Санкт-Петербургский Государственный Университет

Факультет Прикладной Математики - Процессов Управления

**Семёнов Николай Сергеевич**

**Синхронизация данных академической структуры**

**с адресной книгой сервисов Google Apps**

Направление 010500

Прикладная математика и информатика

Руководитель направления

Д. ф.-м. н., профессор Жабко А. П.

Научный руководитель

Ст. преподаватель Севрюков С. Ю.

Рецензент

Архитектор департамента разработки ПО

Производственный центр Java технологий

Компания Рексофт Акимов В.А.

Санкт-Петербург

2011

Оглавление

[ВВЕДЕНИЕ 3](#_Toc295168231)

[ПОСТАНОВКА ЗАДАЧИ 6](#_Toc295168232)

[ГЛАВА 1. Определение требований и выбор оптимального решения 8](#_Toc295168233)

[§1. Исследование бизнес процессов 8](#_Toc295168234)

[§2. Формирование требований 10](#_Toc295168235)

[§3. Анализ решений и выбор оптимального подхода 13](#_Toc295168236)

[3.1. Первый подход. Выбор готового решения 13](#_Toc295168237)

[3.2. Второй подход. Разработка собственного решения 14](#_Toc295168238)

[§4. Выводы 16](#_Toc295168239)

[ГЛАВА 2. Разработка модуля синхронизации 17](#_Toc295168240)

[§1. Основная концепция и выбор технологий 17](#_Toc295168241)

[§2. Архитектура разрабатываемого решения 18](#_Toc295168242)

[§3. Определение сущностей будущей системы 21](#_Toc295168243)

[§4. Подмодули системы 23](#_Toc295168244)

[4.1. Модуль бизнес логики (EjbCore) 23](#_Toc295168245)

[4.2. WEB модуль 26](#_Toc295168246)

[4.3. Модуль синхронизации со службами Google 28](#_Toc295168247)

[4.4. Модуль репликации БД 29](#_Toc295168248)

[§5. Достигнутые результаты 30](#_Toc295168249)

[ГЛАВА 3. Тестирование модуля синхронизации 32](#_Toc295168250)

[§1. Анализ работа приложения на тестовых данных 32](#_Toc295168251)

[§2. Профилирование системы 34](#_Toc295168252)

[§3. Проверка возможности масштабирования системы 36](#_Toc295168253)

[§4. Выводы 37](#_Toc295168254)

[ЗАКЛЮЧЕНИЕ 39](#_Toc295168255)

[Список литературы и источников 40](#_Toc295168256)

[Приложение 1. Реализация модулей 43](#_Toc295168257)

[Приложение 2. Скриншоты программы 65](#_Toc295168258)

[Приложение 3. Рекомендуемые энциклопедические статьи 71](#_Toc295168259)

# ВВЕДЕНИЕ

Последнее десятилетие характеризуется ускоренными темпами вторжение информационных технологий во все сферы жизни человека. Компьютерные системы в корне изменили характер взаимоотношений между людьми. По-новому стало строиться взаимодействие между преподавателями и студентами. Новые эффективные средства коммуникации позволяют обеспечить гибкую организацию работы предприятий и учебных заведений, повысить результативность труда сотрудников, снизив до минимума объем обмена бумажными документами. В организациях с многоуровневой иерархической структурой подразделений применяются корпоративные информационные системы. Эти системы созданы для управления внутренними и внешними ресурсами предприятия. Они обеспечивают автоматизацию большинства направлений и функций типового предприятия: от бухгалтерской и налоговой отчетности до управления логистикой и ведением кадрового учета. Системы электронного документооборота обеспечивают оперативный доступ к документам, позволяют отслеживать их согласование и принятие решений.

С развитием компании количество используемых информационных систем неуклонно растет, возникает серьезный вопрос об их эффективном взаимодействии. Добавление нового сервиса в развернутую инфраструктуру организации увеличивает риск серьезно снизить или даже нарушить работоспособность системы. Для многих предприятий отказ системы является критичным для нормального функционирования бизнеса: час «простоя» влечет за собой потерю значительной части прибыли или может вовсе лишить половины клиентской базы. Поставщики программного обеспечения стараются максимально снизить риск подобных происшествий. На помощь приходят квалифицированные кадры и фирмы, занимающиеся системной интеграцией.

Одним из основных ресурсов на любом предприятии являются его сотрудники. Рост числа сотрудников обостряет проблему управления персональными данными и вопрос о налаживании эффективного общения внутри коллектива. Одним из самых распространенных способов корпоративного взаимодействия является электронная почта. В настоящее время любой пользователь может завести свой бесплатный электронный почтовый ящик: достаточно зарегистрироваться на одном из интернет-порталов. К сожалению, для корпоративных задач этого мало. Зачастую, требуется возможность совместной работы в Интернете с текстовыми документами, электронными таблицами и презентациями, причем доступ к этим документам должен предоставляться только отдельным сотрудникам, определенным политикой предприятия. Так же стоит учесть, что каждый день происходит смена статуса персон, входящих в организацию: кто-то увольняется, кто-то переходит на другую должность или в другой отдел. Необходимо, чтобы эти изменения производились и в корпоративной почтовой системе.

Как правило, подобные проблемы решаются путем внедрения автоматизированных модулей, обеспечивающих синхронизацию персональных данных между внутренней базой предприятия и базой почтового сервиса. Для управления доступом и групповой политикой используются сценарии синхронизации, которые формируются либо автоматически, согласно общей иерархической структуре компании, либо индивидуально каждым сотрудником.

Многие почтовые сервисы имеют собственные средства синхронизации, позволяющие наладить взаимодействие с большинством корпоративных систем. Однако, если предприятие имеет нестандартную инфраструктуру и различные компоненты системы поставляются разными поставщиками программного обеспечения, возникает ряд трудностей и приходится прибегать к разработке собственного модуля, выполняющего необходимые функции. Из всего выше сказанного следует важный тезис: существующие механизмы синхронизации персональных данных и управления групповой политикой не являются универсальными, и, в ряде случаев, они просто не способны обеспечить необходимый уровень «гибкости» для выполнения специфических задач.

# ПОСТАНОВКА ЗАДАЧИ

**Целью данной работы** является повышение эффективности корпоративного взаимодействия между различными подразделениями и субъектами университета. Для достижения поставленной цели предполагается решить **задачу** **разработки прототипа** модуля синхронизации персональных данных студентов, сотрудников и преподавателей, а так же академической структуры факультетов с адресной книгой сервисов Google Apps.

Решение данной задачи требует **реализации следующих этапов**:

1. Исследование предметной области:
   * Анализ бизнес процессов и определение потребностей персонала организации;
   * Изучение сервисов электронной почты, обмена документами и работы с календарями;
   * Исследование возможностей, достоинств и недостатков коммерческих проектов, обеспечивающих синхронизацию с почтовыми службами;
2. Формирование требований к программному продукту;
3. Анализ и обоснование выбора оптимального подхода (на основе пунктов 1 и 2);
4. Разработка модуля синхронизации:
   * Выбор технологий, обеспечивающих максимальную интеграцию с API Google, с учетом требований;
   * Разработка архитектуры предполагаемого решения;
   * Реализация необходимых модулей;
5. Тестирование модуля синхронизации:
   * Проверка работы приложения на тестовых данных;
   * Анализ времени генерации страницы, расхода памяти на пользовательскую сессию, времени непосредственной синхронизации;
   * Проверка возможности масштабирования системы с использованием вычислительного кластера.

Сформулированная выше задача будет решаться для Юридического факультета Санкт-Петербургского Государственного Университета, который обладает необходимой технической и организационной базой, а также потребностью в обозначенном исследовании.

# ГЛАВА 1. Определение требований и выбор оптимального решения

В данной главе будет исследована предметная область и проведен анализ бизнес процессов Юридического факультета Санкт-Петербургского Государственного Университета. Будут сформулированы требования к разрабатываемому решению и выбран оптимальный подход, позволяющий в полной мере покрыть весь необходимый функционал и, в тоже время, минимизировать финансовые и временные затраты.

## §1. Исследование бизнес процессов

Основным объектом учебного процесса является информация. Она создается, изменяется и передается от одного участника учебного процесса к другому (другим). Наглядным примером может служить взаимодействие преподавателя с учебной группой или администрации университета с педагогическим коллективом.

Обмен информацией может проходить как в одностороннем режиме, так и в режиме диалога. Сама информация может иметь различную форму и значение. Это могут быть короткие текстовые сообщения или многостраничные юридические документы, научные публикации или данные об анкетировании студентов. В общем и целом информация отличается по сложности, продолжительности (сроку актуальности), атомарности и важности.

В рамках работы были исследованы основные бизнес задачи студентов, преподавателей и сотрудников университета. Ниже представлены некоторые из них:

* обеспечения оперативной связи между студентами, преподавателями и руководством;
* совместная работа с документами;
* рассылка учебно-методических материалов;
* организация консультации студентов;
* своевременное информирование студентов об академических и финансовых задолженностях;
* автоматизированная рассылка уведомлений;
* обеспечение связи с выпускниками.

Существует несколько подходов к решению описанных выше задач. В любом случае, для ежедневной деловой и личной переписки в качестве средства коммуникации разумно выбрать электронную почту. По мнению аналитиков Gartner Group [1], почти 97% организаций поддерживают взаимную деловую активность с её помощью.

Необходимо учесть, что мы рассматриваем конкретную организацию с уже созданной и активно использующейся информационной средой. Учитывая это дополнение, можно рассматривать два подхода для решения задач:

1. создание всей необходимой инфраструктуры с применением собственных мощностей;
2. использование сторонних почтовых сервисов, например таких как Microsoft Live@edu или Google Apps.

На сегодняшний день первый подход частично реализован: у факультета есть система управления электронной почтой на базе Microsoft Exchange Server. Она обладает всеми необходимыми функциями, однако является дорогостоящей (с точки зрения владения) и применяется в первую очередь для взаимодействия сотрудников и администрации факультета. Использование данного решения для всех участников учебного процесса – тысяч студентов и сотен преподавателей – не оправдано по финансовым соображениям. В настоящее время активно внедряются службы Google Apps (второй подход), хорошо зарекомендовавшие себя в большом количестве учебных заведений.

Внедрение почтового сервиса позволяет разрешить большинство бизнес задач. Однако оба подхода не являются в полной мере универсальными, и остается ряд существенных недостатков. Основными из них являются:

* отсутствие согласованности с организационной структурой;
* администрирования профиля и данных возлагается исключительно на владельца;
* вопросы рассылок и публикации информации зависят только от пользователей.

Эти недостатки могут показаться не такими критичными, но, прежде всего, они создают неудобства для пользователей. Количество человек задействованных в учебном процессе в определенный момент времени может достигать тысяч человек и поддержание актуальности информации становится трудновыполнимой задачей.

## §2. Формирование требований

В ходе анализа бизнес процессов были выделены базовые сценарии – последовательности действий, производимых пользователем в рамках решения отдельной бизнес задачи. В большинстве случаев сценарии имеют схожую **последовательность начальных действий**:

1. пользователь выбирает субъектов для информационного взаимодействия (студентов, преподавателей, академические группы и т.п.)
2. первичный экспорт необходимых данных в адресную книгу Google;
3. синхронизация персональных данных в соответствии с базой данных факультета.

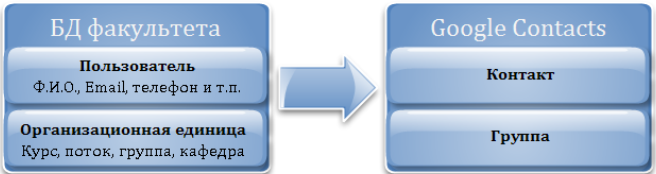
Таким образом, разрабатываемый программный продукт должен в полной мере обеспечивать выполнение этих 3 базовых операций.

Под **синхронизацией** следует понимать ликвидацию различий между двумя копиями данных (хранящихся в базе данных факультета и службах Google). Предполагается, что ранее эти копии были одинаковы, а затем одна из них, либо обе были независимо изменены. В нашем случае, под данными подразумевается:

* персональная информация пользователей;
* описание структурных подразделений факультета;
* членство пользователей в организационных единицах.

Обычно, рассматривают одностороннюю и двустороннюю синхронизацию. В рамках данной работы вопроса двусторонней синхронизации утрачивает свою актуальность, так как авторитетным источником данных является подсистема управления персональными данными факультета и никакие изменение контактной информации извне не планируются. Таким образом, нет необходимости разрешать так называемый «конфликт правок».

