хранимые наборы данных, без необходимости дорогостоящих операций по изменению схемы (ALTER TABLE).

Недостатки NoSQL:

1) Отказ от привычного и универсального SQL требует получения дополнительных навыков для работы с каждой конкретной СУБД для построения запросов.

2) Программные компоненты для работы с такими СУБД не всегда доступны, так как все актуальные системы на данный момент еще новы (относительно традиционных СУБД, таких как MSSQL, Oracle, MySQL).

3) Отсутствие схемы требует дополнительных затрат на поддержание целостности и согласованности хранимых данных на уровне приложения. Кроме того, необходим иной подход к работе с данными, полученными из Schema-less коллекций. Например, нельзя всегда полагаться на тип того или иного поля и при необходимости осуществлять валидацию и/или фильтрацию данных.

*MongoDB* [12] был выбран в качестве СУБД по следующим причинам:

- Хорошая поддержка разработчиками - частые обновления, четкие направления развития технологии;
- Доступная и актуальная документация;
- Популярная технология, а это значит обширная информация по примерам использования и решению возникающих проблем;
- Широкие возможности по выборке данных, приближающие по функциональности к традиционным СУБД:
  - Отбор последовательностей записей из коллекций с использованием различных условий, в том числе по вложенным объектам;
  - Сортировка данных;
  - Всевозможные средства агрегации данных;
  - Постраничная выборка.

– Наличие "в комплекте" таких функций как шардинг и репликации, предоставляют широкие возможности по горизонтальному масштабированию в случае, если система окажется популярной, а нагрузка на базу данных - достаточно высокой.

Для анализа возможностей различных типов СУБД была использована информация из [1312].

## 2.2 Слой доступа к данным

### 2.2.1 База данных

Для компонента была создана база данных в СУБД MongoDB. База содержит всего одну коллекцию - "contacts". Коллекции не имеют схемы, как в реляционных таблицах, однако есть возможность задавать индексы для ускорения запросов. Были созданы следующие индексы:

– *userID* - идентификатор пользователя адресной книги;
– *categoryID* - категория приватности контакта;
– *groups.groupID* - идентификатор группы во вложенном массиве;
– *created* - время создания;
– *updated* - время последнего изменения;
– *readOnly* - флаг, определяющий возможность редактирования контакта;
– *photo* - имя файла фотографии;
– *firstName* - имя контакта;
– *lastName* - фамилия;
– *companyName* - компания;
– *phones.value* - номер телефона;
– *emails.value* - адрес электронной почты.

### 2.2.2 Репозиторий данных и его основные функции

В качестве репозитория для контактов выступает класс *MongoContactRepository*. Данный класс предназначен для извлечения контактных данных из коллекции MongoDB, а также для изменения адресных книг. Репозиторий выполняет следующие функции.

1. Извлечение списка контактов из базы.

Параметры:

– *userID* (число) - идентификатор пользователя-владельца адресной книги;

– *filter* (массив) - дополнительный фильтр для поиска;

– *paging* (массив) - информация для постраничной разбивки результата.

Возвращает массив объектов класса *ContactEntity*.

2. Извлечение контакта по заданному идентификатору.

Параметры: id (строка) - идентификатор записи в коллекции MongoDB.

Возвращает объект класса *ContactEntity*, заполненный данными из соответствующей записи в коллекции.

3. Сохранение контакта.

Параметры: объект класса *ContactEntity*.

Сохраняет новый или существующий контакт в базу данных. При этом срабатывают алгоритмы обеспечения целостности и объединения дубликатов. Данные алгоритмы будут рассмотрены в следующих подразделах.

### 2.2.3 Модель контакта и обеспечение целостности данных

Для работы с контактами был создан класс *ContactEntity* (в дальнейшем просто Модель). Он выполняет 2 основные функции:

1. Единое представление одного контакта в компоненте. На всех этапах работы с контактами, будь то получение записи из базы данных или его обработка, будет использован один класс. Данный подход позволяет избежать лишнего, дублирующего кода.

Для записи контакта были определены поля (свойства класса), позволяющие хранить всю информацию о контакте, определенную на этапе анализа предметной области. Поля контакта представлены в таблице 2 (через точку указаны поля вложенных объектов, звездочкой помечены обязательные поля).

Таблица 2 – Свойства модели

| **Название** | **Тип** | **Назначение** |
|---|---|---|
| 1 | 2 | 3 |
| id* | MongoId | уникальный идентификатор |
| userID* | целое | идентификатор пользователя адресной книги |
| groups | массив | группы, к которым принадлежит контакт |
| groups.groupID | целое | идентификатор группы |
| categoryID | строка | категория приватности |
| firstName | строка | имя |
| middleName | строка | отчество |
| lastName | строка | фамилия |
| nickName | строка | псевдоним |
| title | строка | титул |
| companyName | строка | название организации |
| photo | строка | имя файла-фотографии |
| phones | массив элементов | номера телефонов |
| phones.category | строка | категория элемента |
| phones.value | строка | значение элемента |
| emails | массив элементов | адреса электронной почты |

Окончание таблицы 2

| 1 | 2 | 3 |
|---|---|---|
| addresses | массив адресов | физические адреса |
| addresses.category | строка | категория адреса |
| addresses.countryID | строка | код страны |
| addresses.city | строка | город |
| addresses.stateID | строка | код штата |
| addresses.zip | строка | почтовый индекс |
| addresses.value | строка | улица, дом, квартира, офис и т.д. |
| notes | строка | прочие заметки |
| created | целое | UNIX-timestamp времени создания |
| updated | целое | UNIX-timestamp времени последнего изменения |
| readOnly | булевое | если истина - контакт недоступен для изменения |
| serviceSync | массив | информация о синхронизации с почтовыми службами |
| serviceSync.service | строка | тип сервиса: yahoo, hotmail, gmail |
| serviceSync.* | * | набор прочих полей зависит от реализации конкретного сервиса |
| deviceSync | массив | информация о синхронизации с устройствами |
| deviceSync.type | строка | тип устройства (например, iOS, Android и т.п.) |
| deviceSync.deviceID | строка | идентификатор устройства |
| deviceSync.abID | строка | идентификатор контакта в адресной книге внешнего устройства |

Стоит заметить что обязательными для заполнения являются всего 2 поля - id и userID. Слой хранения допускает пустые значения любых полей, однако сервисные классы, отвечающие за создание, импорт, синхронизацию контактов могут иметь свои требования по минимальному набору полей.

2. Поддержание целостности данных. Поскольку для хранения контактов используется нереляционная СУБД, которая не имеет такого понятия как "схема базы данных", необходимо обеспечивать контроль целостности данных на уровне приложения. Такой контроль происходит внутри данного класса.

Примером типичной проблемы, которая может возникнуть при использовании schema-less данных, может быть поле-идентификатор пользователя

(*userId*), которому принадлежит контакт. Данное поле должно иметь целочисленный тип. Так как программная часть написана на динамически-типизированном языке PHP, в котором числа и строки, содержащие числа, взаимозаменяемы, может возникнуть ситуация, когда в репозиторий будет передан контакт для сохранения в БД, у которого значение поля *userId* - строкового типа. Если не предпринять никаких действий, MongoDB позволит сохранить такую запись, так как отсутствует схема данных со стороны СУБД. Но в отличии от PHP, при поиске записей в БД MongoDB работает более сильная типизация, где целочисленный и строковый типы не взаимозаменяемы. В результате такие записи просто "пропадут", так как не попадут в выборку по значению *userId*.

Во избежание таких ситуаций, необходимо реализовать прослойку фильтрации и валидации данных. Перед любым сохранением записи в MongoDB, она будет подвергнута обработке. При этом возможны два случая:

- ошибка валидации - отсутствуют необходимые данные (такие как *userId*), либо иным путем нарушена целостность записи и не может быть разрешена автоматически - требуется корректировка внешнего приложения;
- неверные типы данных - целостность нарушена, однако есть вся необходимая информация для её восстановления - например строка, переданная в числовое поле, будет интерпретирована и приведена к числовому типу.

Класс модели опирается на специальные сервисные функции для фильтрации и валидации данных. Остановимся более подробно на этих функциях.

### 2.2.4 Валидация и фильтрация данных

Перед сохранением записи, сначала происходит фильтрация данных. Для каждого поля контакта (см. таблицу 2) прописан определенный тип данных и,

возможно, дополнительные ограничения. Выполняются следующие правила преобразования:

- если значение поля строкового типа содержит значение любого другого типа, происходит преобразование в нужный тип средствами PHP (таким образом, числа будут преобразованы строки, содержащие их, а значения *null*, *false* и 0 будут преобразованы в "" - пустую строку);
- если значение поля числового типа содержит значение любого другого типа, происходит преобразование в нужный тип средствами PHP (таким образом, числа помещенные в строку будут преобразованы в сами числа, а *null*, *false* и "" будут преобразованы в 0);
- если тип поля - массив, но содержит значение другого типа (в том числе *null*), ему присваивается пустой массив;
- аналогичным образом обрабатываются все объекты (фактически - ассоциативные массивы) во вложенных массивах;
- если элемент контактных данных некорректен (не является ассоциативным массивом, либо значения полей *category* и/или *value* пусты) - такой элемент исключается из массива;
- схожим путем проверяется массив *groups*, если любой его элемент имеет некорректное значение *groupID* (число, отличное от нуля), такой элемент отбрасывается.

Далее измененный контакт валидируется - проверяется принципиальная возможность хранения такой записи в БД, то есть происходит контроль целостности данных. В валидацию входит:

- проверка на наличие обязательных полей (*userID*);
- проверка по критерию минимального набора данных контакта:
  - как минимум имя или фамилия;
  - как минимум один адрес электронной почты или один номер телефона.

