Содержание

Введение 3

Глава 1. Семантические сети 4

1.1. Введение в семантические сети 4

1.2. OWL 6

1.3. SPARQL 6

Глава 2. Сравнение RDF с другими моделями хранилищ . 8

2.1. Реляционные хранилища 8

2.2. Key-Value хранилища(NoSQL) 9

2.3. RDF-хранилища 13

Глава 3. Разработка RDF-хранилища 15

3.1. Построение онтологии 15

3.2. Выбор архитектуры и инструментов 16

3.3. RDFrb 17

3.4. SPARQL-точка 18

3.5. Spira 20

Глава 4. Заполнение RDF хранилища 22

4.1. Парсинг интернет-ресурса Web-аптека 22

4.2. Сохранение информации в RDF-формате 23

Глава 5. Построение web-интерфейса 25

5.1. Главная страница 25

5.2. Страница препарата 26

5.3. Калькулятор лекарств 27

5.4. SPARQL форма 30

Заключение 30

Листинг 31

Парсинг Webapteka 32

Настройка RDF-репозитория 33

Заполнение RDF-хранилища 33

Контроллеры 36

Модели 37

Представления 41

Введение

В последнее время наблюдается взрывной рост количества новых источников информации о генах, протеинах, химических исследованиях, лекарствах и болезнях. Растет при этом и разрозненность информации. Этот фактор приводит к тому, что результаты новых исследований просто теряются в море устаревших сведений. Требуется совершить переход к новой модели хранения и обработки информации - Linked Data. Использование этой методологии существенно повысит связанность медицинских знаний, позволит открывать новые закономерности и получать новейшие данные об исследованиях.

Основная цель моей работы - это создание семантического хранилища медицинских знаний. Для достижения поставленной цели решались следующие задачи:

1. Скачивание и парсинг информации с ресурса Webapteka
2. Разработка онтологии лекарственных препаратов
3. Конвертация html данных в rdf представление
4. Разработка SPARQL-запросов для извлечения информации и выявления дополнительных связей в RDF-хранилище.
5. Разработка пользовательского интерфейса
6. Кеширование элементов приложения для повышения производительности

Глава 1

Семантические сети

1.1. Введение в семантические сети

Семантическая сеть (англ. Semantic Web) — это набор технологий, позволяющих представлять информацию в виде пригодном для машинной обработки: RDF, OWL, SPARQL. RDF используется для представления информации, SPARQL - для доступа к ней, OWL - добавляет метаинформацию, связи между концептами.

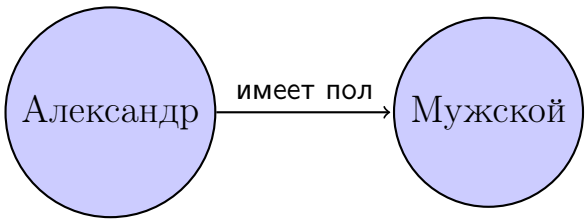
В RDF вся информация представляется в виде триплетов: субъект, предикат, объект. Триплеты по форме похожи на простое предложение. Например:

Субъект: Александр

Предикат: Имеет пол

Объект: Мужской

Триплет может быть выражен в виде графа

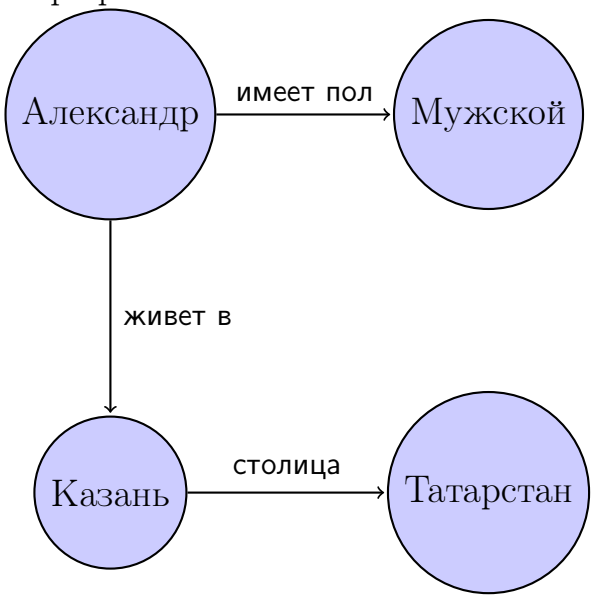


Субъекты и объекты могут быть представлены URI, либо литералом. URI - это уникальный идентификатор, который обозначает сущность: например URI для собаки может быть таким *'http://example.ru/animals/dog'*. Литерал - это просто строка, например 'Jack Nickolson', с возможными добавлениями, указывающими язык, тип данных (поддерживаемый XML, такие как integer и datetime). В идеале между сущностями и URI составлено

взаимно однозначное соответствие: каждый URI принадлежит только одной сущности и каждая сущность имеет только один URI. Для обозначение предикатов всегда используются URI.

Использование URI в RDF облегчает нахождение документов, связанных с сущностью. Например, если кто-то(или чья-то программа) ищет информацию о собаках, то ему надо искать все триплеты содержащие URI *http://www.example.ru/animals/dog*.

RDF-документ представляет собой набор триплетов. Его можно выразить в виде графа, если представить URI как вершины, а предикаты как ребра графа.



Таким образом можно построить граф неограниченного размера. RDF как язык для хранилища имеет ряд преимуществ:

1. Простое агрегирование данных. Необходимо только добавить триплеты с указанием связи между сущностями.

2. Использование URI дает возможность объединять информацию о сущности с нескольких источников данных.

3. Поскольку RDF не имеет жестких, заведомо заданных требований к структуре данных, к наличию или отсутствию свойства, повышается

плотность хранения информации.

4. RDF предлагает единый язык для представления практически любого знания.

1.2. OWL

Технологии Semantic Web дают возможность выводить новые факты из базовых фактов, хранящихся в RDF. OWL добавляет к RDF информацию о классах, типах, логических зависимостях, доменов у свойств, пространстве возможных значений.

1.3. SPARQL

SPARQL - язык запросов к RDF-хранилищу. SPARQL, как и большинство языков такого типа, содержит переменные в тексте запроса, в которые подставляются извлеченные данные. Запрос вида

```
SELECT ?x WHERE {
?x <http://www.example.com/has-gender> <http://www.example.com/male> .
}
```

найдет все триплеты с указанным предикатом и объектом(ИМЕЕТ ПОЛ, МУЖСКОЙ) и вернет список субъектов. Информация возвращается в XML-формате.

Запрос может быть построен из нескольких триплетов. В следующем примере кода две конструкции, которые должны вернуть всех людей мужского пола.

```
SELECT ?x WHERE {
?x <http://www.example.com/has-gender> <http://www.example.com/male> .
?x <http://www.example.com/has-species> <http://www.example.com/human> .
}
```

Это тип запросов основной в SPARQL. Хотя SPARQL беднее по функциональности чем SQL, он поддерживает схожий функционал для уточнения запроса: сортировка результатов, получение подмножества результатов, удаление дубликатов и т.д. Следующий запрос вернет всех мужчин, которые имеют больше, чем 20 книг и, если имеется информация о предпочтениях в еде, она вернется тоже.

```
PREFIX ex: <http://www.example.com/>
SELECT ?x ?foods WHERE {
?x ex:has-gender ex:male .
?x ex:has-species ex:human .
?x ex:has-book-count ?bookcount .
FILTER (?bookcount < 20)
}
OPTIONAL {
?x ex:likes-food ?foods .
}
}
```

Преимущества, которые могут быть получены за счет использования этого языка запросов понятны: человек или компьютер могут соединиться с любым открытым репозиторием, сделать очень специфичный запрос и получить машино-обрабатываемые данные.