На рисунке 1 схематично представлен процесс синхронизации:

 Рис. 1. Схема синхронизации данных

Следует обратить внимание на то, что существует несколько категорий пользователей. Для каждой группы предусмотрены свои права доступа (в соответствии с групповой политикой факультета). Так же иногда возникает потребность обеспечить определенные лица дополнительными правами.

Обобщая все вышесказанное, можно сформулировать **требования**, которым должен удовлетворять разрабатываемый программный комплекс:

1. Возможность выбора объектов синхронизации с учетом ролевой модели;
2. Первичный экспорт и последующая односторонняя синхронизация пользовательских данных между базой данных факультета и адресной книгой Google;
3. Первичный экспорт и синхронизация организационных единиц;
4. Первичный экспорт и синхронизация членства контактов в организационных единицах;
5. Ведение истории синхронизации;
6. Контроль ошибок, возникающих при синхронизации;
7. Масштабируемость комплекса и способность выдерживать необходимое количество пользователей;
8. Гибкость архитектура комплекса (адаптивность к изменениям функциональных требований).

Безусловно, безопасность является одним из ключевых моментов деятельности Юридического факультета. В связи с этим разрабатываемый комплекс должен обеспечивать необходимый уровень информационной безопасности.

## §3. Анализ решений и выбор оптимального подхода

На сегодняшний момент существует значительное количество способов решения поставленных выше задач. Из всего многообразия решений можно выделить два принципиально различных подхода.

### 3.1. Первый подход. Выбор готового решения

Одним из самых распространенных подходов решения поставленной задачи является использование средств синхронизации, поставляемых компанией Google.

Наиболее универсальным продуктом является **Google Apps Directory Sync** [2]. С помощью этой специальной утилиты можно автоматически создавать пользователей, группы и контакты для лиц на основе пользовательских данных на сервере LDAP, например Microsoft Active Directory или Lotus Domino. Google Apps Directory Sync подключается к каталогу Google Apps и добавляет или удаляет аккаунты пользователей согласно существующей организационной схеме. Выполняется односторонняя синхронизация. Данные на сервере LDAP не обновляются и не изменяются.

Мастер конфигурации Google Apps Directory Sync помогает настроить и выполнить сопоставление списка пользователей на сервере LDAP с пользователями, псевдонимами, общими контактами и группами Google Apps. Кроме того, можно синхронизировать данные расширенных профилей пользователей, например номера домашних, рабочих и мобильных телефонов, адреса и должности. Средства управления синхронизацией позволяют выполнять тестовые синхронизации, ограничивать изменения, настраивать уведомления и график синхронизаций.

Кроме того компания Google разработала механизмы для быстрой миграции с самых популярных почтовых серверов. Выделяют два способа переноса электронной почты:

* Со стороны администратора/сервера;
* Со стороны конечного пользователя/клиента.

В рамках поставленных функциональных требований к ПО стоит рассмотреть только первый способ. Таким образом, для синхронизации на серверной стороне применимы следующие утилиты:

* **Google Apps Migration for Microsoft® Exchange** (рекомендуется для большинства переносов, в том числе с большинства серверов IMAP). Перенос почты, календаря и контактов с сервера Microsoft® Exchange в Google Apps [3];
* **Google Apps Migration for Lotus Notes**. Собственный инструмент Lotus Notes для работы с базой данных, который позволяет переносить почту, календарь, контакты и информацию о группах из аккаунтов Notes в Google Apps [4].

### 3.2. Второй подход. Разработка собственного решения

Следующим подходом является написание собственного комплекса синхронизации. Такое решение позволяет сохранить уже существующую инфраструктуру, а также дает возможность адаптировать разрабатываемый программный комплекс под изменяющиеся требования и политику управления информационной безопасностью на факультете. Программный продукт, созданный «с нуля», как правило, обладает гибкой архитектурой и предполагает потенциал дальнейшего развития.

Безусловно, этот подход не лишен недостатков. Разработка собственного решения требует большего времени и, как правило, больших финансовых затрат нежели покупка и внедрение «коробочного» продукта. Не стоит забывать, что создание полноценного программного продукта – это очень ответственное занятие, требующее продуманной методологии разработки и четкой организации трудового процесса.

Разработка программного обеспечения, как правило, включает в себя следующие этапы:

* анализ требований заказчика;
* проектирование;
* реализация проекта;
* тестирование системы;
* внедрение и сопровождение.

В общем и целом, второй подход хорошо зарекомендовал себя в тех случаях, когда фирме-заказчику требуется реализовать специфический функционал, не имеющий реализации на рынке корпоративного ПО. К этому подходу так же обращаются, когда внедрение «коробочного» продукта представляет собой сложное и дорогостоящее организационно-техническое мероприятие.

## §4. Выводы

Анализ функциональных требований, предъявляемых к разрабатываемому программному обеспечению, и двух основных подходов решения позволил сделать следующие выводы:

* Использование первого подхода, т.е. применение встроенных средств синхронизации Google Apps, позволит осуществить быструю и недорогую миграцию с сервера Microsoft Exchange [5] Юридического факультета (реализовать требования 2-6). Однако данных подход не способен обеспечить выполнение одного из основных требований – возможности выбора объектов синхронизации с учетом ролевой модели. Для его реализации придется прибегать к разработке дополнительного модуля и обеспечить их интеграцию;
* Применение второго подхода в рамках данной работы наиболее оправдано, так как позволяет удовлетворить всем поставленным требованиям. Следует учесть, что при разработке такого программного продукта придется столкнуться с вопросами качества, стоимости и надёжности. Таким образом, прежде чем заказывать разработку ПО, необходимо подготовить развернутое техническое задание и все сопутствующие документы;
* Для анализа работы предполагаемой системы и подготовки документации разумно обратиться к созданию прототипа. При использовании этого подхода можно выявить важные архитектурные ошибки, внести поправки в интерфейсы и перераспределить функциональность между модулями системы.

В итоге, было решено создать прототип, содержащий реализацию базовой функциональности. Этапы разработки и тестирования этого прототипа представлены в следующих главах.

# ГЛАВА 2. Разработка модуля синхронизации

На основании выводов, сформулированных в предыдущей главе, перед составлением подробных спецификаций было решено разработать прототип будущей системы. Прототип подразумевает полностью работающую систему, удовлетворяющую предъявляемым базовым функциональным требованиям. Так как не предполагается дальнейшее развитие прототипа до эксплуатируемой системы, задача обеспечения максимального уровня информационной безопасности является второстепенной и рассматриваться не будет.

В этой главе рассматриваются вопросы архитектуры, технологических особенностей и реализации разрабатываемого прототипа.

## §1. Основная концепция и выбор технологий

Одной из ключевых особенностей разрабатываемого решения является тесная интеграция со службами Google. Предоставляемое сервисами Google API предусматривают два типа взаимодействия: посредством протокола или с использованием клиентских библиотек. Каждый из типов имеет свои преимущества и недостатки. Например, протокол, основанный на XML и HTTP, имеет расширенные возможности по сравнению с функционалом, реализованным в библиотеках. С другой стороны, использование библиотек позволяет сконцентрировать внимание разработчика на написании бизнес логики, оперируя высокоуровневым API. На момент написания работы, стабильные версии библиотек были выпущены для языков программирования Java и Python.

Безусловно, на этапе прототипирования, разумно применять механизмы, позволяющие максимально снизить время реализации программного комплекса. Таким образом, в данной работе для обеспечения взаимодействия со службами Google будут применяться клиентские библиотеки. В качестве инструмента разработки чаще всего используют языки программирования высокого уровня абстракции.

Реализации программного комплекса будет осуществляться на языке **Java, Enterprise Edition** (сокращенно Java EE) [6, 7, 8]. Такое решение было принято по нескольким причинам, основными из которых являются следующие:

Во-первых, Java хорошо зарекомендовал себя в сфере разработки портальных решений и корпоративных систем. Java EE является промышленной технологией и в основном используется в высокопроизводительных проектах, в которых необходима надежность, масштабируемость и гибкость.

Во-вторых, наличие клиентской библиотеки для языка Java позволит значительно ускорить процесс создания прототипа.

## §2. Архитектура разрабатываемого решения

После выбора основных технологий, необходимо определиться с архитектурой разрабатываемого комплекса. Проанализировав требования, предъявляемые к программному продукту, было принято решение об использовании модульной системы с трехуровневой архитектурой. Такое решение позволит получить максимальную гибкость, обеспечить возможность повторного использования модулей, облегчить поддержку и модифицирование системы в будущем, таким образом выполнив требования 7-8.

Существует несколько подходов построения модульной архитектуры. Например, с использованием спецификации OSGi (Open Services Gateway Initiative) [9, 10]. Существуют несколько реализаций стандарта OSGi с открытым исходным кодом, например Apache Felix [11] и Equinox [12]. Безусловно, у этой технологии немало преимуществ. Только тот факт, что она стала отраслевым стандартом компонентной сборки программ, говорит сам за себя. Однако, как показывает практика, добиться полной автономности модулей бывает достаточно трудно, остаются однонаправленные и циклические зависимости. Недостатком так же является тот факт, что построение приложений на основе OSGi значительно увеличивает продолжительность начальных этапов разработки.

Проанализировав большинство популярных подходов, было принято решение о построении программного комплекса с применением EJB и Web контейнеров. Такой подход поддерживается всеми современными серверами приложений. Это позволит производить раздельное модифицирование модулей и даст возможность независимого развертывания на сервер приложений.

На рисунке 2 приведена схема, отображающая архитектуру разрабатываемого комплекса и используемые в каждом модуле ключевые технологии. Более подробное описание применяемых технологий будет дано в §4.

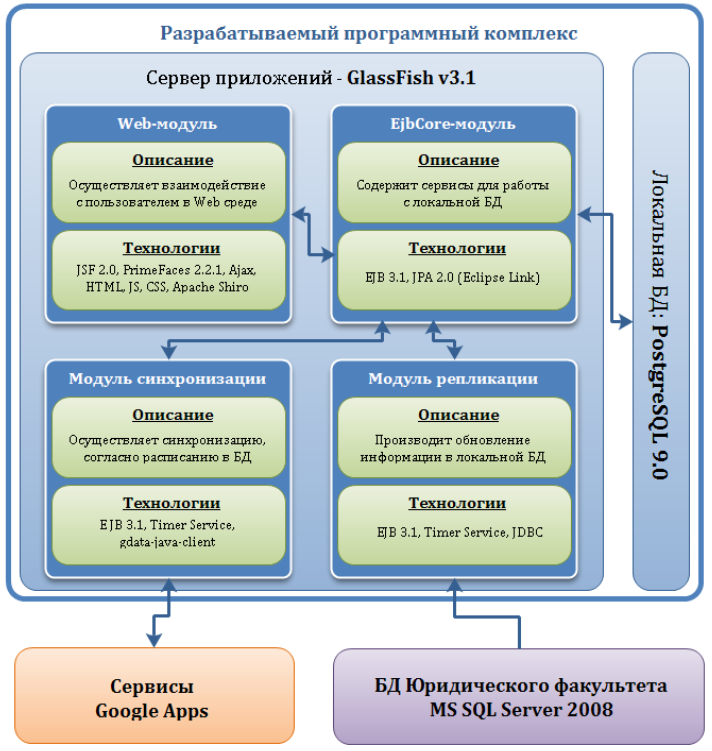


Рис. 2. Схема архитектуры программного комплекса

Из приведенной схемы видно, что проект содержит три EJB модуля и один Web модуль, развернутые на сервере GlassFish v3.1 [13]. Каждый модуль имеет определенное число внутренних и внешних зависимостей. Для платформы Java существуют два основных инструмента для сборки: Ant [14] и Maven [15]. У каждого из них есть свои достоинства и недостатки. В рамках данной работы было принято решение об использовании **Maven 3.0** [16, 17, 18] ввиду следующих преимуществ:

* Декларативное описание проекта на основе XML;
* Управление зависимостями. Сторонние библиотеки зачастую тоже в свою очередь используют библиотеки разных версий. Maven позволяет управлять такими сложными зависимостями. Что позволяет разрешать конфликты версий и в случае необходимости легко переходить на новые версии библиотек;
* Автоматическая загрузка библиотек из удаленных репозиториев.

## §3. Определение сущностей будущей системы

В соответствии с представленной схемой архитектуры разрабатываемый комплекс включает в себя локальную базу данных – **PostgreSQL 9.0** [19]. Решение о применении отдельной базы данных было принято по нескольким причинам:

* для работы Web модуля и модуля синхронизации требуется непрерывный доступ к базе данных, хранящей информацию о структуре подразделений и персональных данных пользователей;
* при использовании базы данных факультета есть риск нарушения нормального функционирования остальных систем факультета, ввиду падения производительности. Такая опасность может возникнуть в случае значительного роста количества запросов к базе данных со стороны нашего комплекса;
* применение локальной базы данных на этапе прототипирования позволит сформулировать требования к её производительности.