Если одно из условий валидации не выполнено - поднимается исключение. В таком случае программист должен либо отлавливать такое исключение, либо устранить ошибку в коде, которая к нему приводит.

### 2.2.5 Объединение дублирующих записей

Перед добавлением новых записей в систему, репозиторий вызывает специальную функцию, которая предотвращает дублирование записей. Важнейшим вопросом является определение дублирования. Для прототипа компонента было решено остановиться на следующих критериях дублирующей записи:

- Владелец адресной книги совпадает (поле *userID*);
- Имя и фамилия совпадают (без учета регистра или пробелов);
- По меньшей мере один номер телефона или адрес электронной почты совпадают, при этом учитываются только значимые цифры телефона (не учитываются прочие символы, такие как пробелы, скобки), а email-адреса сравниваются без учета регистра и пробелов.

Объединение происходит автоматически и работает следующим образом.

В репозиторий был передан новый контакт для добавления. Происходит поиск аналогичной записи по указанным выше критериям. В случае если таких записей не найдено, просто добавляется новая запись.

В случае если найдена дублирующая запись, происходит слияние - добавление данных переданного контакта в существующий контакт по следующим правилам:

- Для всех "скалярных" полей, таких как имя, фамилия, название компании и т.п. происходит проверка - если у существующей записи поле не заполнено (содержит пустую строку или одни пробелы), то его значение заменяется значением соответствующего поля нового контакта;
- Для всех полей-элементов, таких как адреса и телефоны, происходит добавление всех элементов нового контакта в старый, которые в нем отсутствовали (при проверке на отсутствие элемента работают аналогичные правила как при поиске дублирующих записей, например зна-

чение номера телефона 7-555-555-55-55 будет соответствовать значению 7 (555) 555-55-55 и так далее).

В последнем случае новая запись не добавляется в базу данных.

### 2.2.6 Диаграмма классов слоя доступа к данным

На рисунке 4 представлена диаграмма основных классов слоя доступа к данным. Все представленные классы принадлежат пространству имен DAL.

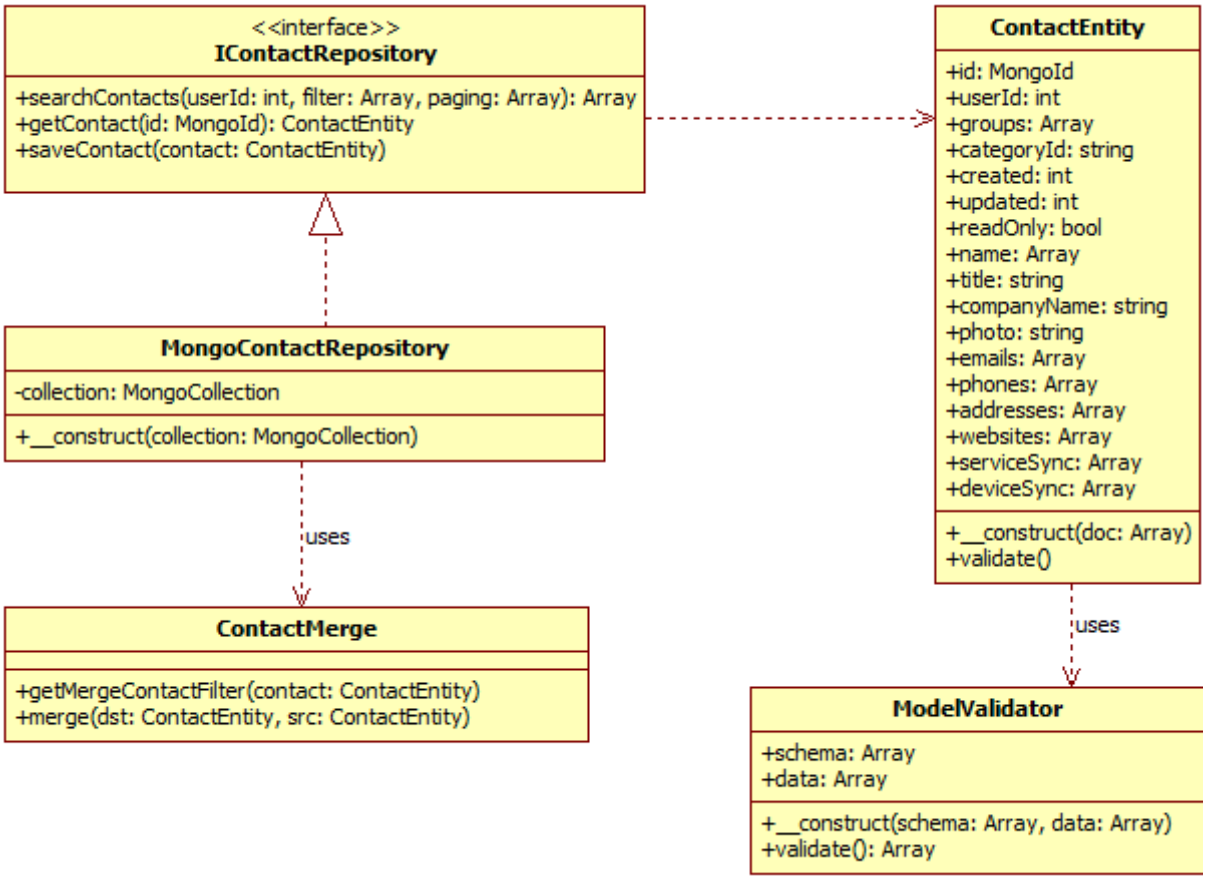



Рисунок 4 - Диаграмма классов DAL

## 2.3 Активная синхронизация

### 2.3.1 Описание задачи

Одним из ключевых модулей разрабатываемого компонента является модуль активной синхронизации, или если быть точным - модуль синхронизации контактов пользователей с почтовыми сервисами. В рамках разрабатываемой версии реализована синхронизация с сервисами *GMail, Yahoo Mail, Hotmail* (он же *Outlook*, далее для простоты будем называть *Hotmail*).

Суть задачи активной синхронизации сводится к тому, чтобы все изменения в адресной книге внутри стороннего сервиса автоматически и с приемлемой задержкой попадали в локальную систему. "Активной" она названа по той причине, что для рассматриваемых сервисов не предусмотрено механизма автоматической передачи обновлений "подписчикам" (например, шаблон Наблюдатель). Но существуют интерфейсы программирования приложении, через которые возможно получать информацию об адресных книгах пользователей в виде, удобном для машинной обработки. Кроме того, эти интерфейсы позволяют изменять адресную книгу. Таким образом возможна двусторонняя синхронизация контактов.

Проанализировав интерфейсы программирования трёх сервисов (по информации из [1-3]), рассмотрим их возможности, представленные в таблице 3.

Таблица 3 - Интеграционные возможности почтовых сервисов

| **Функция** | **GMail** | **Yahoo** | **Hotmail** |
|---|---|---|---|
| Получение полной информации о контактах | да | да | да |
| Получение только измененных данных | да | да | нет |
| Добавление контактов | да | да | да |
| Изменение контактов | да | да | нет |
| Удаление контактов | да | да | нет |
| Обновление данных одним запросом | да | да | нет |
| Неограниченное число контактных данных | да | да | нет |
| Пользовательские категории | да | нет | нет |

### 2.3.2 Интерфейс программирования GMail Contacts

Интерфейс программирования GMail Contacts использует формат XML для получения и передачи данных. Данный формат имеет более богатые возможности по представлению данных в сравнении, например, с JSON. При этом для передачи и для получения данных используется одна и та же схема данных [3].

Рассматриваемый интерфейс построен по архитектуре REST, поэтому не составит труда сориентироваться в доступных для использования методах, опираясь на хорошо известные принципы RESTful.

Рассмотрим используемые в нашем модуле методы GMail Contacts API и их параметры.

1. Получение списка контактов.

Путь: */feeds/contacts/default/full*

Метод: GET.

Формат ответа: XML.

Параметры:

– *start-index* - порядковый номер первой получаемой записи (с 1, целое число);
– *max-results* - максимальное количество возвращаемых записей (целое число);
– *updated-min* - минимальное дата/время последнего обновления запрашиваемых записей (строка, в формате RFC 3339 [14])

2. Получение изображения (аватара) контакта.

Путь: */feeds/photos/media/default/{id}*

Метод: GET

Формат ответа: бинарный файл изображения.

Параметры:

– *{id}* - идентификатор контакта в базе данных Google.

3. Добавление нового контакта.

Путь: */feeds/contacts/default/full*

Метод: POST

Формат запроса: XML-документ с данными контакта.

Формат ответа: XML-документ с описанием результата выполнения операции.

4. Обновление данных о контакте.

Путь: */feeds/contacts/default/full/{id}*

Метод: PUT

Формат запроса: XML-документ с данными контакта.

Формат ответа: XML описание результата выполнения операции.

5. Обновление изображения контакта.

Путь: */feeds/photos/media/default/{id}*

Метод: PUT

Формат запроса: бинарный файл изображения.

Формат ответа: XML описание результата выполнения операции.

6. Удаление контакта

Путь: */feeds/contacts/default/full/{id}*

Метод: DELETE

Формат ответа: XML описание результата выполнения операции.

На рисунке 5 представлена диаграмма последовательности взаимодействия модуля с интерфейсом программирования GMail Contacts.