Глава 2

Сравнение RDF с другими моделями хранилищ

В любом хранилище данных, доступ к информации осуществляется в соответствии с некоторой моделью, логической концепцией. В этой главе описываются модели хранения данных, использующиеся в настоящее время. Исследуются сходства RDF-модели с остальными и определяется, в какой степени подходы для традиционных баз данных применимы к RDF-хранилищам.

2.1. Реляционные хранилища

Эта модель была предложена в 1970г. Э.Коддом. В этом подходе теория множеств и логика предикатов используются для определения логической структуры хранилища данных и операций, которые могут быть к нему применены. В часности, разделяются логическая структура и физическая. СУБД может выбрать любой способ физического хранения данных, но то, как информация отображается пользователю, остается неизменным.

Реляционная модель описывает данные в терминах реляций, состоящих из неограниченного числа кортежей и аттрибутов. Реляции в целом схожи с таблицами, состоящих из строк и столбцов. Каждый кортеж является уникальным (ведь не имеет смысла один и тот же факт дважды). Запросы к СУБД пишутся на декларативном языке, позволяющим пользователям указать, какие данные они хотят получить, не заставляя их указывать, каким способом это сделать. Как правило, это ответственность СУБД сделать запрос как можно более быстрым. Компонент, который выполняет

оптимизацию, называется оптимизатором запросов.

Реляционная модель предназначена для поддержки запросов с перекрестными ссылками между блоками данных: *'Получить всех механиков, которые работали над машиной, содержащей деталь X.'*

2.2. Key-Value хранилища(NoSQL)

Имеют ряд существенных отличий от реляционных хранилищ:

1. В этой модели записи идентифицируются по ключу, при этом каждая запись имеет динамический набор атрибутов, связанных с ней.

2. Вместо реляций вводится понятие домена. Для доменов можно провести аналогию с таблицами, однако в отличие от таблиц для доменов не определяется структура данных. Домен – это коробка, в которую можно складывать все что угодно. Записи внутри одного домена могут иметь разную структуру.

3. В некоторых реализациях атрибуты могут быть только строковыми. В других реализациях атрибуты имеют простые типы данных, которые отражают типы, использующиеся в программировании: целые числа, массива строк и списки.

4. Между доменами, также как и внутри одного домена, отношения явно не определены.

Хранилища типа ключ-значение ориентированы на работу с записями. Это значит, что вся информация, относящаяся к данной записи, хранится вместе с ней. Домен может содержать бессчетное количество различных записей. Например, домен может содержать информацию о клиентах и о заказах. Это означает, что данные, как правило, дублируются между разными

доменами. Это приемлемый подход, поскольку дисковое пространство дешевое. Главное, что он позволяет все связанные данные хранить в одном месте, что улучшает масштабируемость, поскольку исчезает необходимость соединять данные из различных таблиц. При использовании реляционной БД, потребовалось бы использовать соединения, чтобы сгруппировать в одном месте нужную информацию.

Хотя для хранения пар ключ-значение потребность в отношения резко падает, отношения все же нужны. Такие отношения обычно существуют между основными сущностями. Например, система заказов имела бы записи, которые содержат данные о покупателях, товарах и заказах. При этом неважно, находятся ли эти данные в одном домене или в нескольких. Вместо этого, запись о заказе должна содержать ключи, которые указывают на соответствующие записи о покупателе и товаре. Поскольку в записях можно хранить любую информацию, а отношения не определены в самой модели данных, система управления базой данных не сможет проконтролировать целостность отношений. Это значит, что можно удалять покупателей и товары, которые они заказывали. Обеспечение целостности данных целиком ложится на приложение.