В процессе анализа бизнес процессов ВУЗа и исследования возможностей сервисов Google были выявлены основные сущности, оперирование которыми предусматривает разрабатываемый комплекс.

На рисунке 3 приведена диаграмма классов, являющихся интерпретаторами сущностей базы данных. На диаграмме приведены классы, относящиеся к логическому представлению организационной структуры и персональной информации. Остальные классы, например, используемые для составления расписания синхронизации, на диаграмме не представлены.

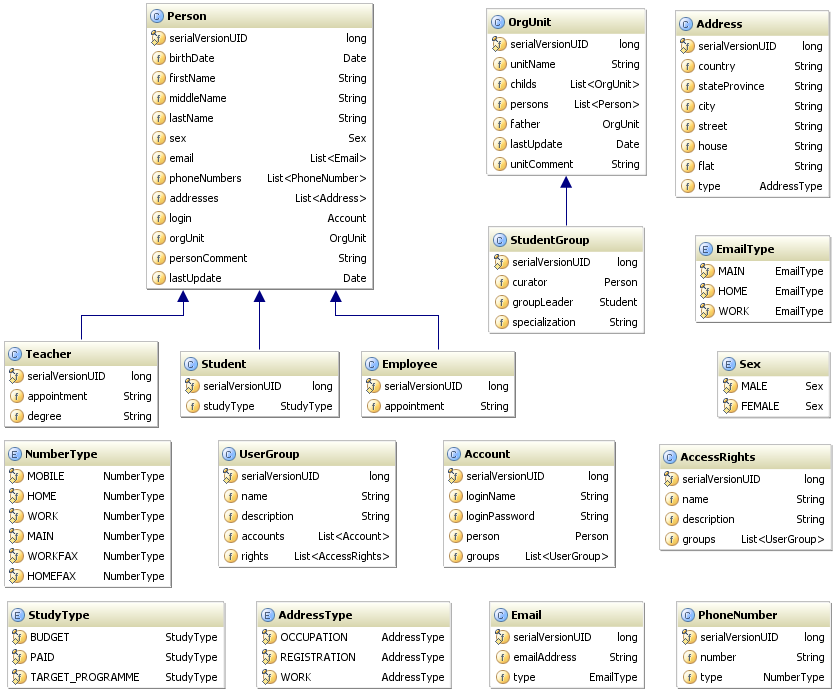
****

Рис. 3. Диаграмма классов–сущностей базы данных

## §4. Подмодули системы

### 4.1. Модуль бизнес логики (EjbCore)

Уровень бизнес логики и взаимодействия с базой данных в разрабатываемом комплексе представлен модулем EjbCore. Технология Enterprise Java Beans [8, 7, 20] обычно применяется, когда требуется как минимум один из следующих сервисов:

* поддержка сохранности данных (Java Persistence API [8, 7, 21, 22]). Данные будут сохранены даже в случае полной остановки программы;
* поддержка распределённых транзакций;
* поддержка конкурентного изменения данных и многопоточность;
* безопасность и ограничение доступа к данным;
* поддержка автоматизированной установки на сервер приложений;
* удалённый доступ.

Согласно требованиям к разрабатываемому комплексу и политике безопасности Юридического факультета большинство описанных сервисов, так или иначе, являются востребованными.

Модуль построен согласно классическому шаблону проектирования на платформе Java EE с применением DAO [23] объектов. Такие объекты предоставляют абстрактный интерфейс к какому-либо типу базы данных и не имеют жесткой связи с конкретной реализацией механизмов хранения.

Другой особенностью модуля является применение объектно-реляционного отображения базы данных (Object-relational mapping). В рамках проекта это – JPA 2.0. Java Persistence API – стандартизированный интерфейс для Java ORM фреймворков. Существует несколько популярных реализаций JPA спецификации: JBoss Hibernate, Eclipse Link, OpenJPA. Безусловно, данный подход имеет свои плюсы и минусы. Однако, нет необходимости приводить подробное сравнение, т.к. это выходит за рамки данной работы.

Подводя итоги можно сказать, что модуль реализован на основе технологий **EJB 3.1** [24, 25]. Работа с базой данных осуществляется строго по спецификации **JPA 2.0** [26]с использованием фреймворка **Eclipse Link 2.1** [27].

На пакетном уровне все классы модуля логически делятся на четыре группы:

* JPA Entities (классы-интерпретаторы сущностей базы данных и классы, соответствующие их полям);
* DAO классы;
* Сервисы;
* Вспомогательные классы:
  + классы с дополнительным описанием сущностей БД;
  + классы системы информационной безопасности;
  + классы, осуществляющие генерирование тестовых данных.

На рисунке 4 представлена диаграмма классов данного модуля, с учетом представленного разбиения.

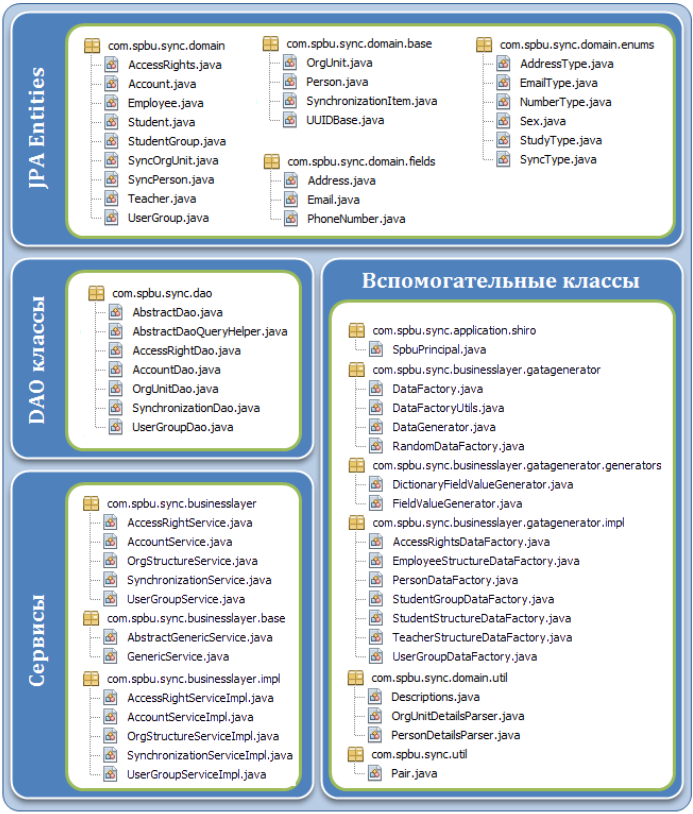
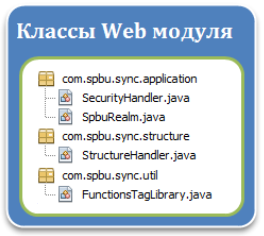


Рис. 4. Диаграмма классов модуля EjbCore (начало)

Более подробно с реализацией модуля бизнес логики можно ознакомиться в приложении 1.

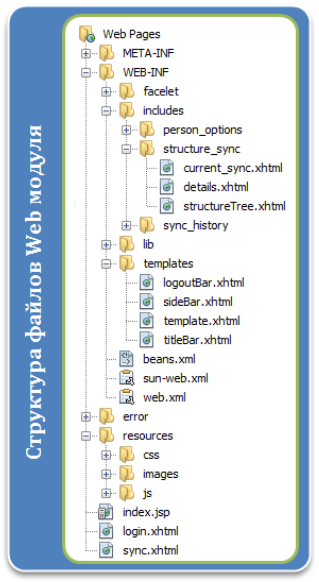
### 4.2. WEB модуль

 В соответствии с функциональными требованиями, которые предъявляются к программному комплексу, пользователь должен иметь возможность выбирать объекты синхронизации. Кроме того, разные группы пользователей имеют ограничения на типы объектов, которые они могут выбирать и синхронизировать. Так же ограничивается доступ к определенным полям этих объектов. Например, студенты не имеют доступа к личным телефонам и адресу проживания преподавателей и сотрудников факультета, в то время как сотрудники отдела кадров имеют доступ к данной информации.

После анализа бизнес процессов было принято решение о введение двух видов синхронизации: ежедневной и единовременной. Такое разделение позволяет обеспечить дополнительную гибкость при работе пользователей с адресной книгой сервисов Google Apps. Вторым преимуществом является уменьшение количества производимых синхронизаций, что влечет увеличение производительности комплекса.

Подводя итог, данный модуль позволяет авторизованным пользователям просматривать разрешенные политикой факультета объекты и осуществлять их синхронизацию. Модуль так же имеет возможность просматривать историю и выполнять повторную синхронизацию.

Модуль реализован с применением следующих технологий:

* **JSF 2.0** [28, 29];
* **PrimeFaces 2.2.1** [30, 31];
* HTML + CSS;
* Ajax.

Для контроля доступа и реализации групповой политики применяется библиотека **Apache Shiro** [32, 33]. Этот фреймворк безопасности Java упрощает аутентификацию, авторизацию, криптографию и управление пользовательскими сессиями. Apache Shiro поддерживает различные механизмы шифрования и протоколы, аутентификацию через LDAP, JDBC, Kerberos, ActiveDirectory и многое другое.

На рисунке 5 представлен графический интерфейс программного комплекса. На приведенном скриншоте, пользователь обладает максимальными правами и может просматривать всю информацию о студентах, преподавателях и сотрудниках. В приложении 2 приведены дополнительные скриншоты демонстрирующие работу программы.

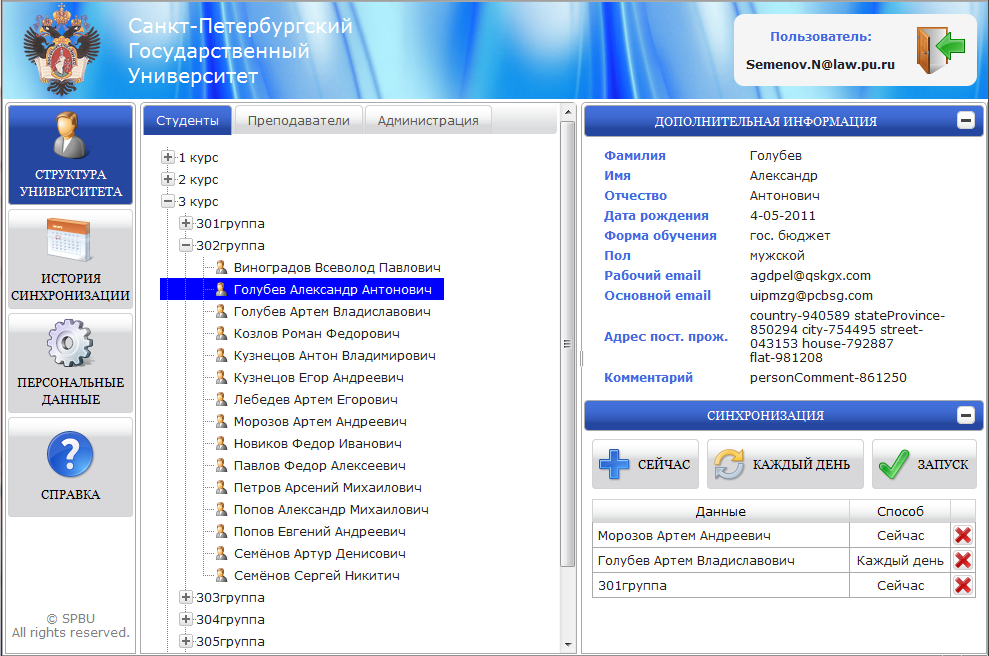
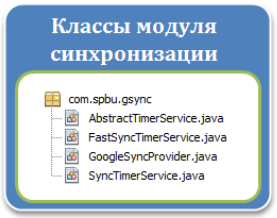


Рис. 5. Пример работы Web модуля

После того как пользователь выбирает интересующие его объекты синхронизации и нажимает кнопку «запуск*»*, Web модуль создает необходимые сценарии синхронизации. Далее посредством EjbCore модуля сценарии записываются в локальную базу данных и инициализируется запуск методов модуля синхронизации.

Более подробно с деталями реализации Web модуля можно ознакомиться в приложении 1.

### 4.3. Модуль синхронизации со службами Google

Для осуществления взаимодействия со службами Google Apps был разработан отдельный модуль. Модуль содержит классы для единовременной и ежедневной синхронизации. Функционально классы несколько отличаются друг от друга, но имеют схожую последовательность действий. Вначале устанавливается соединение с Google Contacts API. После осуществления необходимых проверок создаются соответствующие объекты, и производится заполнение их полей. Далее составляется запрос и выполняется на стороне Google. Запуск синхронизации может быть инициирован Web модулем или внутренним расписанием.