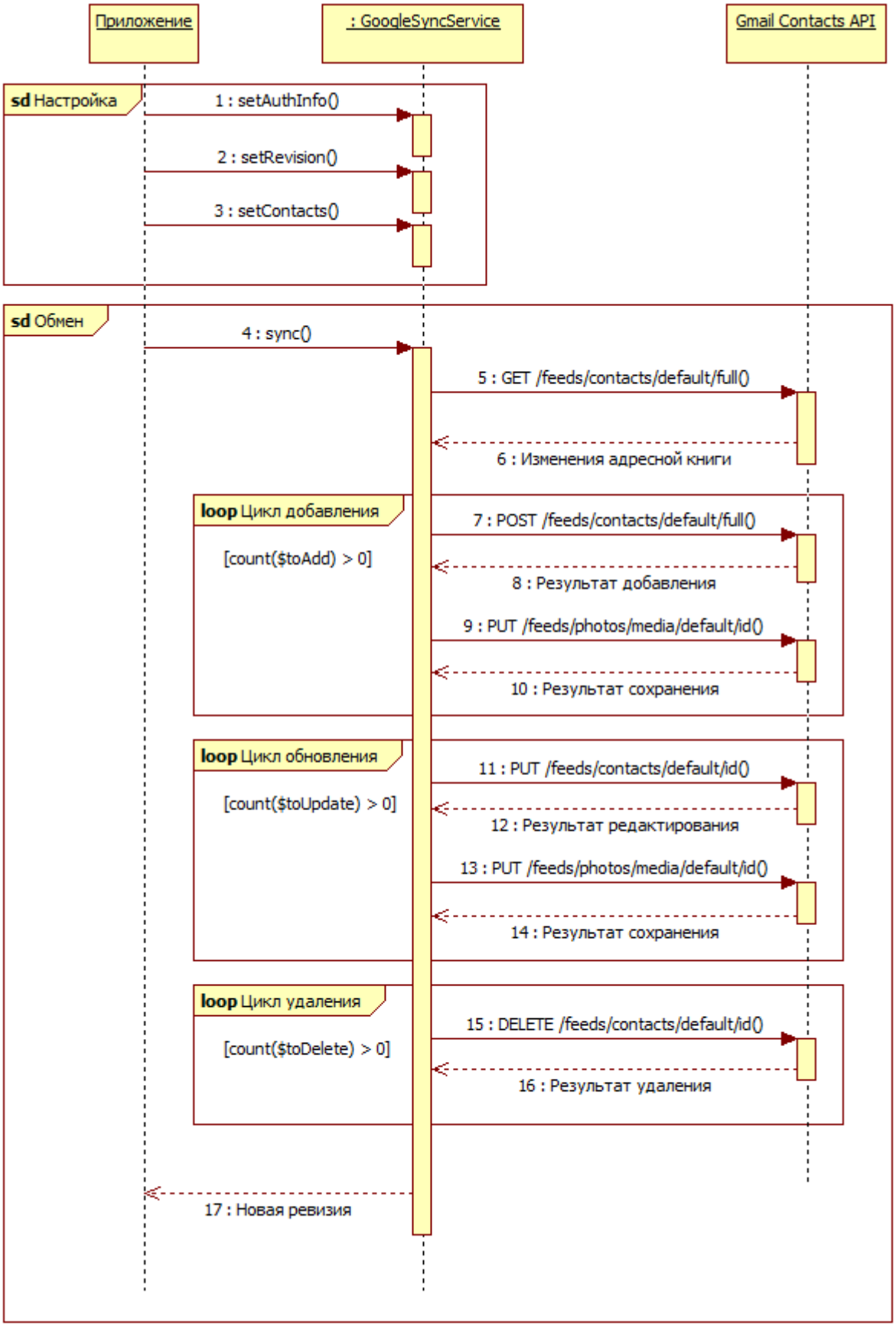



Рисунок 5 - Диаграмма последовательности взаимодействия с GMail

### 2.3.3 Интерфейс программирования Yahoo Contacts

Интерфейс Yahoo Contacts имеет преимущество перед остальными сервисами в наличии специального функционала, который позволяет в 2-3 запроса выполнить полный цикл синхронизации данных. При этом могут выполняться сразу множество операций с контактами, такие как изменение, добавление или удаление [2].

Центральным понятием в механизме синхронизации контактов Yahoo является номер ревизии. После каждого изменения данных на стороне Yahoo, инкрементируется номер ревизии. При каждом получении контактов, клиент передает номер ревизии, который он получал последний раз с сервера, а сервер на основе этой информации возвращает только те изменения, которые произошли с этой ревизии и номер текущей ревизии сервера. Клиент должен внести все изменения на своей стороне и сохранить новый номер в локальном хранилище.

Рассмотрим простой пример.

- На сервере в данный момент ревизия 1, на клиенте 0.
- Клиент запрашивает текущее состояние с сервера, передавая ревизию 0. В ответ он получает полные данные по всем текущим контактам, так как для этого клиента это будет первая синхронизация. Кроме того ему приходит номер ревизии 1, которой он сохраняет в своей базе данных.
- Клиент вносит локальные изменения в адресную книгу, сервер также вносит изменения, увеличивая номер ревизии до 2.
- Клиент снова запрашивает новые данные, ссылаясь на последнюю известную ему ревизию - 1. В ответ ему приходят только те изменения, которые произошли между версиями 1 и 2. Клиент сравнивает изменения на сервере с локальными изменениями, принимает решения касательно возможных конфликтных изменений и сохраняет номер ревизии 2.

– Клиент формирует и передает на сервер запрос на изменение адресной книги, который актуализирует данные на стороне сервера с учетом изменений на клиенте.

Рассмотрим все методы, используемые компонентом (стандартным форматом для данных, передаваемых и получаемых с сервера, является JSON).

1. Получение списка контактов для синхронизации.

Путь: *https://social.yahooapis.com/v1/user/{guid}/contacts*

Метод: GET.

Параметры:

– *{guid}* - идентификатор пользователя в базе данных Yahoo;

– *view* - тип отображения, имеет только одно возможное значение - *sync*, оно означает возвращать данные для синхронизации;

– *rev* - номер ревизии.

Возвращает: перечень изменений, которые необходимо применить на клиенте для актуализации данных с учетом изменений на сервере.

2. Передача данных для синхронизации изменений на сервере.

Путь: *https://social.yahooapis.com/v1/user/{guid}/contacts*

Метод: PUT.

Запрос: перечень изменений, которые необходимо внести на сервере. Представлен в виде объекта *contactsync* с двумя свойствами - *rev* (номер ревизии) и *contacts* (перечень изменений по каждому контакту).

Возвращает: результат внесения изменений на сервере, в том числе идентификаторы вновь созданных контактов, которые следует сохранить на клиенте и использовать при дальнейших запросах

3. Получение данных о контакте.

Путь: *https://social.yahooapis.com/v1/user/{guid}/contact/{id}*

Метод: GET.

Параметры:

*{id}* - идентификатор контакта в базе данных Yahoo.

Возвращает: полные данные о выбранном контакте.

На рисунке 6 представлена диаграмма последовательности взаимодействия модуля с интерфейсом программирования Yahoo Contacts.

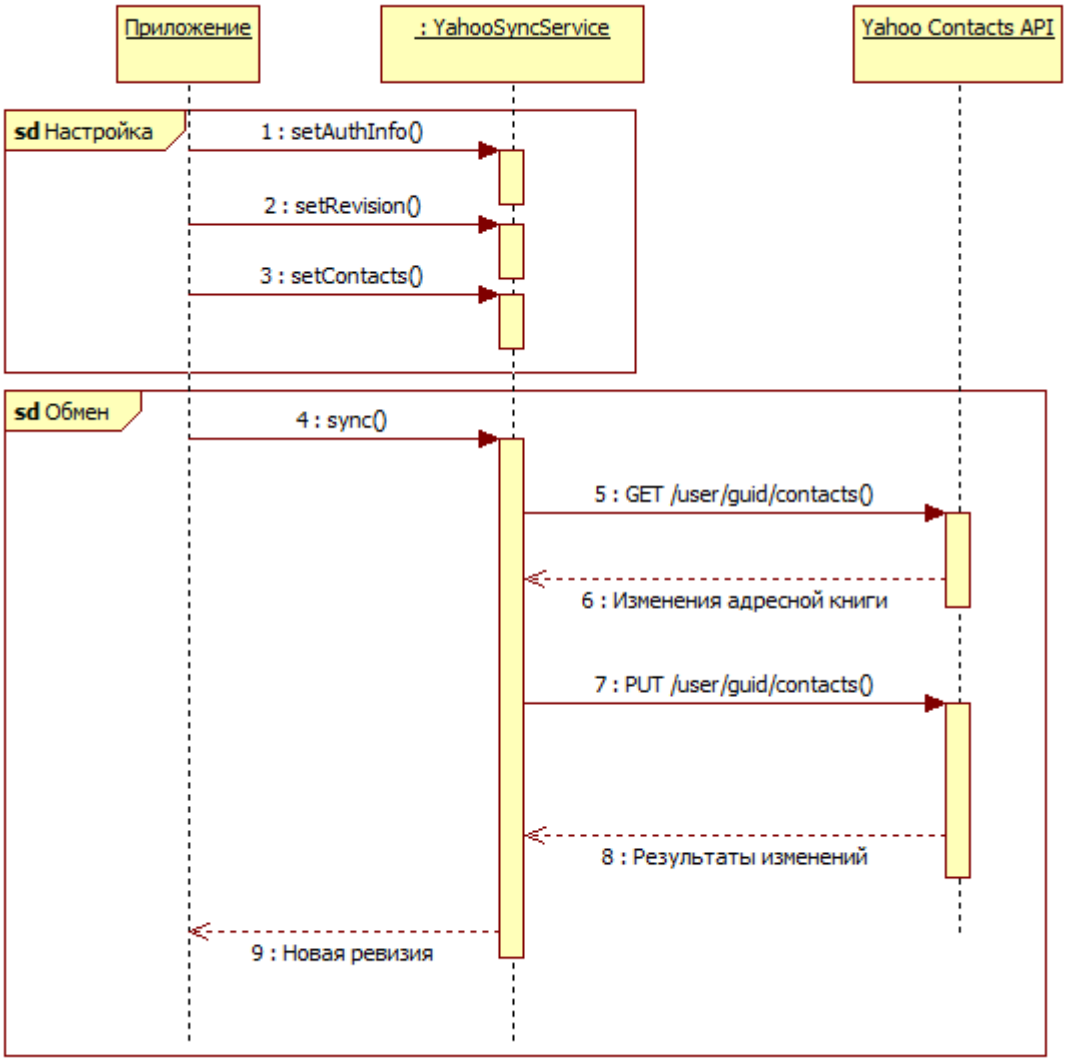



Рисунок 6 - Диаграмма последовательности взаимодействия с Yahoo API

### 2.3.4 Интерфейс программирования Hotmail

Hotmail Contacts API имеет сравнительно простой интерфейс программирования. В качестве основного формата сериализации данных используется JSON [1]. Рассмотрим используемые методы.