Сравнение key-value хранилищ и реляционных: Доступ к данным

Реляционная БД	Хранилище типа ключ-значение
Данные создаются, обновляются, удаляются и запрашиваются с использованием языка структурированных запросов (SQL).	Данные создаются, обновляются, удаляются и запрашиваются с использованием вызова API методов.
SQL-запросы могут извлекать данные как из одиночной таблица, так и из нескольких таблиц, используя при этом соединения (join'ы).	Некоторые реализации предоставляют SQL-подобный синтаксис для задания условий фильтрации.
SQL-запросы могут включать агрегации и сложные фильтры.	Зачастую можно использовать только базовые операторы сравнений (=, !=, <, >, <= и =>).
Реляционная БД обычно содержит встроенную логику, такую как триггеры, хранимые процедуры и функции.	Вся бизнес-логика и логика для поддержки целостности данных содержится в коде приложений.

Взаимодействие с приложениями

Реляционная БД	Хранилище типа ключ-значение
Чаще всего используются собственные API, или обобщенные, такие как OLE DB или ODBC.	Чаще всего используются SOAP и/или REST API, с помощью которых осуществляется доступ к данным.
Данные хранятся в формате, который отображает их натуральную структуру, поэтому необходим маппинг структур приложения и реляционных структур базы.	Данные могут более эффективно отображаться в структуры приложения, нужен только код для записи данных в объекты.

Преимущество хранилищ типа ключ-значение состоит в том, что они проще, а значит обладают большей масштабируемостью, чем реляционные БД. Такие хранилища легко и динамически расширяются. Но, в свою очередь, ограничения в реляционных БД гарантируют целостность данных на самом низком уровне. Данные, которые не удовлетворяют ограничениям, физически не могут попасть в базу. В хранилищах типа ключ-значение таких ограничений нет, поэтому контроль целостности данных полностью лежит на приложениях. Однако в любом коде есть ошибки. Если ошибки в правильно спроектированной реляционной БД обычно не ведут к проблемам целостности данных, то ошибки в хранилищах типа ключ-значение обычно приводят к таким проблемам.

Другое преимущество реляционных БД заключается в том, что они вынуждают вас пройти через процесс разработки модели данных. Если вы хорошо спроектировали модель, то база данных будет содержать логическую структуру, которая полностью отражает структуру хранимых данных, однако расходится со структурой приложения. Таким образом, данные

становятся независимы от приложения. Это значит, что другое приложение сможет использовать те же самые данные и логика приложения может быть изменена без каких-либо изменений в модели базы. Чтобы проделать то же самое с хранилищем типа ключ-значение, попробуйте заменить процесс проектирования реляционной модели проектированием классов, при котором создаются общие классы, основанные на естественной структуре данных.

2.3. RDF-хранилища

RDF-системы баз данных относятся к типу NoSQL моделей, основанных на графах. Ключевые отличия от других реализаций:

1. Поддерживают спецификацию W3C - Linked Data.

2. Мощный стандартный язык запросов. NoSQL базы данных в основном не предоставляют высокоуровненого декларативного языка запросов, аналогичного SQL. Запросы в таких базах данных основываются на программной модели хранимой информации и языке реализации. SPARQL - очень большое преимущество RDF-хранилищ, обеспечивающее стандартизированный и совместимый язык с мощным функционалом.

3. Стандартизированный формат данных. Все RDF-хранилища поддерживают XML и N-Triples форматы данных.

4. Простое объединение данных, источников данных. Так как RDF представляет собой перечень триплетов, для объединения данных необходимо только конкатенировать два списка. SPARQL язык поддерживает обращение к нескольким источникам в одном запросе.

5. Содержат общедоступное описание схемы хранимой информации на декларативном языке OWL.

Суммируя выше сказанное, RDF является оптимальным решением для организации сильно-распределенных хранилищ информации.

Глава 3

Разработка RDF-хранилища

3.1. Построение онтологии

При построении RDF-хранилища, в первую очередь разрабатывается онтология - описание схемы данных хранилища на языке OWL. Уже существует множество полезных онтологий, части которых можно использовать в своей онтологии либо создавать на них ссылки.(проект http://swoogle.umbc.edu/ позволяет искать онтологии). Мощным средством для построения онтологий является Protege.

Основной класс в онтологии проекта - это Drug(лекарство). Большинство остальных классов образуют объектную часть предикатов.