Данный модуль реализован как **EJB 3.1** архив. Для запуска синхронизации в соответствии с внутренним расписанием применяется **Timer Service** (поддерживает возможность ведения нескольких расписаний) [34, 35]. Для упрощения взаимодействия с API Google используется клиентская библиотека **gdata-java-client** [36, 37].

Подробности реализации данного модуля представлены в приложении 1.

### 4.4. Модуль репликации БД

Согласно разработанной архитектуре, необходимо осуществлять синхронизацию базы данных Юридического факультета и локальной базы данных. Кроме того, требуется механизм, который будет выполнять первоначальное наполнение локальной БД. За выполнение этих двух операций отвечает модуль репликации.

Предполагается **два типа синхронизации**: автоматическая и ручная. Автоматическая синхронизация запускается согласно составленному расписанию, например, каждую ночь в 3:00. В ряде случаев необходимо, чтобы изменения, производимые в базе данных факультета, мгновенно были доступны всем пользователям разрабатываемого комплекса. Для решения этой задачи предусмотрен ручной запуск механизмов синхронизации.

С точки зрения реализации, модуль представляет собой **EJB 3.1** архив. Запуск синхронизации по расписанию осуществляется путем использования **Timer Service**. Модулю необходимо производить взаимодействие с двумя базами данных. Взаимодействие с локальной базой происходит посредством модуля бизнес логики (EjbCore). Для осуществления взаимодействия с базой данных факультета применяется **JDBC** (Java Database Connectivity) [38, 39]. JDBC позволяет разрабатывать модуль, не опираясь на специфику конкретной базы данных. Причем в случае перехода на другую базу данных, придется изменить только файл конфигурации, а код модуля останется прежним.

## §5. Достигнутые результаты

В ходе работы над созданием прототипа модуля синхронизации персональных данных были достигнуты следующие результаты:

* Создана инфраструктура проекта и добавлены все необходимые зависимости и внешние библиотеки;
* Полностью реализован модуль бизнес логики;
  + Все производимые с базой данных операции осуществляются через транзакции посредством **Java Transaction API** [40];
* Web модуль реализован на 90%. Для всех оставшихся частей (просмотра истории и повторного запуска синхронизации) созданы заготовки страниц и классов. Пользователь имеет возможность выбирать «разрешенные» ему объекты, просматривать их информацию и осуществлять запуск синхронизации;
  + Безопасное соединение между клиентом и сервером осуществляется через протокол SSL [41];
  + Внедрена ролевая модель доступа ко всем ресурсам модуля. Созданы базовые группы и соответствующие им права. Разработанные механизмы позволяют настраивать доступ максимально гибко: например, можно запретить студентам просматривать личные телефоны преподавателей, но открыть доступ к рабочим;
* Полностью реализован модуль синхронизации с сервисами Google;
* На момент написания работы модуль репликации БД не реализован. Такое решение было принято, поскольку для создания спецификации и технических документов анализ данного модуля является второстепенным и может внести лишь незначительные коррективы;
* Реализованы базовые механизмы шифрования на уровне базы данных;
* В модуль бизнес логики внедрен генератор тестовых данных;
  + Генератор позволяет создать организационную структуру факультета и заполнить её объектами;
  + При генерировании объектов так же инициализируются и заполняются все соответствующие им поля;
* Локальная база данных создается и наполняется непосредственно из разработанного модуля.

# ГЛАВА 3. Тестирование модуля синхронизации

Тестирование является неотъемлемым этапом процесса разработки программного обеспечения. На этом этапе выявляется большинство дефектов функционирования анализируемой программы. Речь идет о разработанном прототипе модуля синхронизации. Необходимо убедится в правильности выбранной концепции и работоспособности предложенной архитектуры.

## §1. Анализ работа приложения на тестовых данных

Для проведения тестирования пользовательского интерфейса и проверки системы в целом необходимо обеспечить программный комплекс необходимыми данными. Так как разработанный прототип не обладает достаточным уровнем информационной безопасности, использовать реальные персональные данные категорически запрещается. Одним из выходов из сложившейся ситуации является имитации данных в достаточном объеме. Иными словами необходимо сгенерировать персональные данные предполагаемого количества пользователей.

Для наполнения локальной базы данных модуль бизнес логики (EjbCore) был оснащен генератором, позволяющим создавать произвольное количество объектов, а так же наполнять их «адекватными» данными. Т.е. не произвольными строками и цифрами, а понятными человеку лексическими конструкциями: например, «Голубев Александр Антонович» вместо «sbgkbhubsfasd». Для каждого пользователя можно сгенерировать адреса проживания и регистрации; домашний, рабочий и мобильный телефоны; рабочий и личный адресы электронной почты; а так же все остальные поля, согласно диаграмме классов на рисунке 3.

Генератор был разработан согласно организационной структуре факультета и обеспечивает создание следующих данных:

* **2150** пользователей, из которых:
  + 1200 студентов;
  + 300 преподавателей;
  + 650 сотрудников;
* **157** структурных подразделений, из которых:
  + 48 академических групп;
  + 20 учебных кафедр;
  + 89 административных отделов;
* **4500** телефонных номеров;
* **4500** адресов временного и постоянного проживании;
* **4500** адресов электронной почты.

По результатам тестирования программного комплекса на тестовых данных можно сделать следующие выводы:

* реализованы все функциональные требования, предъявляемые к разработанному модулю;
* проблем с отображением графического интерфейса и обработкой скриптов в последних версиях Opera, Firefox и Explorer обнаружено не было;
* синхронизацию с адресной книгой сервисов Google проходит практически моментально (в пределах секунды).

Стоит отметить, что минимальное разрешение монитора, обеспечивающее комфортную работу с приложением, составляет 1024х768. Возможности просмотр приложения с мобильных устройств ограничены.

## §2. Профилирование системы

Для выявления требуемых вычислительных мощностей, необходимых для нормального функционирования системы, было решено провести поверхностное профилирование программного комплекса. Из всевозможных вариантов был выбран встроенный в GlassFish v3.1 профилировщик – в рамках исследования его использование наиболее оправдано.

На рисунках 6 и 7 представлена информация о расходуемом процессорном времени, количестве запросов и ошибок, объеме расходуемой оперативной памяти.

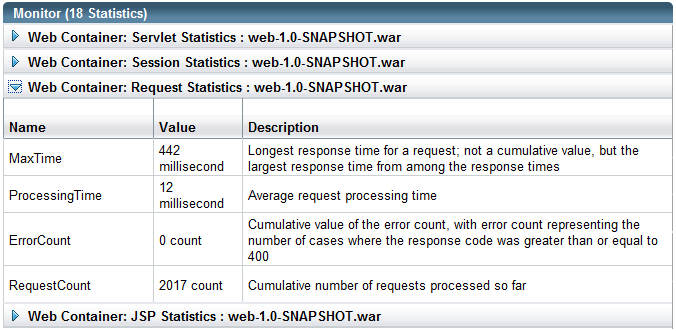


Рис. 6. Скриншот профилировщика - Web модуль

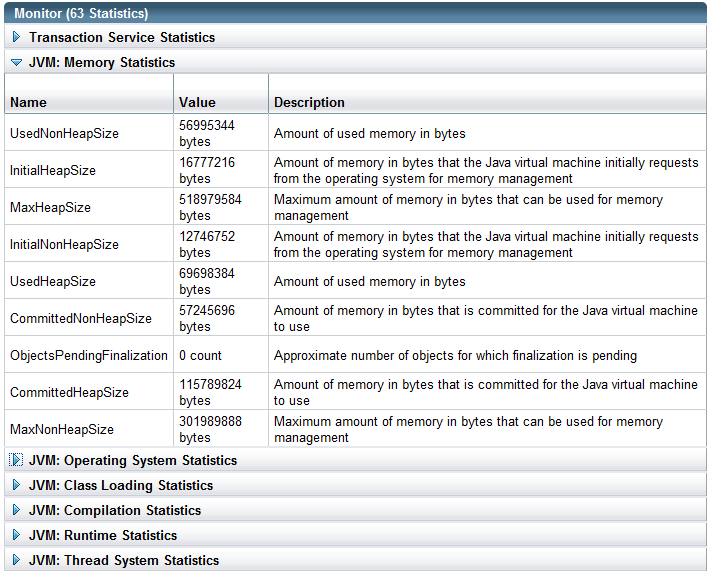


Рис. 7. Скриншот профилировщика - Java Virtual Machine

Дополнительные сведения были собраны при помощи Firebug (плагина для браузера Firefox). Анализ показал, что объем загружаемой страницы достаточно велик и составляет 902,2 KB (в основном это файлы JavaScript, реализующие динамические элементы интерфейса). Однако 866,2 KB загружается из кеша и последующая загрузка этих данных не требуется. Все взаимодействие с сервером осуществляется через Ajax, постоянное обновление страницы не требуется. Применение асинхронного взаимодействия, позволяет значительно снизить нагрузку на сервер и повысить удобство пользовательского интерфейса.

## §3. Проверка возможности масштабирования системы

Ввиду того, что разработанный прототип имеет модульную архитектуру, вопрос о масштабируемость решается достаточно просто. Можно выделить два основных способа:

1. отдельные модули системы разнести по разным серверам приложений;
2. запустить приложение в кластере серверов приложений.

Для реализации первого способа достаточно внести небольшие изменения в конфигурационные файлы модулей и добавить ряд дополнительных аннотации. В общем и целом данной подход хорошо себя зарекомендовал и активно применяется в высоконагруженных проектах.

Гораздо больший интерес представляет второй способ. С его помощью можно сколь угодно гибко варьировать нагрузку. Например, можно создать два экземпляра Web-модуля и по одному экземпляру остальных модулей. В рамках работы нет необходимости приводить детали кластеризации серверов приложений, описывать достоинства и недостатки. Цель исследования несколько иная – убедиться в возможности запуска приложения на кластере GlassFish v3.1.

На рисунке 8 приведена техническая база исследования. На компьютере с операционной системой Windows 7 Professional был развернут сервер приложений GlassFish v3.1. Был создан кластер «SPBU-Cluster» и три виртуальных сервера (в терминологии сервера – instances), располагающиеся на том же компьютере. Потенциально, кластер можно развернуть на нескольких компьютер, однако нет никаких принципиальных отличий. В этом случае, всю «ответственность» за поддержание целостности кластера берет на себя один из серверов.

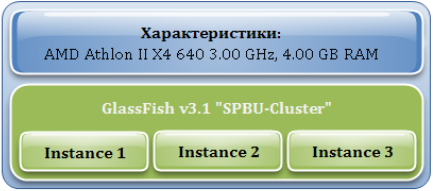


Рис. 8. Кластер GlassFish v3.1

После настройки кластера разработанное приложение было успешно развернуто. Никаких проблем с пользовательским интерейсом и ошибок функционирования выявлено не было. Таким образом можно констатировать, что **возможен полноценный запуск на кластере** GlassFish v3.1. Кроме того, применением плагина Load Balancer [42] можно добиться более гибкого управления входящим трафиком.

## §4. Выводы

В ходе работы было проведено минимальное тестирование разработанного прототипа и его профилирование. Полученное решение полностью удовлетворяет предъявляемым функциональным требованиям и обладает необходимым потенциалом гибкости и масштабируемости. Возможна независимая модификация отдельных модулей системы и распределенный запуск программного комплекса.

Стоит отметить, что прототип не является полноценной системой: он не оснащен надлежащей системой безопасности и имеет ряд узких мест. Одним из таких мест является уровень представления, т.е. JSF компоненты. Основная часть процессорного времени тратится на обработку JSF сервлетов и фильтров. Учитывая, что разработанный комплекс спроектирован с использованием шаблона Model-View-Controller (MVC), можно достаточно легко заменить JSF на легковесные компоненты. Например, применение фреймворка Apache Click [43] позволит уменьшить время ответа сервера в несколько раз. Применение только сервлетов позволит получить ещё больший выигрыш производительности. Однако, возникает и ряд недостатков связанных с ручной обработкой Ajax запросов, контролем сессий и т.д.

Как показывает практика, разработчику всегда приходится выбирать и искать компромисс. С одной стороны - скоростью, удобство, гибкость разработки, абстрактные компоненты и событийная модель взаимодействия, легкость тестирования и сопровождения кода. С другой стороны – высокая производительность и низкий расход оперативной памяти. Выбор JSF и PrimeFaces в нашем случае был обусловлен спецификой задачи – требовалось разработать прототип для последующего анализа и составления технической документации.