1. Получение списка контактов.

Путь: */v5.0/me/contacts*

Метод: GET.

Формат ответа: JSON.

2. Добавление нового контакта.

Путь: */v5.0/me/contacts*

Метод: POST

Формат запроса: JSON-документ с данными контакта.

Формат ответа: JSON-документ с описанием результата выполнения операции.

### 2.3.5 Анализ ограничений интерфейса программирования Hotmail

Интеграционные возможности рассматриваемого интерфейса крайне ограничены. Перечислим ключевые недостатки интерфейса:

1. Для получения контактов существует всего один метод, который возвращает сразу всю адресную книгу пользователя. Нет возможности получения только измененных контактов с какой-то отметки времени или ревизии. Нет возможности фильтрации или постраничного получения данных.

2. Максимальный объем данных о каждом контакте ограничен, что обусловлено особенностями самой подсистемы контактов почтового сервиса. Например, количество физических адресов и адресов электронной почты ограничено тремя, причем по одному каждой из трех фиксированных категорий: *Personal* (личный), *Business* (деловой), *Custom* (пользовательский).

Стоит заметить, что слово Custom фиксировано и может быть только один адрес с такой категорией.

Номера телефонов ограничены следующими категориями: *Personal* (личный), *Business* (деловой), *Mobile* (мобильный), *Home* (домашний), *Custom* (пользовательский).

3. Нет возможности обновлять уже существующие в Hotmail контакты, ровно как и метода для их удаления. Единственный доступный, согласно документации, метод изменения адресной книги - добавление контакта (по одному).

В сложившихся условиях было решено сделать все возможное для наиболее полной синхронизации данных в вышеописанных рамках. Были реализованы следующие компромиссные решения:

1. Вновь добавляемые контакты будут переданы в Hotmail в виде серии вызовов метода создания контакта. Изменения или удаления контактов не будут отражены в почтовом сервисе.

2. Для передачи контактных данных будет применена следующая схема. Для каждой из возможных категорий (3 для адресов, 5 для телефонов) будут выбраны элементы соответствующих типов исходного контакта, имеющие категорию с таким же названием, либо его смысловой "синоним" (например Personal - Home, Business - Work и т.д.). Для категории Custom также будут выбраны все элементы, не соответствующие ни одной из доступных категорий. Если какой-либо категории соответствует более одного элемента - будет выбран первый по счету.

3. При опросе интерфейса программирования Hotmail будут получены все имеющиеся контакты, после чего данный список будет сравниваться с локальным и, таким образом, все изменения на стороне почтового сервиса должны быть переданы обратно в систему.

4. Возможна ситуация, при которой контакт был создан в локальной системе и имел элементов данных больше, чем поддерживает Hotmail. Далее этот контакт был передан в Hotmail, после чего он был изменен пользователем. При следующей синхронизации контакт в локальной системе должен быть обнов-

лен, однако если заменять существующие элементы контактных данных новыми, неизбежна потеря существующей информации. Чтобы этого избежать было принято решение не удалять существующие элементы, а только лишь добавлять новые (дубликаты в данных при этом не допускаются на уровне репозитория).

На рисунке 7 представлена диаграмма последовательности взаимодействия модуля с интерфейсом программирования Hotmail. Данная диаграмма описывает внешнюю сторону алгоритма, полученного в результате анализа описанных выше ограничений и синтеза соответствующих решений.

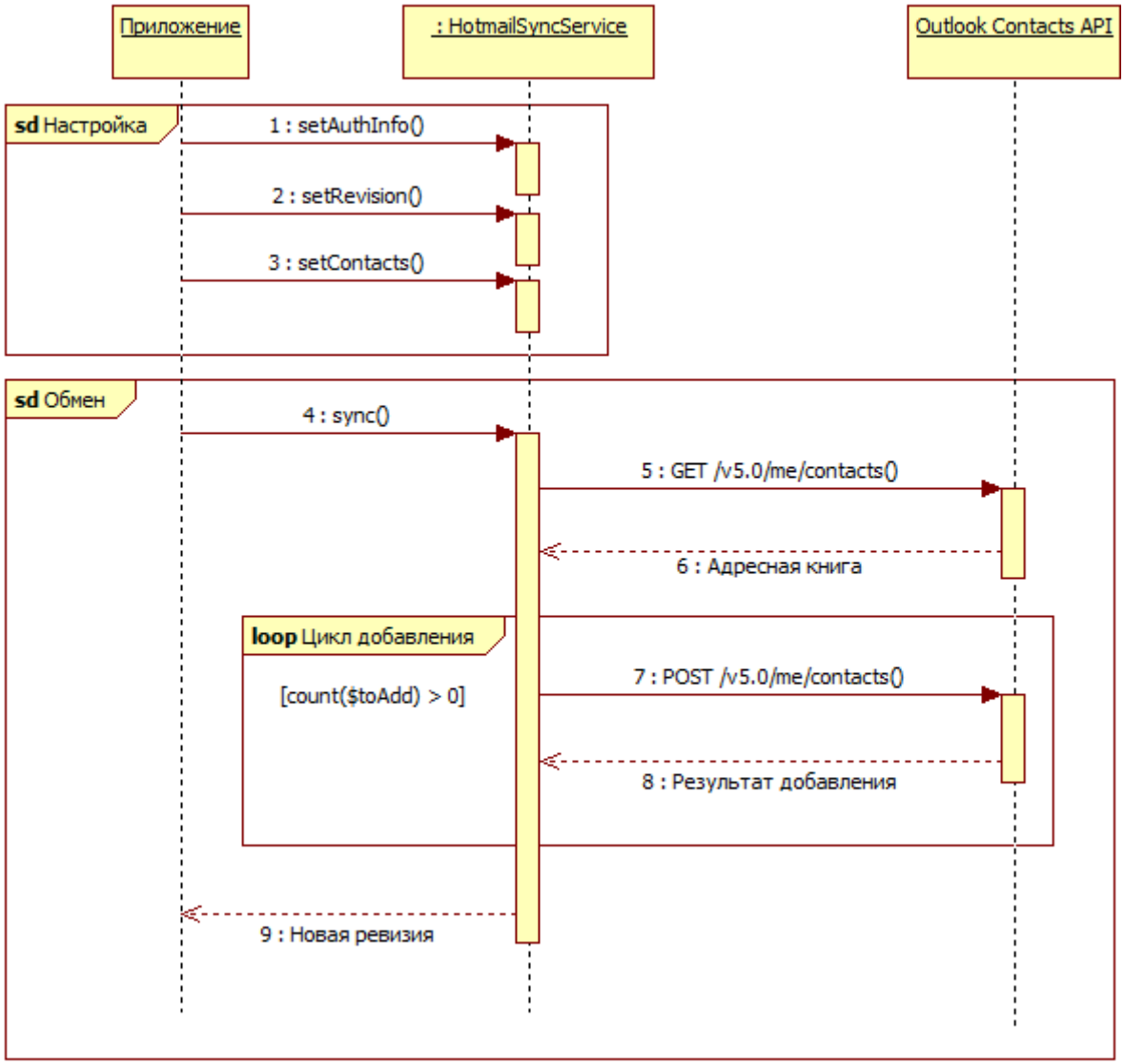



Рисунок 7 - Диаграмма последовательности взаимодействия с Outlook API

### 2.3.6 Диаграмма классов модуля активной синхронизации

На рисунке 8 представлена диаграмма классов модуля активной синхронизации, реализованных в рамках пространства имен *kvv\Contacts\Sync*.