Список свойств класса Drug:

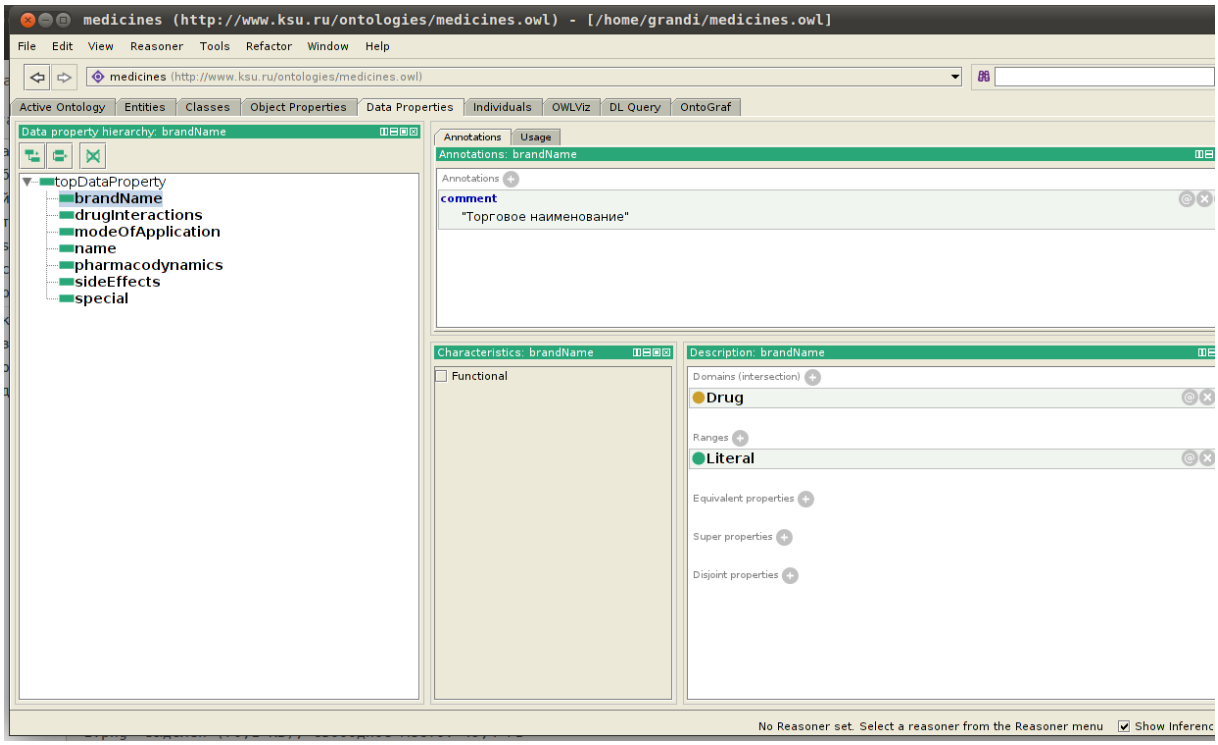
1. имеет международное название
2. имеет торговое наименование
3. принадлежит нескольким категориям
4. имеет множество форм применения
5. имеет фармакологию
6. имеет множество показаний
7. имеет множество противопоказаний
8. имеет множество побочных эффектов
9. имеет дозировку

10. взаимодействует с другими лекарствами и группами лекарств

11. содержит лекарственные ингридиенты

12. имеет специальные указания

В Protege онтология имеет следующий вид:



3.2. Выбор архитектуры и инструментов

Ruby - мощный и гибкий язык. Он предоставляет лаконичный синтаксис, возможность изменять классы и методы во время выполнения программы, на нем написано множество библиотек, существенно облегчающих построение Web-приложения. Руководствуясь этими соображениями, RDF-хранилище будет писаться на Ruby.