Развитие прототипа до эксплуатационной системы в настоящий момент не планируется. В рамках данной работы тестирование модуля синхронизации является завершающим этапом разработки.

# ЗАКЛЮЧЕНИЕ

В рамках проекта по внедрению почтовых служб Google Apps в инфраструктуру Юридического факультета Санкт-Петербургского Государственного Университета был проведен анализ требований и поиск оптимального решения. Разработка собственного программного комплекса в нашем случае наиболее оправдана. Однако, такое решение является сложным и дорогостоящим мероприятием. Требуется детализированное техническое задание и подробная спецификация. Для составления необходимой документации было принято решение о применении технологий прототипирования.

В ходе работы был разработан прототип модуля синхронизации, реализующий необходимый функционал, и проведено его тестирование. Исходные коды программного комплекса переданы руководителю службы информационных технологий Юридического факультета для проведения дальнейшей экспертизы.

# Список литературы и источников

1. Материалы официального сайта компании Gartner Group - http://www.gartner.com
2. Google Apps Directory Sync – Руководство по администрированию служб Google
3. Google Apps Migration for Microsoft® Exchange – Справка Google Apps Администратор (https://www.google.com/support/a/bin/answer.py ?hl=ru&answer=154630)
4. Google Apps Migration for Lotus Notes – Справка Google Apps Администратор (https://www.google.com/support/a/bin/answer.py ?hl=ru&answer=172212)
5. Официальный сайт [Microsoft Exchange Server](http://www.google.ru/search?hl=ru&newwindow=1&client=firefox-a&hs=IdO&rls=org.mozilla:en-US:official&sa=X&ei=tfjITe2gG4Gc-wb3wYDTBg&ved=0CD4QBSgA&q=Microsoft+Exchange+Server&spell=1) - http://www.microsoft. com /exchange/2010/ru/ru/default.aspx
6. Техническая документация по платформе Java EE 6 - http://download.oracle.com/ javaee/
7. [A. Goncalves](http://www.amazon.com/s/ref=ntt_athr_dp_sr_1?_encoding=UTF8&sort=relevancerank&search-alias=books&field-author=Antonio%20Goncalves). Beginning Java EE 6 Platform with GlassFish 3: From Novice to Professional, second edition. Apress. 2010
8. The Java EE 6Tutorial. Oracle. August 2010
9. Материалы официального сайта компании OSGi Alliance - http://www.osgi.org/Main/HomePage
10. Richard Hall, Karl Pauls, Stuart McCulloch, David Savage. OSGi in Action: Creating Modular Applications in Java. Apr 28, 2011.
11. Документация по Apache Felix 3.2.1 - http://felix.apache.org/site/ documentation.html
12. Документация по Equinox 3.2 - http://eclipse.org/equinox/ documents/
13. Oracle GlassFish Server 3.0.1 Embedded Server Guide. Oracle. June 2010
14. Официальный сайт [Apache Ant](http://www.google.ru/search?hl=ru&newwindow=1&client=firefox-a&hs=3z3&rls=org.mozilla:en-US:official&sa=X&ei=OvnITcWdKNHKswbd8eSHAw&ved=0CBcQBSgA&q=Apache+Ant&spell=1)   - http://ant.apache.org/
15. Официальный сайт [Apache Maven](http://www.google.ru/search?hl=ru&newwindow=1&client=firefox-a&hs=3z3&rls=org.mozilla:en-US:official&sa=X&ei=OvnITcWdKNHKswbd8eSHAw&ved=0CBcQBSgA&q=Apache+Ant&spell=1)  - http://maven.apache.org/
16. Maria Odea Ching, Brett Porter. Apache Maven 2 Effective Implementation: Build and manage applications with Maven, Continuum, and Archiva. Packt Publishing. 2009
17. Документация и инструкции по внедрению Maven 3.0 - http://maven.apache.org/guides/index.html
18. Материалы сайта разработчиков IBM - http://www.ibm.com/ developerworks/ru/edu/j-mavenv2
19. Документация по PosgreSQL 9.0 - http://www.postgresql.org/ docs/9.0/static/index.html
20. Официальная спецификация по технологии EJB - http://www.oracle.com/technetwork/java/docs-135218.html
21. Официальная спецификация по технологии JPA - http://www.oracle.com/ technetwork/java/javaee/tech/persistence-jsp-140049.html
22. M. Keith. Pro JPA 2: Mastering the Java(TM) Persistence API. Apress. 2009
23. Паттерны проектирования на Core J2EE (Data Access Object) - http://java.sun.com/blueprints/corej2eepatterns/Patterns/ DataAccessObject.html
24. Построение приложений на EJB 3.1. Статьи и материалы сайта - java.dzone.com/articles
25. Технология разработки бизнес компонентов - http://netbeans.org/ kb/docs/javaee/javaee-entapp-ejb\_ru.html
26. Руководство по JPA 2.0 - http://www.vogella.de/articles/ JavaPersistenceAPI/article.html.
27. Официальный сайт EclipseLink - http://www.eclipse.org/eclipselink
28. Официальное описание технологии JSF - http://www.oracle.com/ technetwork/java/javaee/javaserverfaces-139869.html
29. Статьи и материалы сообщества JSF - http://javaserverfaces.java.net
30. Официальный сайт PrimeFaces - http://www.primefaces.org
31. Çağatay Çivici. PrimeFaces 2.2 Users Guide. 05.02.2011
32. Официальный сайт Apache Shiro - http://shiro.apache.org
33. API Apache Shiro - http://shiro.apache.org/static/current/apidocs
34. Описание технологии EJB 3.1 Timer Service - http://download.oracle. com/docs/cd/E18930\_01/html/821-2418/beahw.html
35. API Timer Service - http://download.oracle.com/javaee/6/api/javax/ ejb/TimerService.html
36. Документация по Java Client Library - http://code.google.com/p/gdata-java-client/
37. Инструкции по внедрению Java Client Library - http://code.google. com/p/gdata-java-client
38. Документация по JDBC - http://download.oracle.com/javase/6/ docs/technotes/guides/jdbc
39. JDBC драйвер для PostgreSQL - http://jdbc.postgresql.org/
40. Официальная информация по Java Transaction API - http://www. oracle.com/technetwork/java/javaee/jta/index.html
41. SSL сертификаты - http://www.ssl.com
42. Административная панель Load Balancer - http://glassfish.java.net/ javaee5/lb-admin
43. Официальный сайт Apache Click - http://click.apache.org/

# Приложение 1. Реализация модулей

Ниже приведены исходные коды некоторых классов разработанного программного комплекса. Объем кода слишком велик и в рамках данной работы привести его полностью не представляется возможным.

C:\Program Files\Microsoft Office\MEDIA\OFFICE12\Lines\BD15155_.gif

package com.spbu.sync.businesslayer.base;

import com.spbu.sync.dao.AbstractDao;

import com.spbu.sync.domain.base.UUIDBase;

import java.util.List;

/\*\*

\* @author nsemenov

\*/

public abstract class AbstractGenericService<T extends UUIDBase> implements GenericService<T> {

@Override

public List<T> getData() {

return getMainDao().findAll();

}

@Override

public List<T> getFilteredData(int startIndex, int count, String sortField, boolean sortOrder) {

return getMainDao().findRange(startIndex, count, sortField, sortOrder);

}

@Override

public List<T> getFilteredData(String sortField, boolean sortOrder) {

return getMainDao().findAll(sortField, sortOrder);

}

@Override

public long count() {

return getMainDao().count();

}

@Override

public T save(T data) {

return getMainDao().edit(data);

}

@Override

public void delete(T data) {

getMainDao().remove(data);

}

@Override

public T get(String entityId) {

return getMainDao().find(entityId);

}

protected abstract AbstractDao<T> getMainDao();

}

C:\Program Files\Microsoft Office\MEDIA\OFFICE12\Lines\BD15155_.gif

package com.spbu.sync.businesslayer.base;

import com.spbu.sync.domain.base.UUIDBase;

import java.util.List;

/\*\*

\* Generic interface for non business object related services.

\*

\* @author nsemenov

\*/

public interface GenericService<T extends UUIDBase> {

List<T> getData();

List<T> getFilteredData(int startIndex, int count, String sortField, boolean sortOrder);

List<T> getFilteredData(String sortField, boolean sortOrder);

long count();

T save(T data);

void delete(T data);

T get(String entityId);

}

C:\Program Files\Microsoft Office\MEDIA\OFFICE12\Lines\BD15155_.gif

package com.spbu.sync.businesslayer.impl;

import com.spbu.sync.businesslayer.UserGroupService;

import com.spbu.sync.businesslayer.base.AbstractGenericService;

import com.spbu.sync.dao.AbstractDao;

import com.spbu.sync.dao.AccountDao;

import com.spbu.sync.dao.UserGroupDao;

import com.spbu.sync.domain.AccessRights;

import com.spbu.sync.domain.Account;

import com.spbu.sync.domain.UserGroup;

import java.util.List;

import javax.ejb.EJB;

import javax.ejb.Stateless;

/\*\*

\*

\* @author Nick Semenov <nsemenov@reksoft.ru>

\*/

@Stateless

public class UserGroupServiceImpl extends AbstractGenericService<UserGroup> implements UserGroupService {

@EJB

private UserGroupDao userGroupDao;

@EJB

private AccountDao accountDao;

@Override

public List<Account> getAccountsForGroup(String groupId) {

return accountDao.getAccountsForGroup(groupId);

}

@Override

public List<AccessRights> getAccessRightsForGroup(String groupId) {

return userGroupDao.getAccessRightsForGroup(groupId);

}

@Override

public List<UserGroup> getGroupsForAccount(String accountId) {

return userGroupDao.getGroupsForAccount(accountId);

}

@Override

protected AbstractDao<UserGroup> getMainDao() {

return userGroupDao;

}

}

C:\Program Files\Microsoft Office\MEDIA\OFFICE12\Lines\BD15155_.gif

package com.spbu.sync.dao;

import com.spbu.sync.domain.base.UUIDBase;

import java.util.List;

import javax.annotation.PostConstruct;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

/\*\*

\*

\* @author nsemenov

\*/

public abstract class AbstractDao<T extends UUIDBase> {

@PersistenceContext

private EntityManager entityManager;

private final Class<T> entityClass;

private AbstractDaoQueryHelper<T> queryHelper;

public AbstractDao(Class<T> entityClass) {

this.entityClass = entityClass;

}

@PostConstruct

protected void postConstruct() {

queryHelper = createQueryHelper();

}

protected EntityManager getEntityManager() {

return entityManager;

}

protected Class<T> getEntityClass() {

return entityClass;

}

public void create(T entity) {

getEntityManager().persist(entity);

}

public T edit(T entity) {

return getEntityManager().merge(entity);

}

public void refresh(T entity) {

getEntityManager().refresh(entity);

}

public void remove(T entity) {

getEntityManager().remove(getEntityManager().merge(entity));

}

public T find(Object id) {

return getEntityManager().find(entityClass, id);

}

public T getReference(Object id) {

return getEntityManager().getReference(entityClass, id);

}

public List<T> findAll() {

return queryHelper.createAllQuery().getResultList();

}

public List<T> findAll(String sortField, boolean ascending) {

return findRange(0, Integer.MAX\_VALUE, sortField, ascending);

}

public List<T> findRange(int startIndex, int count) {

return queryHelper.createRangeQuery(startIndex, count).getResultList();

}

public List<T> findRange(int startIndex, int count, String sortField, boolean ascending) {

return queryHelper.createRangeQuery(startIndex, count, sortField, ascending).getResultList();

}

public long count() {

return queryHelper.createCountQuery().getSingleResult();

}

private AbstractDaoQueryHelper<T> createQueryHelper() {

return new AbstractDaoQueryHelper<T>(getEntityManager(), entityClass);

}

}

C:\Program Files\Microsoft Office\MEDIA\OFFICE12\Lines\BD15155_.gif

package com.spbu.sync.dao;

import com.spbu.sync.domain.base.UUIDBase;

import javax.persistence.EntityManager;

import javax.persistence.TypedQuery;

import org.apache.commons.lang3.StringUtils;

/\*\*

\* Helper class for constructing generic queries used in abstract dao.