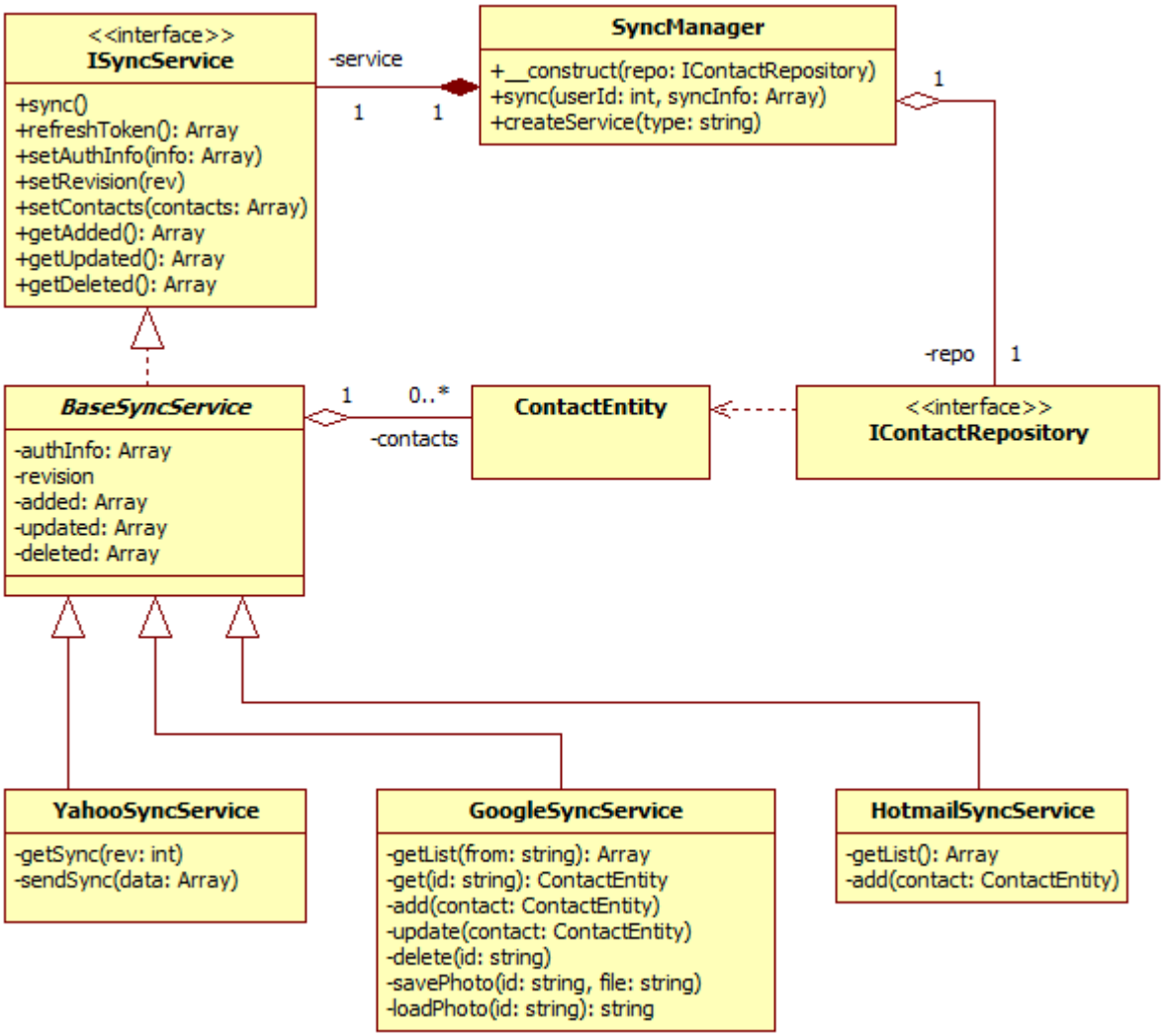



Рисунок 8 - Диаграмма классов Sync

### 2.3.7 Обеспечение отказоустойчивости

Для обеспечения надежной работы процесса синхронизации, при разработке компонента были применены следующие решения:

- Для большинства классов модуля были написаны юнит-тесты, которые гарантируют работоспособность модуля в типичных режимах работы после внесения каких-либо изменений в их код.
- В случае возникновения ошибки при обновлении какого-либо контакта, ошибка отлавливается, логируется и процесс продолжается. Таким образом ошибка, связанная с особенностями одного контакта (например, наличие символа, неподдерживаемого системой) приведет лишь к сбою синхронизации проблемной записи, а не всей адресной книги.
- Применение механизмов фильтрации на уровне доступа к данным позволяет минимизировать возможные проблемы от некорректных данных, получаемых с внешних источников.

Кроме вышеописанных мер, при внедрении компонента в приложение и для обеспечения требуемой отказоустойчивости соблюдать все рекомендации по интеграции (см. раздел 4.3). Это необходимо, поскольку компонент не является самостоятельным программным обеспечением и ошибки в коде приложения могут свести на нет все усилия по обеспечению надежной работы синхронизации.

В процессе пробной эксплуатации компонента были выявлены высокие показатели отказоустойчивости. При различных сценариях работы и длительной синхронизации с почтовыми сервисами серьезных сбоев замечено не было. Таким образом, можно сделать вывод о том, что требования к модулю по надежности работы были выполнены.

## 2.4 Пассивная синхронизация

### 2.4.1 Описание задачи

Задача пассивной синхронизации заключается в построении интерфейса программирования приложений, который позволил бы любым сторонним системам и приложениям (в частности мобильным приложениям, под управлением операционных систем Android или iOS) управлять адресными книгами пользователей, хранимых на центральном сервере.

Для выполнения данной задачи был разработан модуль пассивной синхронизации. Модуль представляет собой набор сервисных классов, обеспечивающих необходимую логику и корректные потоки данных.

В совокупности, модуль пассивной синхронизации предоставляет следующую функциональность:

- получение списка всех контактов пользователя (в том числе удаленных) с возможностями различной фильтрации;
- получение отдельных контактов по идентификатору;
- обновление данных одного или нескольких контактов;
- удаление одного или нескольких контактов по их идентификаторам;
- автоматическое слияние адресной книги в системе с адресной книгой мобильного устройства, посредством передачи последней.

В разделе 4 рассмотрен процесс интеграции данного модуля в API приложения.

### 2.4.2 Диаграмма классов модуля пассивной синхронизации

На рисунке 9 представлена диаграмма классов модуля пассивной синхронизации, реализованного в виде набора сервисных классов в пространстве имен *kvv\Contacts\API*.

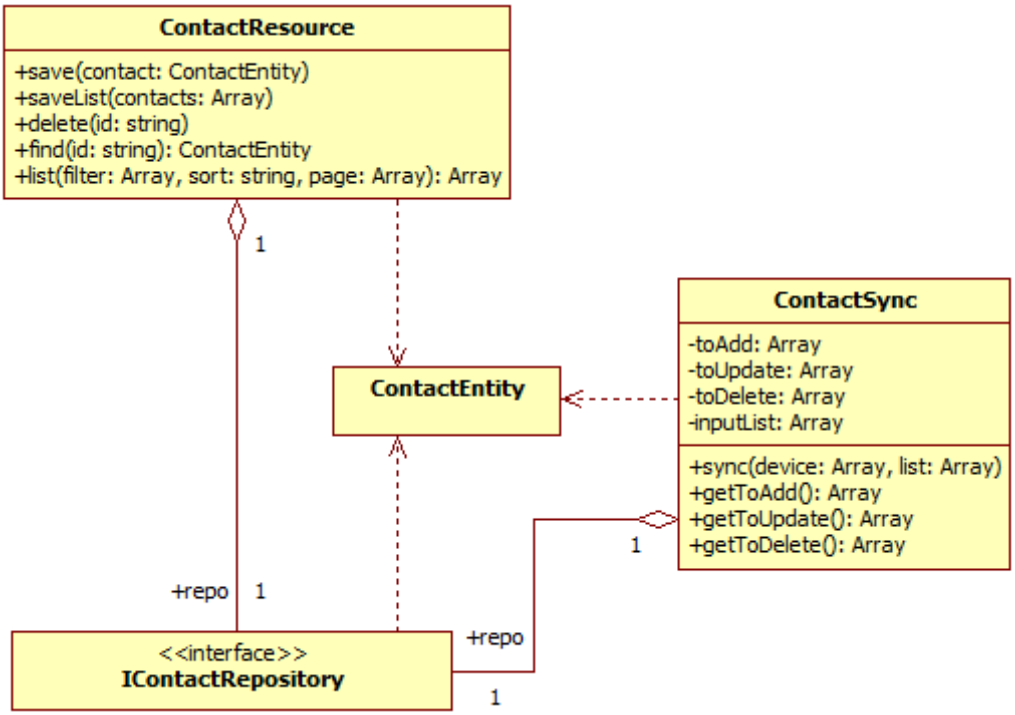



Рисунок 9 - Диаграмма классов API

### 2.4.3 Синхронизация адресной книги

Рассмотрим процесс синхронизации адресной книги. В качестве входных данных выступает массив объектов - адресная книга пользователя с внешнего источника.

1. Для каждого элемента входного массива происходит поиск соответствующего элемента в локальной БД. При этом поиск осуществляется по значению полей *deviceSync.type, deviceSync.deviceID, deviceSync.abID.*

2. Если элемент найден, сравнивается время последнего изменения контакта в локальной БД с временем последней синхронизации с данного устройства. Только в том случае, если контакт не менялся локально в период с прошлого цикла, происходит его обновление по стандартным правилам. Таким образом, локальные изменения адресной книги имеют приоритет над "внешними".

3. Если элемент не найден, происходит поиск похожего контакта по правилам устранения дублирующих записей (см. раздел 2.2.5). Если такой контакт уже существует, происходит слияние данных по тем же правилам. Первичным выступает существующий локально контакт, а дополняющим - переданный извне. В массив *deviceSync* добавляется новый элемент, идентифицирующий "внешний" контакт.

4. Если контакт для слияния не найден, создается новый контакт по стандартным правилам создания. В массив *deviceSync* также добавляется новый элемент.

5. Происходит выборка локально существующих контактов по следующим критериям:

- контакт принадлежит данной адресной книге (поле *userID*);
- контакт не принадлежит множеству добавленных, измененных или слитых в текущем цикле записей.