Существует несколько библиотек для работы с RDF-хранилищами, реализованных на Ruby. Основные отличия между ними – поддержка функций вывода, степень поддержки вывода для OWL, возможности использо-

вания в качестве точки доступа SPARQL, веб-доступ, возможность хранения четверок вместо триплетов, а так же поддержка хранилища языками программирования (наличие модулей).

Кроме того, важным критерием является способность проецировать схему субъект-предикат-объект на экземпляр-свойство-значение, таким образом используя объектно-ориентированное программирование. Существуют, однако, глубинные различия объектов из ООП языков и RDF-объектами. Особенность RDF-сущностей в том, что они имеют URI, и они могут экземплярами нескольких классов. Для преодоления этой проблемы используется общая модель данных для ООП и RDF. В этой модели сущности могут иметь методы и свойства, должны иметь URI и могут быть экземплярами нескольких классов. Таким образом сочетаются возможности ООП и RDF.

Для использования в проекте, выбрана библиотека RDFrb. Она предоставляет наиболее обширный диапазон возможностей, и, что важно с исследовательской точки зрения, предоставляет абстрактный класс для переопределения функций хранилища.

Web-сервер будет базироваться на Ruby on Rails - фреймворке, написанном на Ruby, предоставляющем возможности xml-сериализации, html-парсинга, кеширования, REST-архитектуры.

3.3. RDFrb

RDFrb - библиотека для работы с семантическими данными. Для создания семантического графа нужно только указать онтологию и перечислить триплеты.

```
include RDF
MyOnt = RDF::Vocabulary.new("http://ricardolopes.net/myont.owl#")
graph = RDF::Graph.new
```

```
items.each do |item|
  graph << [item["uri"], RDF.type, MyOnt.Item]
  graph << [item["uri"], MyOnt.hasName, item["name"]]
end
return graph
```

В этом примере предполагается, что у нас есть онтология с URI *'http://ricardolopes.net/myont.owl'*, которая содержит по меньшей мере класс Item и свойство hasName.

Теперь сохраним граф в N-Triples формате:

```
RDF::Writer.open("data/graph.nt") do |writer|
  graph.each_statement do |stmt|
    writer << stmt
  end
end
```

3.4. SPARQL-точка

RDFrb дает возможность писать запросы к RDF-хранилищу как на чистом Ruby, так и на языке SPARQL. Для организации внешнего доступа к SPARQL-точке был добавлен роут по адресу /sparql, который получает GET-параметром текста запроса и возвращает ответ RDF-хранилища в XML-формате. Также было написано несколько SPARQL-запросов для получения данных из RDF-хранилища внутри приложения.

3.4.1. Получение списка всех лекарств

Лекарство называются те сущности, которые имеют тип *Лекарство*. SPARQL запрос выглядит так:

```
SELECT DISTINCT ?drug
  WHERE {
    ?drug <#{RDF.type}> <#{@type}>
  }
```

```
}
```

type - переменная экземпляра, которая содержит его RDF-тип.

3.4.2. Нахождение списка похожих лекарств

Похожими считаются лекарства, у которых есть одинаковый компонент. SPARQL запрос выглядит так:

```
SELECT ?drug
WHERE {
  ?drug <http://osmanov.me/drugComponent> ?drugPart .
  <#{uri}> <http://osmanov.me/drugComponent> ?drugPart
  FILTER (?drug != <#{uri}>)
}
```

Оператор FILTER в запросе убирает из списка похожих препаратов само лекарство.