\*

\* @author nsemenov

\*/

class AbstractDaoQueryHelper<T extends UUIDBase> {

private static final String COUNT\_QUERY = "SELECT count(e) FROM %s e";

private static final String FIND\_ALL\_QUERY = "SELECT e FROM %s e";

private static final String ORDER\_FIELD\_PARAMETER = "sortField";

private static final String ORDER\_CONDITION = "ORDER BY :" + ORDER\_FIELD\_PARAMETER + " %s";

private final EntityManager entityManager;

private final Class<T> entityClass;

private final String findAllQuery;

private final String findAllOrderedQuery;

private final String countQuery;

public AbstractDaoQueryHelper(EntityManager entityManager, Class<T> entityClass) {

this.entityManager = entityManager;

this.entityClass = entityClass;

findAllQuery = String.format(FIND\_ALL\_QUERY, entityClass.getSimpleName());

findAllOrderedQuery = findAllQuery + " " + ORDER\_CONDITION;

countQuery = String.format(COUNT\_QUERY, entityClass.getSimpleName());

}

public TypedQuery<Long> createCountQuery() {

TypedQuery<Long> query = entityManager.createQuery(countQuery, Long.class);

return query;

}

public TypedQuery<T> createAllQuery() {

TypedQuery<T> query = entityManager.createQuery(findAllQuery, entityClass);

return query;

}

public TypedQuery<T> createRangeQuery(int startIndex, int count) {

return createRangeQuery(startIndex, count, null, true);

}

public TypedQuery<T> createRangeQuery(int startIndex, int count, String sortField, boolean ascending) {

TypedQuery<T> query;

if (StringUtils.isNotBlank(sortField)) {

query = entityManager.createQuery(String.format(findAllOrderedQuery, ascending ? "ASC" : "DESC"),

entityClass);

query.setParameter(ORDER\_FIELD\_PARAMETER, sortField);

} else {

query = entityManager.createQuery(findAllQuery, entityClass);

}

query.setFirstResult(startIndex);

query.setMaxResults(count);

return query;

}

}

C:\Program Files\Microsoft Office\MEDIA\OFFICE12\Lines\BD15155_.gif

package com.spbu.sync.dao;

import com.spbu.sync.domain.base.SyncItem;

import com.spbu.sync.domain.enums.SyncType;

import java.util.List;

import javax.ejb.Stateless;

import javax.persistence.EntityManager;

import javax.persistence.TypedQuery;

/\*\*

\*

\* @author nsemenov

\*/

@Stateless

public class SynchronizationDao extends AbstractDao<SyncItem> {

public List<SyncItem> getSyncItemsForAccount(String accountId) {

String jqpl = "Select a from SyncItem a where a.targerAccount = :targerAccount";

TypedQuery<SyncItem> tq = getEntityManager().createQuery(jqpl, SyncItem.class);

tq.setParameter("targerAccount", accountId);

return tq.getResultList();

}

public List<SyncItem> getSyncItemsWithType(SyncType type) {

String jqpl = "Select a from SyncItem a where a.syncType = :syncType";

TypedQuery<SyncItem> tq = getEntityManager().createQuery(jqpl, SyncItem.class);

tq.setParameter("syncType", type);

return tq.getResultList();

}

public SynchronizationDao() {

super(SyncItem.class);

}

@Override

public EntityManager getEntityManager() {

return super.getEntityManager();

}

}

C:\Program Files\Microsoft Office\MEDIA\OFFICE12\Lines\BD15155_.gif

package com.spbu.sync.businesslayer.gatagenerator;

import com.spbu.sync.businesslayer.gatagenerator.generators.FieldValueGenerator;

import com.spbu.sync.domain.base.UUIDBase;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.Map;

/\*\*

\* Base class for generators using random data sets

\*

\* @author nsemenov

\*/

public class RandomDataFactory<T extends UUIDBase> extends DataFactory<T> {

private static final int DEFAULT\_AMOUNT = 10;

public RandomDataFactory(T template) {

super(template);

}

@Override

public List<T> generateData() {

return generateData(DEFAULT\_AMOUNT, getDefaultFieldValueGenerators());

}

public Map<String, FieldValueGenerator<?>> getDefaultFieldValueGenerators() {

return Collections.<String, FieldValueGenerator<?>>emptyMap();

}

public List<T> generateData(int amount, String... skipProperties) {

return generateData(amount, getDefaultFieldValueGenerators(), skipProperties);

}

public List<T> generateData(String... skipProperties) {

return generateData(DEFAULT\_AMOUNT, getDefaultFieldValueGenerators(), skipProperties);

}

public List<T> generateData(Map<String, FieldValueGenerator<?>> customFields, String... skipProperties) {

return generateData(DEFAULT\_AMOUNT, customFields, skipProperties);

}

public List<T> generateData(int amount, Map<String, FieldValueGenerator<?>> customFields, String... skipProperties) {

List<T> result = new ArrayList<T>();

for (int i = 0; i < amount; i++) {

T bean = DataFactoryUtils.populateBean(getTemplate(), customFields, skipProperties);

normalizeBean(bean);

result.add(bean);

}

return result;

}

/\*\*

\* Used to normalize/populate sensitive data like date ranges

\* embedded objects etc

\*

\* @param bean

\*/

protected void normalizeBean(T bean) {

}

}

C:\Program Files\Microsoft Office\MEDIA\OFFICE12\Lines\BD15155_.gif

package com.spbu.sync.businesslayer.gatagenerator.impl;

import com.spbu.sync.businesslayer.gatagenerator.RandomDataFactory;

import com.spbu.sync.domain.Employee;

import com.spbu.sync.domain.base.OrgUnit;

import com.spbu.sync.domain.base.Person;

import java.util.ArrayList;

import java.util.List;

public class EmployeeStructureDataFactory extends RandomDataFactory<OrgUnit> {

private OrgUnit root;

public static final int UNIT\_AMOUNT = 5;

private static final int MAX\_LEVEL = 5;

private static final int PERSONS\_AMOUNT = 10;

private int number = 1;

private int level = 1;

public EmployeeStructureDataFactory(OrgUnit template, OrgUnit root) {

super(template);

this.root = root;

}

public EmployeeStructureDataFactory(OrgUnit template, OrgUnit root, int level, int number) {

this(template, root);

this.level = level;

this.number = number;

}

@Override

protected void normalizeBean(OrgUnit bean) {

bean.setFather(root);

bean.setUnitName("Отдел №" + number);

if (level < MAX\_LEVEL) {

level++;

bean.setChilds(new ArrayList<OrgUnit>(

new EmployeeStructureDataFactory(new OrgUnit(), bean, level, number \* 1 \* 10 + 1).generateData(UNIT\_AMOUNT)));

} else {

List<Employee> employees = new PersonDataFactory<Employee>(new Employee(), bean).generateData(PERSONS\_AMOUNT);

bean.setPersons(new ArrayList<Person>(employees));

}

number++;

}

}

C:\Program Files\Microsoft Office\MEDIA\OFFICE12\Lines\BD15155_.gif

package com.spbu.sync.businesslayer.gatagenerator.impl;

import com.spbu.sync.businesslayer.gatagenerator.RandomDataFactory;

import com.spbu.sync.businesslayer.gatagenerator.generators.DictionaryFieldValueGenerator;

import com.spbu.sync.domain.base.OrgUnit;

import com.spbu.sync.domain.base.Person;

import com.spbu.sync.domain.enums.EmailType;

import com.spbu.sync.domain.fields.Address;

import com.spbu.sync.domain.fields.Email;

import com.spbu.sync.domain.fields.PhoneNumber;

public class PersonDataFactory<T extends Person> extends RandomDataFactory<T> {

private OrgUnit root;

public PersonDataFactory(T template, OrgUnit root) {

super(template);

this.root = root;

}

@Override

protected void normalizeBean(T bean) {

bean.setOrgUnit(root);

bean.setFirstName(DictionaryFieldValueGenerator.FIRST\_NAMES.generate());

bean.setMiddleName(DictionaryFieldValueGenerator.MIDDLE\_NAMES.generate());

bean.setLastName(DictionaryFieldValueGenerator.LAST\_NAMES.generate());

Email email = new Email();

email.setType(EmailType.WORK);

bean.setEmail(new RandomDataFactory<Email>(email).generateData(2));

bean.setAddresses(new RandomDataFactory<Address>(new Address()).generateData(2));

bean.setPhoneNumbers(new RandomDataFactory<PhoneNumber>(new PhoneNumber()).generateData(2));

}

}

C:\Program Files\Microsoft Office\MEDIA\OFFICE12\Lines\BD15155_.gif

package com.spbu.sync.application;

import com.spbu.sync.application.shiro.SpbuPrincipal;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import javax.naming.InitialContext;

import javax.naming.NamingException;

import org.apache.commons.lang3.ObjectUtils;

import org.apache.shiro.authc.AuthenticationInfo;

import org.apache.shiro.authc.AuthenticationToken;

import org.apache.shiro.authc.IncorrectCredentialsException;

import org.apache.shiro.authc.SimpleAuthenticationInfo;

import org.apache.shiro.authc.UnknownAccountException;

import org.apache.shiro.authc.UsernamePasswordToken;

import org.apache.shiro.authz.AuthorizationInfo;

import org.apache.shiro.authz.SimpleAuthorizationInfo;

import org.apache.shiro.cache.MemoryConstrainedCacheManager;

import org.apache.shiro.realm.AuthorizingRealm;

import org.apache.shiro.subject.PrincipalCollection;

import com.spbu.sync.businesslayer.AccountService;

import com.spbu.sync.businesslayer.UserGroupService;

import com.spbu.sync.domain.AccessRights;

import com.spbu.sync.domain.Account;

import com.spbu.sync.domain.UserGroup;

import java.util.ArrayList;

import javax.ejb.EJB;

import javax.ejb.EJBs;

import org.apache.shiro.authc.AuthenticationException;

/\*\*

\* Main realm using core services to autheticate/authorize users.

\*

\* @author Nick Semenov <nsemenov@reksoft.ru>

\*/

@EJBs(value = {

@EJB(name = "AccountService", beanInterface = AccountService.class),

@EJB(name = "UserGroupService", beanInterface = UserGroupService.class)})

public class SpbuRealm extends AuthorizingRealm {

private final AccountService accountService;

private final UserGroupService groupService;

public SpbuRealm() {

super();

try {

accountService = (AccountService) InitialContext.doLookup("java:comp/env/AccountService");

groupService = (UserGroupService) InitialContext.doLookup("java:comp/env/UserGroupService");

} catch (NamingException e) {

throw new RuntimeException("Cannot obtain services", e);

}

setCacheManager(new MemoryConstrainedCacheManager());

setCachingEnabled(Boolean.TRUE);

}

@Override

//This method authenticates a User

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) {

//Get given Password/Username

UsernamePasswordToken token = (UsernamePasswordToken) authenticationToken;

//Get all Users with the same Login Name

List<Account> accounts = accountService.getAccountsByLoginName(token.getUsername());

//No users? -> Throw exception

if (accounts == null || accounts.isEmpty()) {

throw new UnknownAccountException("No account found for user [" + token.getUsername() + "]");

}

List<Account> vaildAccounts = new ArrayList<Account>();

//Remove every user which password does not match

for (int i = 0; i < accounts.size(); i++) {

if (ObjectUtils.equals(String.valueOf(token.getPassword()), accounts.get(i).getLoginPassword())) {

vaildAccounts.add(accounts.get(i));

}

}

//2 Special cases:

// Empty List -> Wrong Password

// More than one Match to password and name -> Exception

if (vaildAccounts.isEmpty()) {

throw new IncorrectCredentialsException("Incorrect password for user [" + token.getUsername() + "]");

} else if (vaildAccounts.size() > 1) {

//Seriously What now :-D

throw new AuthenticationException("More than one User with same Name and Password in Database!");

}

//If were are here we have 1 user with a matching password

return new SimpleAuthenticationInfo(

new SpbuPrincipal(vaildAccounts.get(0).getId(), token.getUsername()),

String.valueOf(token.getPassword()),

getName());

}

@Override

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {

SpbuPrincipal principal = ((SpbuPrincipal) getAvailablePrincipal(principals));

List<UserGroup> userGroups = accountService.getGroupsForAccount(principal.getUserId());

Set<String> roles = new HashSet<String>();

Set<String> permissions = new HashSet<String>();

for (UserGroup userGroup : userGroups) {

roles.add(userGroup.getName());

List<AccessRights> rights = groupService.getAccessRightsForGroup(userGroup.getId());

for (AccessRights accessRight : rights) {

permissions.add(accessRight.getName());

}

}

SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo(roles);

authorizationInfo.setStringPermissions(permissions);

return authorizationInfo;