6. Каждый из выбранных элементов обрабатывается следующим образом. Если время создания или последнего изменения элемента больше времени последнего цикла синхронизации с данным устройством, элемент помещается во

множество контактов, предназначенных для сохранения на стороне клиента. При этом вместе с каждым контактом передаются значения из соответствующего элемента *deviceSync* (по значениям *type* и *deviceID*). Если такой элемент есть, значит клиент должен обновить существующую запись. В противном случае предполагается создание новой записи.

7. Если контакт не был создан или изменен с времени последнего цикла синхронизации и при этом он был ранее получен с того же устройства (проверяется наличием элемента *deviceSync* по паре значений: *type*, *deviceID*), контакт удаляется из локальной базы данных.

8. Извлекаются удаленные контакты по критериям:

- время удаления находится в промежутке между прошлым циклом синхронизации и текущим временем;
- контакт принадлежал текущему устройству (*type*, *deviceID*).

Полученные записи помещаются во множество контактов, предназначенных для удаления на стороне клиента.

Описанный выше процесс реализован в виде функции класса *ContactSync*. В качестве источника данных выступает экземпляр *IContactRepository*. Таким образом, используется логика работы с данными такая же как и в остальных модулях.

# 3 Тестирование компонента и устранение проблем

## 3.1 Организация тестирования

Для тестирования компонента, параллельно с его разработкой создавалось веб-приложение "оболочка" на языке PHP с использованием фреймворка *Laravel 5.2* [15]. Для установки фреймворка и дополнительных библиотек был использован популярный менеджер пакетов *Composer*.

Разрабатываемый компонент интегрирован в приложение в качестве его "ядра". В целом данное приложение выполняет следующие функции:

1. Единовременная синхронизация адресной книги с почтовым сервисом на выбор (используется модуль *Contacts\Sync*). Параметры: *userId,* тип сервиса. Реквизиты для работы со сторонними API берутся из конфигурационного файла приложения. Осуществляется путем обращения к скрипту *sync.php* через браузер.

2. Синхронизация с мобильным приложением. Для этого был реализован REST API напрямую использующий модуль *Contacts\API*. Какая-либо авторизация отсутствует и для идентификации адресной книги используется параметр userId.

3. Просмотр адресной книги при помощи простой страницы, сверстанной с использование фреймворка *Bootstrap* [16]. Функции добавления, редактирования, удаления контактов или распределения по группам не предусмотрены. Данный аспект использования компонента тестировался при помощи мобильного приложения, сопряженного с REST API. Снимки экрана данной страницы представлены в приложении А.

Таким образом, стало возможным тестирование всех аспектов компонента в условиях, приближенных к производственным. В процессе тестирования отдельных модулей были выявлены и устранены различные проблемы. В последующих подразделах будет уделено внимание некоторым из них.

### 3.2 Проблема производительности при большом количестве записей

Использование коллекции в СУБД MongoDB решает ряд проблем, характерных при хранении больших объемов записей в традиционных реляционных таблицах:

- масштабирование (за счет встроенной функции "шардинга" - разбиения одной логической БД на несколько физических, распределенных по разным серверам);
- быстрая выборка одной записи по ключу (в основном за счет отсутствия необходимости обрабатывать текст SQL-запроса).

Однако остается проблема поиска записи в том случае, когда ключ неизвестен. Такой поиск осуществляется в рамках алгоритма синхронизации (активной или пассивной) контактов с внешним источником а также алгоритма объединения дубликатов.

В процессе нагрузочного тестирования ранней версии модуля пассивной синхронизации (web API) при помощи мобильного приложения, была выявлена катастрофически низкая эффективность обработки запроса на обновление данных, получаемых с мобильного устройства. Обработка затягивалась настолько, что Web-сервер обрывал HTTP-соединение по таймауту.

В процессе отладки было выявлено, что основное время затрачивается на поиск "похожей" записи в БД. Параметры нагрузочного тестирования представлены в таблице 4.

Таблица 4 - Параметры нагрузочного тестирования прототипа Web API

| Показатель | Значение |
|---|---|
| Количество записей, передаваемых на сервер | от 3000 до 4000 |
| Среднее время обработки веб-запроса | 74,0 с. |
| Среднее время поиска "похожих" записей | 62,6 с. |
| Среднее время прочих операций выборки | 1,2 с. |
| Среднее время прочих операций записи | 4,1 с. |

Исходя из представленных данных, очевидно, что основное время затрачивается как раз на поиск записей, "похожих" на данную, по таким полям как имя, фамилия, номер телефона, адрес электронной почты. Анализ программного кода и физической структуры коллекции выявил следующую проблему.

В формируемом запросе к СУБД присутствует оператор поиска подстроки с использованием языка регулярных выражений. Данный оператор был использован для того чтобы обеспечить регистронезависимый поиск, так как стандартными средствами MongoDB такой поиск недоступен. При использовании данного оператора в MongoDB невозможно использование индексов по соответствующему полю. В результате, СУБД необходимо обрабатывать каждую запись из некоторого подмножества, отобранного с использованием других индексов, выполняя при этом ресурсоемкую операцию поиска с применением языка регулярных выражений.

В качестве решения были применены следующие меры:

- в каждую запись коллекции добавлены служебные поля, содержащие все поля контакта, подлежащие отбору, но строго в нижнем регистре и без пробелов в начале или конце строк;,
- убран оператор поиска с использованием регулярных выражений и заменен на обычный поиск по вхождению строки, при котором возможно использование обычных строковых индексов;
- добавлен специальный индекс в коллекцию, содержащий все поля контакта, подлежащие отбору (служебные поля);

– входные данные для поиска записей подвергнуты переводу в нижний регистр и очистке от пробелов в конце и в начале строк (таким образом, возможен регистронезависимый поиск записей, с учетом возможных лишних и незаметных для пользователя пробелов).

После всех изменений было проведено повторное тестирование, параметры которого представлены в таблице 5Таблица 5.

Таблица 5 - Параметры нагрузочного тестирования Web API после изменений

| **Показатель** | **Значение** |
|---|---|
| Количество записей, передаваемых на сервер | от 3000 до 4000 |
| Среднее время обработки веб-запроса | 6,0 с. |
| Среднее время поиска "похожих" записей | 1,1 с. |
| Среднее время прочих операций выборки | 1,2 с. |
| Среднее время прочих операций записи | 4,1 с. |

Исходя из представленных данных, можно сделать вывод, что указанная проблема полностью решена. Время обработки запроса с максимально допустимым объемом записей находится в пределах допустимых значений.

### 3.3 Проблема сравнения времени при синхронизации с Hotmail

В процессе синхронизации с внешними хранилищами контактов основной задачей является определение изменений, которые необходимо внести в локальную БД. При этом возможны ситуации, когда с момента прошлой синхронизации контакт был изменен как в локальной системе, так и в сторонней (например в Gmail). В этом случае необходимо определить, какие изменения будут иметь более высокий приоритет.

В соответствии с требованиями, приоритетными являются те изменения, который произошли позже.

В ходе тестирования прототипа модуля интеграции с сервисом Hotmail возникла проблема - часть изменений, внесенных пользователем в адресную книгу Hotmail, не попадала обратно в систему в процессе синхронизации. При этом не удалось обнаружить какой-либо явной закономерности между алгоритмом действий пользователя и успехом в синхронизации.

При обработке записей, полученных с сервиса Hotmail, в случае когда запись уже ранее импортирована, происходит сравнение даты и времени последнего изменения с датой и временем последней синхронизации. Только если эта дата и время больше - происходит обновление записи в БД.

В ходе отладки была выявлена причина ошибки - на тестовом сервере локальное время отставало примерно на 20 секунд от точного времени. В результате, происходило сравнение отметок времени со сторонней системы (где время точное) с временем на локальном сервере (неверное время), и в 1/3 случаев условие не срабатывало, когда в действительности изменения были.

API данного сервиса на момент написания модуля не предоставляло функциональности по версионированию записей, поэтому во избежание лишних операций записи приходилось опираться на отметки времени последнего изменения.

В конечном счете было найдено следующее решение. В качестве времени последней синхронизации вместо локального времени бралось время, получаемой в HTTP-заголовке Date ответа на запрос контактов с Hotmail Contacts API. Множественные тесты показали, что отметка времени, получаемая из указанного заголовка соответствует точному времени во временной зоне стороннего сервера. Отметки времени, содержащиеся внутри данных о контактах, соответствовали этой шкале времени.

Таким образом, происходило сравнение между отметками времени, вырабатываемыми одним и тем же сервером, в котором расхождение в шкалах времени в разных приложениях (веб-сервер и сам сервис API) маловероятно или исключено.

После данных манипуляций, ошибки синхронизации исчезли, что было доказано многократным выполнением пользователем тестовых примеров на протяжении длительного времени с успешным переносом всех изменений из Hotmail в локальную систему.

# 4 Руководство программиста

## 4.1 Структура каталогов

Программный компонент поставляется в архиве в виде набора файлов исходного кода на языке программирования PHP, а также файлов документации в формате TXT. Рассмотрим структуру каталогов архива.

- *src/* - код компонента;
  - *DAL/* - слой доступа к данным;
    - *Model/* - модели данных, содержит всего один файл модели ContactEntity.php;
    - *Repository/* - класс и интерфейс репозитория;
    - *Service/* - вспомогательные службы;
  - *Sync/* - модуль "активной" синхронизации;
  - *API/* - модуль "пассивной" синхронизации;
- *doc/* - документация (содержит руководство программиста, ответы на часто задаваемые вопросы, контактную информацию о разработчике);
- *examples/* - примеры интеграции компонента в виде набора текстовых файлов с "кусками" кода и описанием;
- *tests/* - автоматически тесты для основных классов системы (написаны с использованием фреймворка *PHPUnit*).