3.4.3. Получение списка названий лекарств

Запрашивается список всех литералов, являющихся объектной частью в триплетах вида: *Лекарство имеет имя ...* SPARQL запрос выглядит так:

```
SELECT DISTINCT ?drugname
WHERE {
  ?drug <#{FOAF.name}> ?drugname
}
```

3.4.4. Получение списка суммарных противопоказаний для группы лекарств

Запрос получается конкатенированием условий от каждого лекарства. Метод выглядит так:

```
names_query = drugs.map do |drugname|
  "{ ?drug <#{FOAF.name}> '#{drugname}' }"
end.join(" UNION ")
```

```
query = "
  SELECT ?effects
  WHERE {
    #{names_query} .
    ?drug <http://osmanov.me/has_contraindication> ?effects
  }
"
sparql_query(query).map(&:effects).join(",")
```

3.4.5. Получение списка суммарных побочных эффектов для группы лекарств

Запрос получается конкатенированием условий от каждого лекарства. Метод выглядит так:

```
names_query = drugs.map do |drugname|
  "{ ?drug <#{FOAF.name}> '#{drugname}' }"
end.join(" UNION ")
query = "
  SELECT ?toxicity
  WHERE {
    #{names_query} .
    ?drug <http://osmanov.me/has_toxicity> ?toxicity
  }
"
sparql_query(query).map(&:toxicity).join(",")
```

3.5. Spira

Spira - это RDF ORM(фреймфорк для представления информации, хранящейся в RDF-хранилище, в виде объектов). С его использованием создается Ruby-класс Drug. Экземпляр этого класса содержит значения всех триплетов для сущности, которую он отображает. Для этого в заголовке добавляем информацию о хранимых аттрибутах.

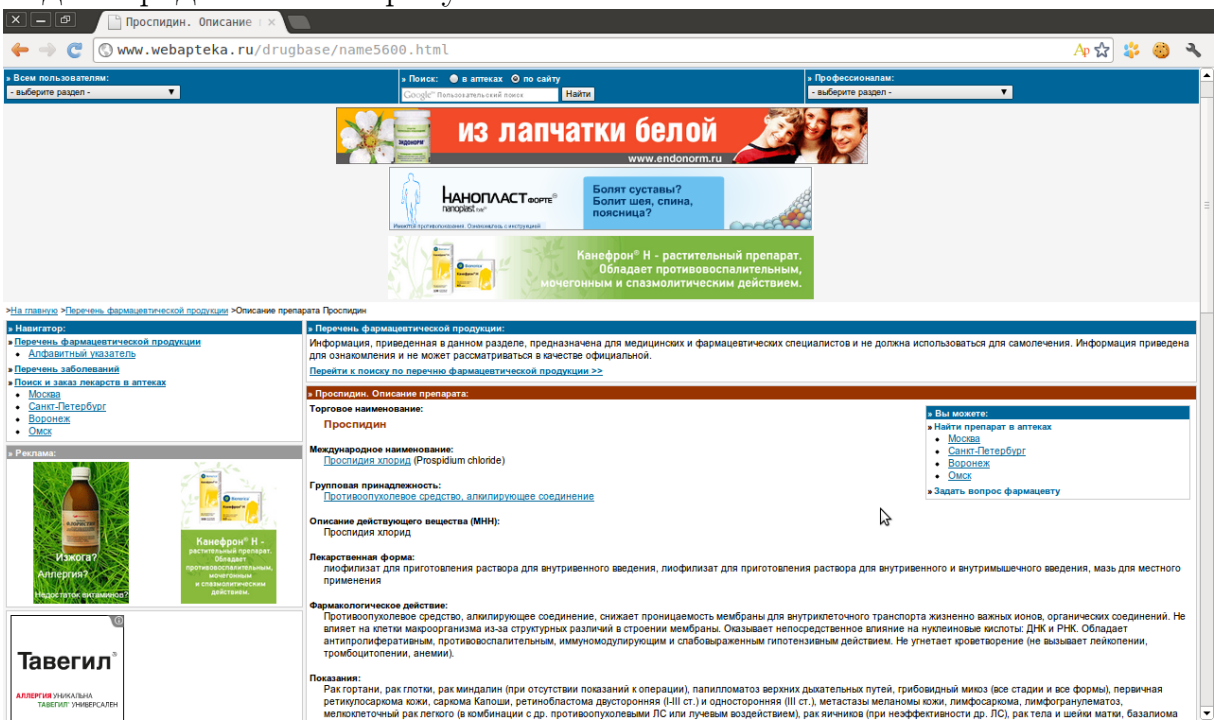
```
include Spira::Resource
base_uri "http://osmanov.me/drugs/"
type URI.new("http://osmanov.me/drugs")
property :intern_title,
  :predicate => URI.new("http://osmanov.me/intern_title"),
  :type => String
property :brandName,
  :predicate => FOAF.name,
  :type => String
has_many :drugCategories,
  :predicate => URI.new("http://osmanov.me/has_drug_category"),
  :type => :DrugCategory
has_many :dosageForms,
  :predicate => URI.new("http://osmanov.me/has_dosage_form"),
  :type => String
property :pharmacology,
  :predicate => URI.new("http://osmanov.me/has_pharmacology"),
  :type => String
has_many :indications,
  :predicate => URI.new("http://osmanov.me/has_indication"),
  :type => String
has_many :contraindications,
  :predicate => URI.new("http://osmanov.me/has_contraindication"),
  :type => String
has_many :sideEffects,
  :predicate => URI.new("http://osmanov.me/has_side_effects"),
  :type => String
property :dosage,
  :predicate => URI.new("http://osmanov.me/dosage"),
  :type => String
has_many :interactions,
  :predicate => URI.new("http://osmanov.me/has_interactions"),
  :type => :DrugInteraction
has_many :drugParts,
  :predicate => URI.new("http://osmanov.me/drugComponent"),
  :type => :DrugPart
property :special,
  :predicate => URI.new("http://osmanov.me/special"),
  :type => String
```