}

}

C:\Program Files\Microsoft Office\MEDIA\OFFICE12\Lines\BD15155_.gif

package com.spbu.sync.application;

import java.io.IOException;

import java.io.Serializable;

import javax.ejb.EJB;

import javax.enterprise.context.RequestScoped;

import javax.faces.application.FacesMessage;

import javax.faces.context.FacesContext;

import javax.inject.Named;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

import org.apache.shiro.SecurityUtils;

import org.apache.shiro.authc.AuthenticationException;

import org.apache.shiro.authc.IncorrectCredentialsException;

import org.apache.shiro.authc.UnknownAccountException;

import org.apache.shiro.authc.UsernamePasswordToken;

import org.apache.shiro.subject.Subject;

import com.spbu.sync.application.shiro.SpbuPrincipal;

import com.spbu.sync.businesslayer.gatagenerator.DataGenerator;

/\*\*

\*

\* This Bean offers multiple security features for the web application

\*

\* @author Nick Semenov

\*/

@Named

@RequestScoped

public class SecurityHandler implements Serializable {

@EJB

private DataGenerator generator;

private static final long serialVersionUID = -5999893590320159471L;

private final String pageRedirect = "index.jsp?faces-redirect=true";

private String name;

private String password;

private SpbuPrincipal getPrincipal() {

return (SpbuPrincipal) SecurityUtils.getSubject().getPrincipal();

}

// Generate test data

public void generateData() {

generator.generateRandomData();

}

// Returns the current User Name

public String getPrincipalName() {

String result = "Неизвестный пользователь";

try {

result = getPrincipal().getName();

} catch (NullPointerException e) {

System.err.println("NullPointerException in getPrincipalName()");

}

return result;

}

// Checks if user has the access right

public boolean isUserAllowedTo(String permission) {

Subject currentUser = SecurityUtils.getSubject();

return currentUser.isPermitted(permission);

}

// Login of a user

public String login() {

Subject currentUser = SecurityUtils.getSubject();

if (currentUser.isAuthenticated()) {

currentUser.logout();

}

UsernamePasswordToken token = new UsernamePasswordToken(name, password);

try {

currentUser.login(token);

return pageRedirect;

} catch (UnknownAccountException uae) {

FacesContext.getCurrentInstance().addMessage(null,

new FacesMessage(FacesMessage.SEVERITY\_ERROR, "Такой пользователь не зарегистрирован!", null));

} catch (IncorrectCredentialsException ice) {

FacesContext.getCurrentInstance().addMessage(null,

new FacesMessage(FacesMessage.SEVERITY\_ERROR, "Неправильный пароль!", null));

} catch (AuthenticationException lae) {

FacesContext.getCurrentInstance().addMessage(null,

new FacesMessage(FacesMessage.SEVERITY\_ERROR, "Ошибка аутентификации! В доступе отказано", null));

}

return "";

}

// Logout and redirect to LoginPage

public void logout() {

Subject currentUser = SecurityUtils.getSubject();

currentUser.logout();

FacesContext.getCurrentInstance().getExternalContext().invalidateSession();

try {

FacesContext.getCurrentInstance().getExternalContext().redirect(pageRedirect);

} catch (IOException ex) {

System.err.println("IOException during Logout");

}

}

public String getName() {

return this.name;

}

public void setName(String name) {

this.name = name;

}

public String getPassword() {

return this.password;

}

public void setPassword(String password) {

this.password = password;

}

}

C:\Program Files\Microsoft Office\MEDIA\OFFICE12\Lines\BD15155_.gif

package com.spbu.sync.structure;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.List;

import javax.annotation.PostConstruct;

import javax.ejb.EJB;

import javax.enterprise.context.SessionScoped;

import javax.inject.Inject;

import javax.inject.Named;

import org.apache.shiro.SecurityUtils;

import org.primefaces.event.NodeSelectEvent;

import org.primefaces.model.TreeNode;

import com.spbu.sync.application.SecurityHandler;

import com.spbu.sync.application.shiro.SpbuPrincipal;

import com.spbu.sync.businesslayer.AccountService;

import com.spbu.sync.businesslayer.OrgStructureService;

import com.spbu.sync.businesslayer.SynchronizationService;

import com.spbu.sync.domain.Account;

import com.spbu.sync.domain.SyncOrgUnit;

import com.spbu.sync.domain.SyncPerson;

import com.spbu.sync.domain.base.OrgUnit;

import com.spbu.sync.domain.base.Person;

import com.spbu.sync.domain.base.SyncItem;

import com.spbu.sync.domain.enums.SyncType;

import com.spbu.sync.domain.util.OrgUnitDetailsParser;

import com.spbu.sync.domain.util.PersonDetailsParser;

import com.spbu.sync.util.Pair;

/\*\*

\*

\* @author nsemenov

\*/

@Named

@SessionScoped

public class StructureHandler implements Serializable {

private static final long serialVersionUID = 6671359083403367173L;

@Inject

private SecurityHandler securityHandler;

@EJB

private OrgStructureService service;

@EJB

private SynchronizationService syncService;

@EJB

private AccountService accountService;

private TreeNode studentRoot;

private TreeNode teacherRoot;

private TreeNode employeeRoot;

private TreeNode selectedNode = null;

private List<Pair<String, String>> details = new ArrayList<Pair<String, String>>();

private List<SyncItem> syncItems = new ArrayList<SyncItem>();

public void removeItem(int n) {

syncItems.remove(n);

}

public List<SyncItem> getSyncItems() {

return syncItems;

}

public List<Pair<String, String>> getDetails() {

return details;

}

@PostConstruct

protected void postConstruct() {

studentRoot = service.getStudentsOrg();

teacherRoot = service.getTeacherOrg();

employeeRoot = service.getEmployeeOrg();

}

public TreeNode getStudentRoot() {

return studentRoot;

}

public TreeNode getTeacherRoot() {

return teacherRoot;

}

public TreeNode getEmployeeRoot() {

return employeeRoot;

}

public TreeNode getSelectedNode() {

return selectedNode;

}

public void setSelectedNode(TreeNode selectedNode) {

this.selectedNode = selectedNode;

}

public void onNodeSelect(NodeSelectEvent event) {

Object unit = event.getTreeNode().getData();

getAllDetails(unit);

}

private void getAllDetails(Object unit) {

if (unit instanceof Person) {

details = getDetailsForPerson((Person) unit);

} else if (unit instanceof OrgUnit) {

details = getDetailsForOrgUnit((OrgUnit) unit);

} else {

details.clear();

}

}

private List<Pair<String, String>> getDetailsForPerson(Person person) {

List<Pair<String, String>> info = PersonDetailsParser.getDetailsForPerson(person);

List<Pair<String, String>> allowedInfo = new ArrayList<Pair<String, String>>();

for (Pair<String, String> pair : info) {

if (securityHandler.isUserAllowedTo("sync:details:" + pair.getFirst())) {

allowedInfo.add(new Pair<String, String>(PersonDetailsParser.getDescription(pair.getFirst()), pair

.getSecond()));

}

}

return allowedInfo;

}

private List<Pair<String, String>> getDetailsForOrgUnit(OrgUnit orgUnit) {

List<Pair<String, String>> info = OrgUnitDetailsParser.getDetailsForUnit(orgUnit);

List<Pair<String, String>> allowedInfo = new ArrayList<Pair<String, String>>();

for (Pair<String, String> pair : info) {

if (securityHandler.isUserAllowedTo("sync:details:" + pair.getFirst())) {

allowedInfo.add(new Pair<String, String>(OrgUnitDetailsParser.getDescription(pair.getFirst()), pair

.getSecond()));

}

}

return allowedInfo;

}

public void runSynchronization() {

Account account = accountService.get(((SpbuPrincipal) SecurityUtils.getSubject().getPrincipal()).getUserId());

for (SyncItem syncItem : syncItems) {

syncItem.setTargerAccount(account);

syncService.save(syncItem);

}

syncItems.clear();

}

public void syncThisTime() {

sync(SyncType.THIS\_TIME);

}

public void syncEveryTime() {

sync(SyncType.EVERY\_TIME);

}

private void sync(SyncType type) {

if (selectedNode != null) {

Object unit = selectedNode.getData();

if (unit instanceof Person) {

SyncPerson sp = new SyncPerson();

sp.setSyncType(type);

sp.setPersonToSync((Person) selectedNode.getData());

if (!syncItems.contains(sp)) {

syncItems.add(sp);

}

}

if (unit instanceof OrgUnit) {

SyncOrgUnit su = new SyncOrgUnit();

su.setSyncType(type);

su.setUnitToSync((OrgUnit) selectedNode.getData());

if (!syncItems.contains(su)) {

syncItems.add(su);

}

}

}

}

}

C:\Program Files\Microsoft Office\MEDIA\OFFICE12\Lines\BD15155_.gif

package com.spbu.gsync;

import java.util.ArrayList;

import java.util.Comparator;

import java.util.List;

import java.util.TreeSet;

import javax.ejb.EJB;

import com.spbu.sync.businesslayer.SynchronizationService;

import com.spbu.sync.domain.Account;

import com.spbu.sync.domain.SyncOrgUnit;

import com.spbu.sync.domain.SyncPerson;

import com.spbu.sync.domain.base.SyncItem;

import com.spbu.sync.util.Pair;

/\*\*

\*

\* @author Semenov Nikolay

\*/

public class AbstractTimerService {

@EJB

protected SynchronizationService syncService;

protected class SyncComparator implements Comparator<SyncItem> {

@Override

public int compare(SyncItem o1, SyncItem o2) {

return o1.getTargerAccount().getId().equals(o2.getTargerAccount().getId()) ? 1 : -1;

}

}

protected <T extends SyncItem> List<List<T>> splitByAccount(TreeSet<T> items) {

Account currentAccount = null;

List<List<T>> list = new ArrayList<List<T>>();

List<T> accountList = new ArrayList<T>();

for (T syncItem : items) {

if (!syncItem.getTargerAccount().equals(currentAccount)) {

if (currentAccount != null) {

list.add(accountList);

}

accountList = new ArrayList<T>();

currentAccount = syncItem.getTargerAccount();

}

accountList.add(syncItem);

}

list.add(accountList);

return list;

}

protected Pair<List<SyncPerson>, List<SyncOrgUnit>> splitSyncItems(List<SyncItem> items) {

List<SyncPerson> syncPersons = new ArrayList<SyncPerson>();

List<SyncOrgUnit> syncUnits = new ArrayList<SyncOrgUnit>();

for (SyncItem item : items) {

if (item instanceof SyncPerson) {

syncPersons.add((SyncPerson) item);

} else if (item instanceof SyncOrgUnit) {

syncUnits.add((SyncOrgUnit) item);

}

}

return Pair.of(syncPersons, syncUnits);

}

}

C:\Program Files\Microsoft Office\MEDIA\OFFICE12\Lines\BD15155_.gif

package com.spbu.gsync;

import java.util.List;

import java.util.TreeSet;

import javax.ejb.Schedule;

import javax.ejb.Singleton;

import javax.ejb.Stateless;

import com.spbu.sync.domain.SyncOrgUnit;

import com.spbu.sync.domain.SyncPerson;

import com.spbu.sync.domain.base.SyncItem;

import com.spbu.sync.domain.enums.SyncType;

import com.spbu.sync.util.Pair;

/\*\*

\*

\* @author Semenov Nikolay

\*/

@Singleton

public class FastSyncTimerService extends AbstractTimerService {

@Schedule(second = "\*", minute = "\*", hour = "\*")

public void RunGoogleAppsSync() {

List<SyncItem> itemsFromService = syncService.getSyncItemsWithThisTimeType();

itemsFromService.addAll(syncService.getSyncItemsWithEveryTimeType());

Pair<List<SyncPerson>, List<SyncOrgUnit>> personsAndUnits = splitSyncItems(itemsFromService);

;

// Prepare GoogleSyncProvider

GoogleSyncProvider syncProvider = new GoogleSyncProvider();

// Prepare Persons to sync

TreeSet<SyncPerson> syncPersonsSet = new TreeSet<SyncPerson>(new SyncComparator());

syncPersonsSet.addAll(personsAndUnits.getFirst());

List<List<SyncPerson>> syncPersons = splitByAccount(syncPersonsSet);

for (List<SyncPerson> syncPersonList : syncPersons) {

if (!syncPersonList.isEmpty()) {

syncProvider.connect(syncPersonList.get(0).getTargerAccount().getLoginName(), syncPersonList.get(0)

.getTargerAccount().getLoginPassword());

syncProvider.runPersonSync(syncPersonList);

}

}

// Prepare OrgUnits to sync

TreeSet<SyncOrgUnit> syncUnitsSet = new TreeSet<SyncOrgUnit>(new SyncComparator());

syncUnitsSet.addAll(personsAndUnits.getSecond());

List<List<SyncOrgUnit>> syncUnits = splitByAccount(syncUnitsSet);

for (List<SyncOrgUnit> syncUnitList : syncUnits) {

if (!syncUnitList.isEmpty()) {

syncProvider.connect(syncUnitList.get(0).getTargerAccount().getLoginName(), syncUnitList.get(0)