## 4.2 Структура классов

Весь программный код был организован с соблюдением стандарта PSR-4 [17]. В данном случае это означает, что структура классов соответствует структуре каталогов. Базовым пространством имен является *kvv/Contacts*. Дочерние пространства имен и классы соответствуют структуре каталога *src/*. Рассмотрим все программные классы и пространства имен компонента и их назначение.

- *kvv\Contacts\DAL* - слой доступа к данным;
  - *Model* - модели данных;
    - *ContactEntity* - модель данных о контакте (рассмотрена в разделе 2.2.3);
  - *Repository* - репозитории;
    - IContactRepository - интерфейс репозитория контактов (см. раздел 2.2.2);
    - *MongoContactRepository* - репозиторий контактов, использующий СУБД MongoDB;
  - *Service* - вспомогательные службы;
    - *ContactMerge* - служба объединения дубликатов;
    - *ModelValidator* - служба валидации данных модели по предоставляемой "схеме" данных;
    - *DataFilters* - статический класс, содержит функции для фильтрации таких данных как номера телефонов, электронные адреса.
- *kvv\Contacts\Sync* - модуль "активной" синхронизации;
  - *SyncManager* - менеджер процесса синхронизации;
  - *ISyncService* - интерфейс для службы синхронизации;
  - *BaseSyncService* - базовый класс для всех служб;
  - *YahooSyncService* - служба для работы с Yahoo Contacts;
  - *GoogleSyncService* - служба для работы с GMail Contacts;
  - *HotmailSyncService* - служба для работы с Outlook Contacts;
- *kvv\Contacts\API* - модуль "пассивной" синхронизации;
  - *ContactResource* - предоставление доступа к адресной книге, как RESTful-ресурсу;
  - *ContactSync* - синхронизация адресной книги с предоставленным набором данных.

### 4.3 Рекомендации по внедрению

Рассмотрим рекомендуемый порядок внедрения программного компонента для организации и синхронизации электронных контактов.

Первым делом необходимо установить сам компонент. Делается это стандартным образом для PHP-библиотек. Содержимое архива (см. 4.1) извлекается в каталог *./{vendor}/kvv/Contacts/*, где {vendor} - каталог сторонних библиотек приложения. В случае если в приложении используется PSR-4-совместимый автозагрузчик классов, никаких дополнительных действий не потребуется. В противном случае необходимо настроить используемый автозагрузчик таким образом, чтобы были доступны все классы компонента. Базовым пространством имен для автозагрузки будет являться *kvv\Contacts*, а базовым каталогом - *src/*.

Для работы с адресными книгами можно использовать репозиторий *MongoContactRepository*, который в своем конструкторе получает данные для подключения к СУБД MongoDB. Соединение с СУБД поддерживается до тех пор пока экземпляр репозитория не будет разрушен. Если необходимо использовать другую СУБД, или представленная реализация не отвечает каким-либо требованиям, можно написать свою реализацию интерфейса *IContactRepository*.

Контроль за осуществлением синхронизации возлагается на разработчиков приложения. Для выполнения одного цикла активной синхронизации, можно использовать класс *SyncManager*, передав в его конструктор экземпляр *IContactRepository*.

Рекомендуется повторять цикл синхронизации с периодичностью в 10 минут. При этом необходимо следить за ограничениями на троттлинг соответствующих сервисов. Самым простым (и медленным) решением будет разделение циклов во времени при помощи, например, блокировок с ожиданием, опираясь на идентификатор процесса (который при каждом запуске PHP-скрипта будет разный).

Для пассивной синхронизации компонент не предоставляет готового к использованию Web API. Для её использования необходимо написать свой API слой (либо использовать одну из множества готовых библиотек). Так же можно просто интегрировать методы представленного компонента в существующий API. Если при этом придерживаться RESTful-подхода, интеграция с модулем пассивной синхронизации будет наиболее простой. Модуль содержит класс *ContactResource*, который придерживается REST-подхода и является сервисной прослойкой между контроллером API и репозиторием.

В типичном случае использование модуля пассивной синхронизации сводится к написанию "тонких" контроллеров API, которые всего лишь вызывают соответствующие методы классов *ContactResource* и *ContactSync*.

При этом на уровне контроллера (и/или middleware) необходимо обеспечить:

- представление данных (модели представления, сериализацию и десериализацию);
- валидацию входных данных;
- обработку ошибок;
- систему авторизации.

При работе с адресными книгами, для их различения необходимо использовать поле *userID*. Рекомендуется в качестве значения передавать, например, идентификатор пользователя системы.

# ЗАКЛЮЧЕНИЕ

В процессе выполнения дипломного проекта, была выявлена и проанализирована проблема хранения, выборки, обработки, синхронизации данных о контактах адресных книг для использования в корпоративной сфере. Были изучены существующие решения в данной области.

На основе полученных данных был спроектирован программный компонент для организации и синхронизации электронных контактов. Были приняты решения о выборе языка программирования, среды разработки и используемой СУБД, а также форма решения в виде встраиваемого программного компонента. Были выявлены задачи и структура каждого модуля, составлены диаграммы важнейших классов будущего компонента.

Наконец, на основе полученных проектных решений был разработан и протестирован прототип компонента. Результаты тестирования были проанализированы, выявлены ключевые проблемы, не позволяющие использовать прототип в полной мере, и пути их решения.

Конечным результатом работы является полностью функционирующий встраиваемый программный компонент, который позволяет эффективно решать поставленные задачи.

Компонент был встроен в реальную программную систему заказчика и находится в стадии пробного использования конечными пользователями. В полной мере используется интеграция с почтовым сервисом GMail. Кроме того, заказчиком был разработан прототип мобильного приложения для операционной системы iOS, использующий функциональность представленного компонента для синхронизации адресной книги мобильного устройства с внутренней корпоративной системой.

По итогам обсуждения результатов тестирования с заказчиком, было решено в дальнейшем развивать компонент в следующих направлениях.

1. Реализовать обмен контактными данными между различными пользователями. Для этого вводится понятие категории приватности контакта (опре-

деляет какой набор данных будет видеть данный контакт). Будет реализован визуальный пользовательский интерфейс, который позволит не только редактировать адресную книгу, но и вводить контактные данные самого пользователя, который также сможет настроить несколько категорий приватности и набор видимых полей для каждой из них.

Работать это будет следующим образом. Каждому своему контакту пользователь сможет посылать запрос на синхронизацию. Если в системе будет найден пользователь (или создан новый), чьи данные по определенным критериям совпадут с данными контакта, он автоматически получит данный запрос (в котором будут содержатся только те данные, которые по решению пользователя-отправителя соответствуют выбранной им категории) и сможет принять или отклонить его. В случае принятия, активируется двусторонний обмен данными. После этого любые изменения в контактных данных двух пользователей будут автоматически отражены в соответствующих записях адресных книг.

2. Реализовать виртуальные визитные карточки. Для каждой категории приватности пользователь сможет выбрать один из предопределенных шаблонов оформления визитной карточки. У каждого шаблона будет набор доступных цветовых схем. Далее, в зависимости от доступного для данной категории набора информации, по определенным правилам будет формироваться визитная карточка.

Предполагается использовать технологию HTML+CSS для реализации карточек. Для улучшения качества отображения визитных карточек (например, в будущем веб-интерфейсе или мобильном приложении), было предложено для каждой виртуальной визитной карточки генерировать растровую версию под несколько предопределенных размеров. Для этого будет использовано приложение *PhantomJS*.

По результатам всей работы была составлена пояснительная записка в соответствии с требованиями [18].