.getTargerAccount().getLoginPassword());

syncProvider.runUnitSync(syncUnitList);

}

}

for (SyncItem item : itemsFromService) {

if (item.getSyncType() == SyncType.THIS\_TIME) {

syncService.delete(item);

}

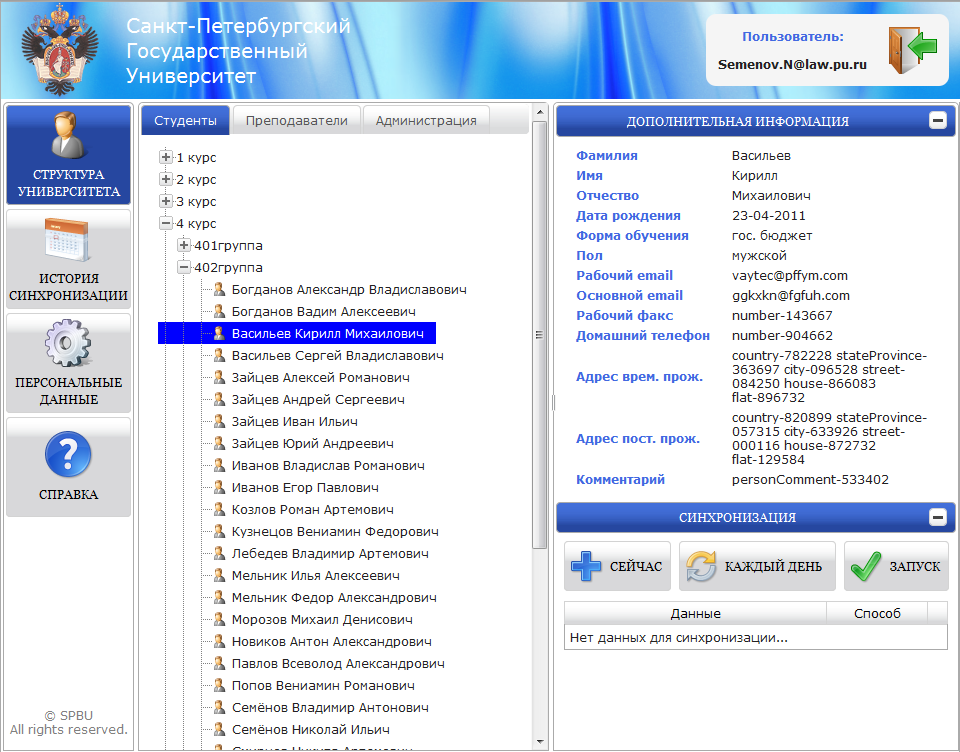
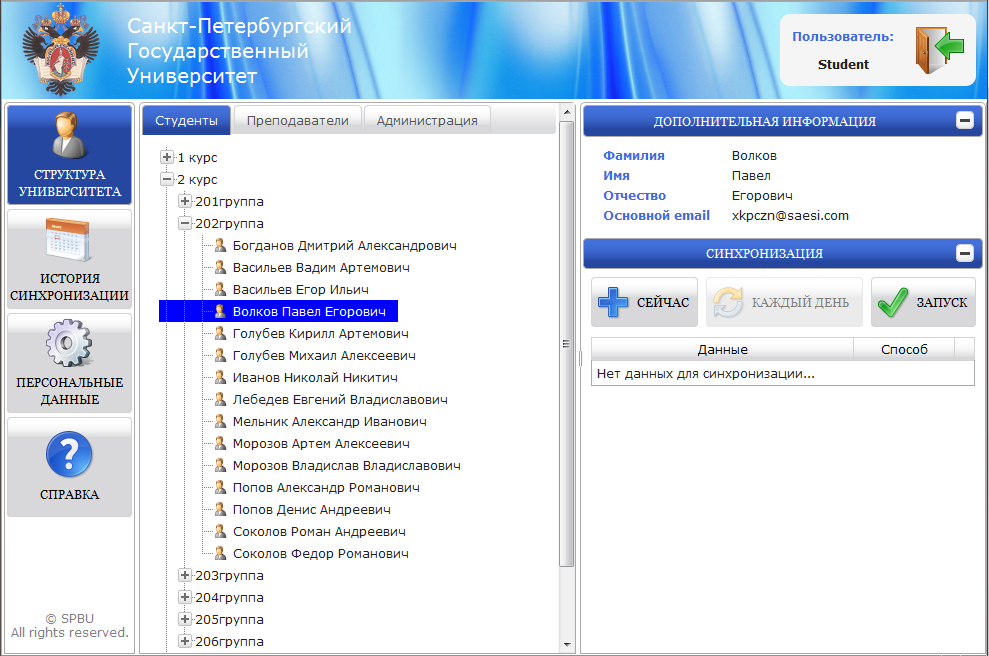
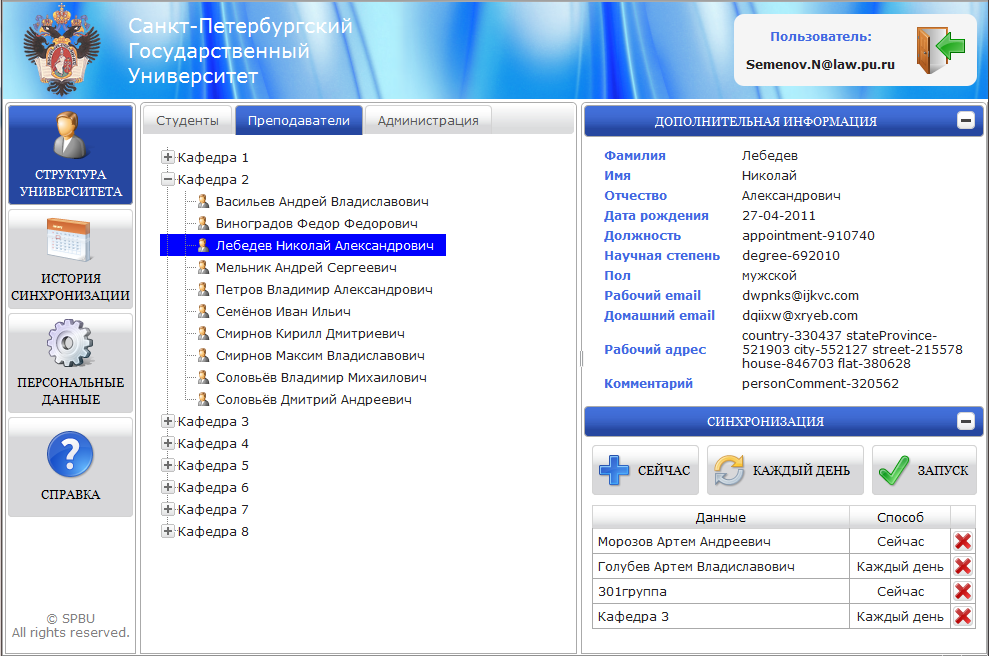
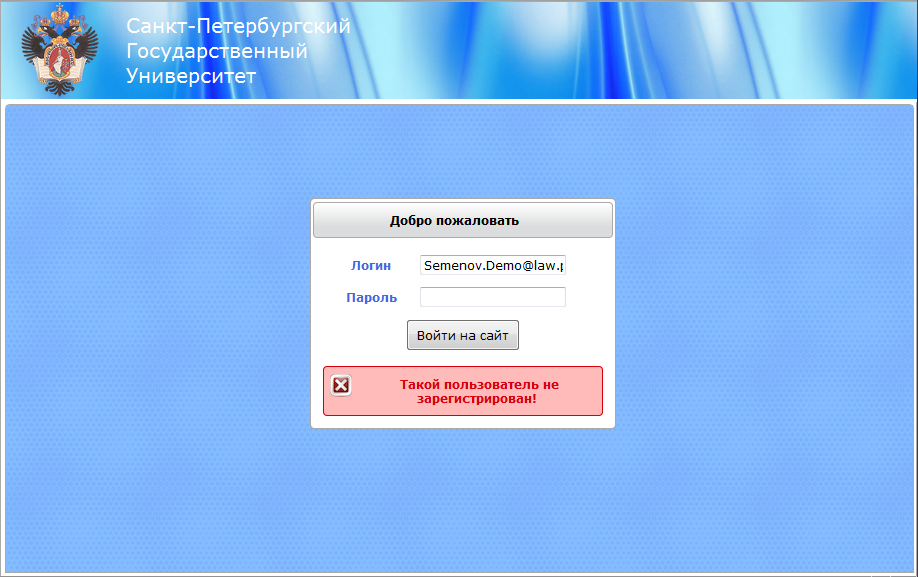
}

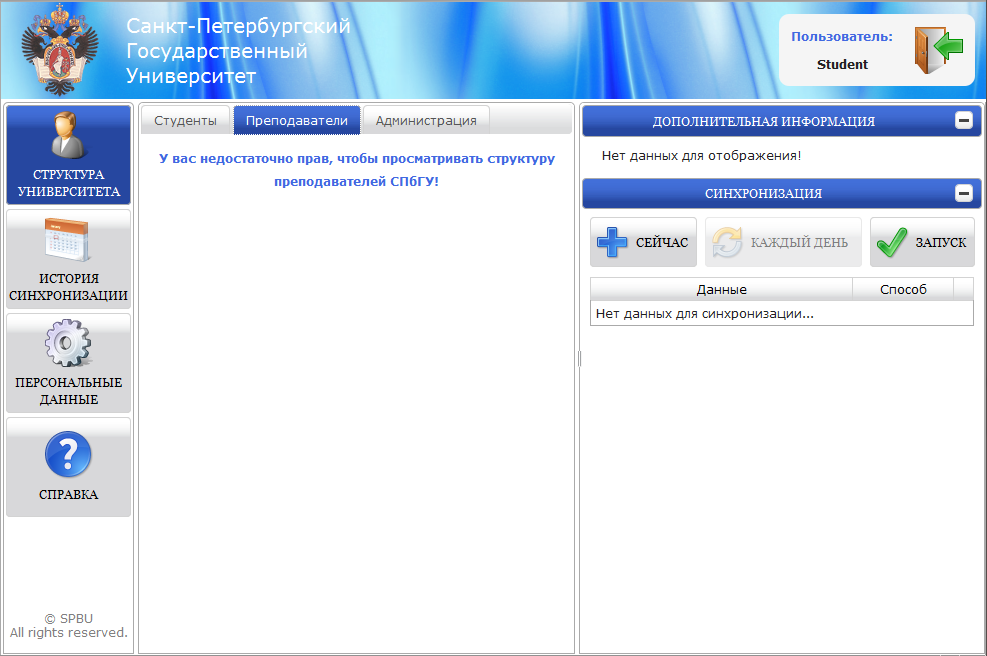
}

}

C:\Program Files\Microsoft Office\MEDIA\OFFICE12\Lines\BD15155_.gif

## C:\Users\Nick\Desktop\Диплом\Материалы\Скриншоты\Страница логина.pngПриложение 2. Скриншоты программы





# Приложение 3. Рекомендуемые энциклопедические статьи

1. http://en.wikipedia.org/wiki/Синхронизация\_(информатика)
2. http://en.wikipedia.org/wiki/Microsoft\_Exchange\_Server
3. http://ru.wikipedia.org/wiki/Java\_EE
4. http://ru.wikipedia.org/wiki/OSGi
5. http://ru.wikipedia.org/wiki/Apache\_Felix
6. http://en.wikipedia.org/wiki/Equinox
7. http://ru.wikipedia.org/wiki/Apache\_Ant
8. http://ru.wikipedia.org/wiki/Maven
9. http://ru.wikipedia.org/wiki/PostgreSQL
10. http://en.wikipedia.org/wiki/GlassFish
11. http://en.wikipedia.org/wiki/Enterprise\_JavaBean
12. http://en.wikipedia.org/wiki/Java\_Persistence\_API
13. http://en.wikipedia.org/wiki/Java\_Transaction\_API
14. http://en.wikipedia.org/wiki/Data\_access\_object
15. http://en.wikipedia.org/wiki/Object-relational\_mapping
16. http://en.wikipedia.org/wiki/JDBC
17. http://ru.wikipedia.org/wiki/EclipseLink
18. http://ru.wikipedia.org/wiki/JavaServer\_Faces
19. http://en.wikipedia.org/wiki/Apache\_Shiro
20. http://en.wikipedia.org/wiki/SSL
21. http://en.wikipedia.org/wiki/Load\_balancing\_(computing)
22. http://en.wikipedia.org/wiki/Apache\_Click