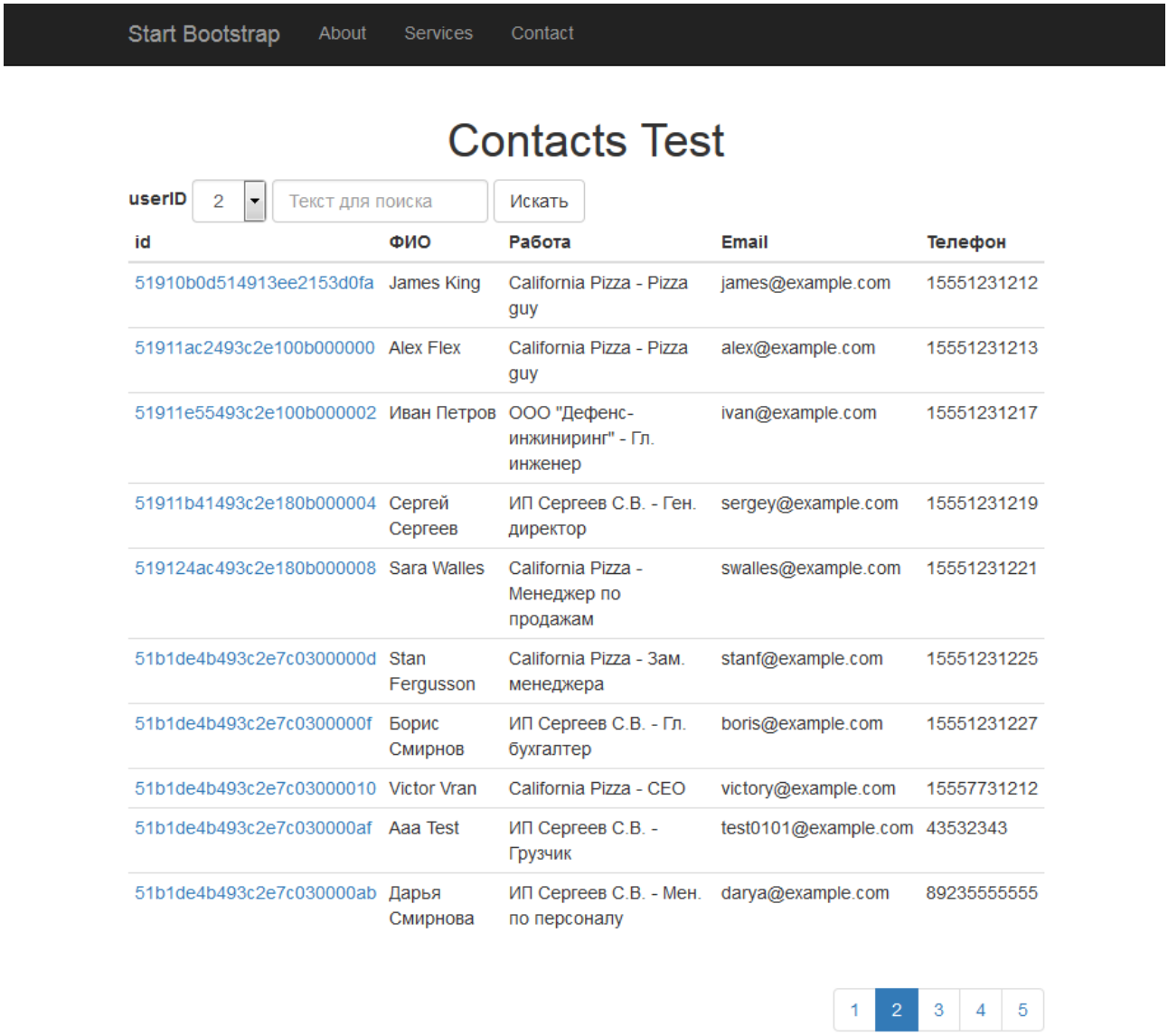
## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. Outlook Contacts REST API Reference [Электронный ресурс] // Документация к Outlook Contacts API. – Режим доступа: https://msdn.microsoft.com/en-us/office/office365/api/contacts-rest-operations (дата обращения: 02.04.2016 г.)
2. Contacts API [Электронный ресурс] // Документация к Yahoo Contacts API. – Режим доступа: https://developer.yahoo.com/social/rest_api_guide/contact_api.html (дата обращения: 02.04.2016 г.)
3. Google Contacts API version 3.0 [Электронный ресурс] // Документация к Google Contacts API. – Режим доступа: https://developers.google.com/google-apps/contacts/v3/ (дата обращения: 02.04.2016 г.)
4. Contact Management Products - FullContact [Электронный ресурс] // Официальный сайт FullContact – Режим доступа: https://www.fullcontact.com/apps/ (дата обращения: 06.04.2016 г.)
5. PHP: Руководство по PHP [Электронный ресурс] // Руководство по PHP на русском языке. – Режим доступа: http://php.net/manual/ru/ (дата обращения: 07.05.2016 г.)
6. Басс, Л. Архитектура программного обеспечения на практике / Л. Басс, П. Клементс, Р. Кацман. – 2-е изд. – СПб. : Питер, 2006. – 575 с.
7. Вендров А.М. CASE–технологии. Современные методы и средства проектирования информационных систем. – М.: Финансы и статистика, 2000.
8. Боггс, Уэнди. UML и Rational Rose / Уэнди Боггс, Майкл Боггс. – М. : Изд-во «Лори», 2001. – 580 с
9. Рамбо, Дж. UML 2.0. Объектно-ориентированное моделирование и разработка / Дж. Рамбо, М. Блаха. – 2-е изд. – СПб. : Питер, 2007. – 544с.

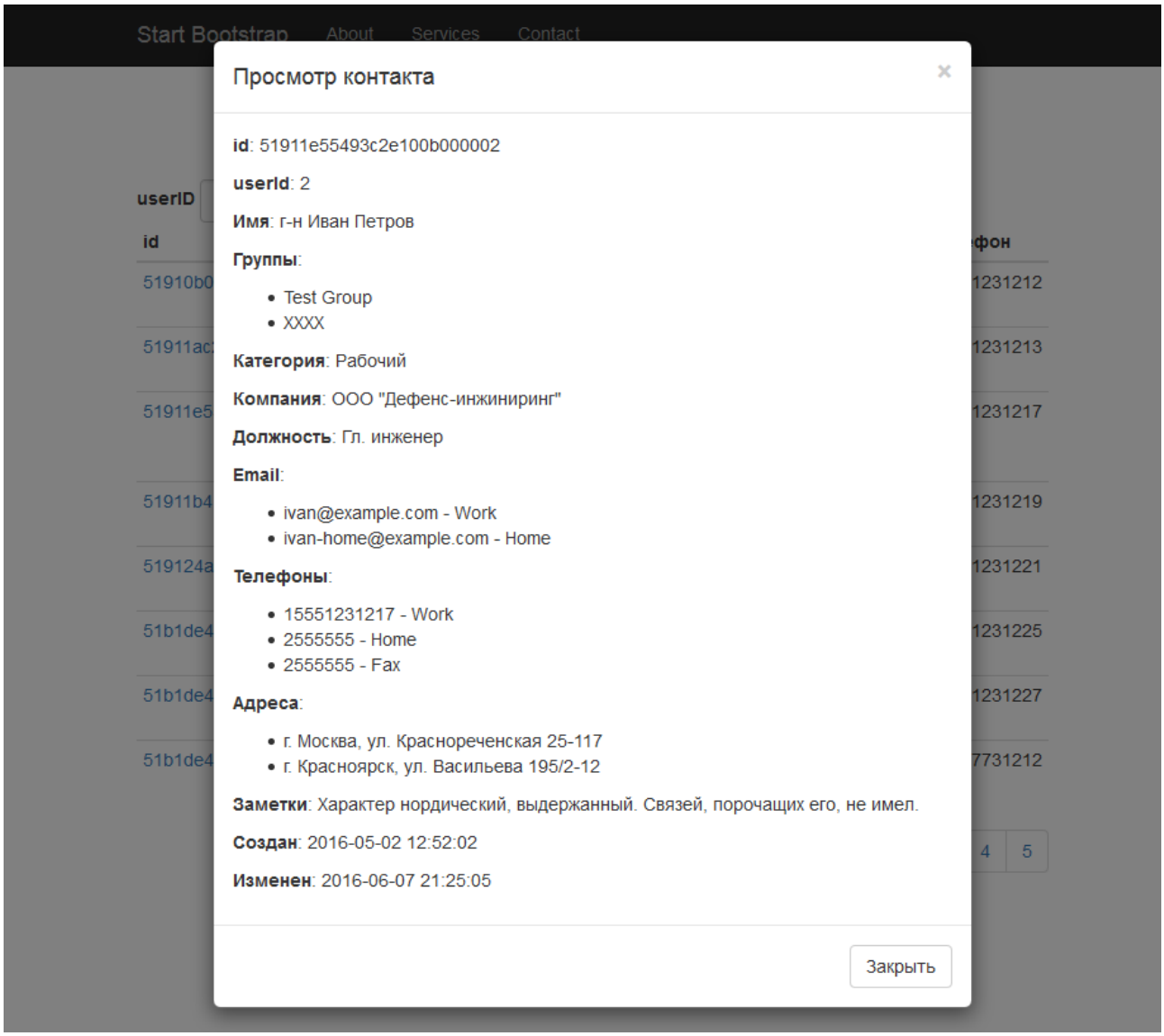
10. Приемы объектно-ориентированного проектирования. Паттерны проектирования. / Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. — СПб:Питер, 2001. — 368 с.
11. Design Principles and Design Patterns. / Robert C. Martin — www.objectmentor.com, 2000. — 34 с.
12. The MongoDB 3.2 Manual [Электронный ресурс] // Официальная документация СУБД MongoDB. – Режим доступа: https://docs.mongodb.org/manual/ (дата обращения: 03.05.2016 г.)
13. Коннолли, Томас, Бегг, Каролин, Страчан. Базы данных: проектирование, реализация и сопровождение. Теория и практика, 2-е изд. – М. : Издательский дом ”Вильямс”, 2001.
14. RFC 3339 - Date and Time on the Internet: Timestamps [Электронный ресурс] // Стандарт представления даты и времени – Режим доступа: https://www.ietf.org/rfc/rfc3339.txt (дата обращения: 03.06.2016 г.)
15. Laravel - The PHP Framework For Web Artisans [Электронный ресурс] // Документация к фреймворку Laravel – Режим доступа: https://laravel.com/docs/5.2 (дата обращения: 18.05.2016 г.)
16. CSS - Bootstrap [Электронный ресурс] // Документация к фреймворку Bootstrap 3 – Режим доступа: http://getbootstrap.com/css/ (дата обращения: 18.05.2016 г.)
17. PSR-4: Autoloader - PHP-FIG [Электронный ресурс] // Соглашение об автозагрузке классов PHP – Режим доступа: http://www.php-fig.org/psr/psr-4/ (дата обращения: 26.05.2016 г.)
18. СТО 4.2–07–2014 Система менеджмента качества. Общие требования к построению, изложению и оформлению документов учебной деятельности. – Введ. 09.01.2014. – Красноярск : ИПК СФУ, 2014. – 60 с.

# ПРИЛОЖЕНИЕ А

## Снимки экрана тестового приложения




Снимок таблицы-списка контактов




Снимок окна просмотра контакта