Глава 4

Заполнение RDF хранилища

4.1. Парсинг интернет-ресурса Web-аптека

Для наполнения RDF-хранилища использовался ресурс *http://webapteka.ru*. Страница лекарства имеет URL вида *http://webapteka.ru/drugbase/name1.html*, вид ее представлен на рисунке.



Для получения базы лекарств необходимо пройти в цикле по всем URL, подставляя значение итератора как номер лекарства и распознать информацию.

Страница представляет собой HTML с inline-стилями, семантические блоки не обозначены классами. Поэтому расположение требуемой информации идентифицировалось по стилям, оформляющего его блока. Для построения DOM-дерева документа и поиску по нему использовалась библиотека Nokogiri.

Теперь для того, чтобы получить DOM-элемент необходимо только

указать его CSS-путь или XPATH-путь. Так, например, выделяется блок с информацией о лекарстве:

```
doc.xpath("//div[@style='_width:100%; padding:5px; clear:left ']").first
```

Названия свойств лекарства выделены жирным шрифтом:

```
table.css("b").first
```

Сами свойства имееют сдвиг влево на 20px:

```
table.xpath(".//div[@style='margin-left:20px']")
```

Распознанная информация сохраняется в файл в формате YAML.

4.2. Сохранение информации в RDF-формате

Как итог парсинга ресурса Webapteka получается файл в формате YAML, содержащей таблицы сопоставления свойства и его названия для каждого лекарства. Для того, чтобы преобразовать эти данные в формат RDF, был написан скрипт, создающий записи в RDF хранилище и ставящий в соответствие записи из YAML-файла и новыми записями.

Свойства примитивного типа данных просто копируются в свойство у Spira-модели. Те же свойства, которые описывают отношение один-ко-многим разбиваются на части, обозначающие название семантической единицы.

Так записывается торговое название лекарства:

```
drug_in_base.brandName = drug["Торговое наименование"]
```

А так список групп, к которым оно принадлежит:

```
groupName = drug["Групповая принадлежность"]
if groupName
  if drugCategories[groupName]
    drugCategory = DrugCategory.for(drugCategories[groupName])
  else
    drugCategory = DrugCategory.for(drugCategoryID)
```

```
      drugCategories[drug["Групповая принадлежность"]] = drugCategoryID
      drugCategoryID += 1
    end
    drugCategory.name = groupName
    drugCategory.save!
    drug_in_base.drugCategories.merge([drugCategory])
end
```

Глава 5

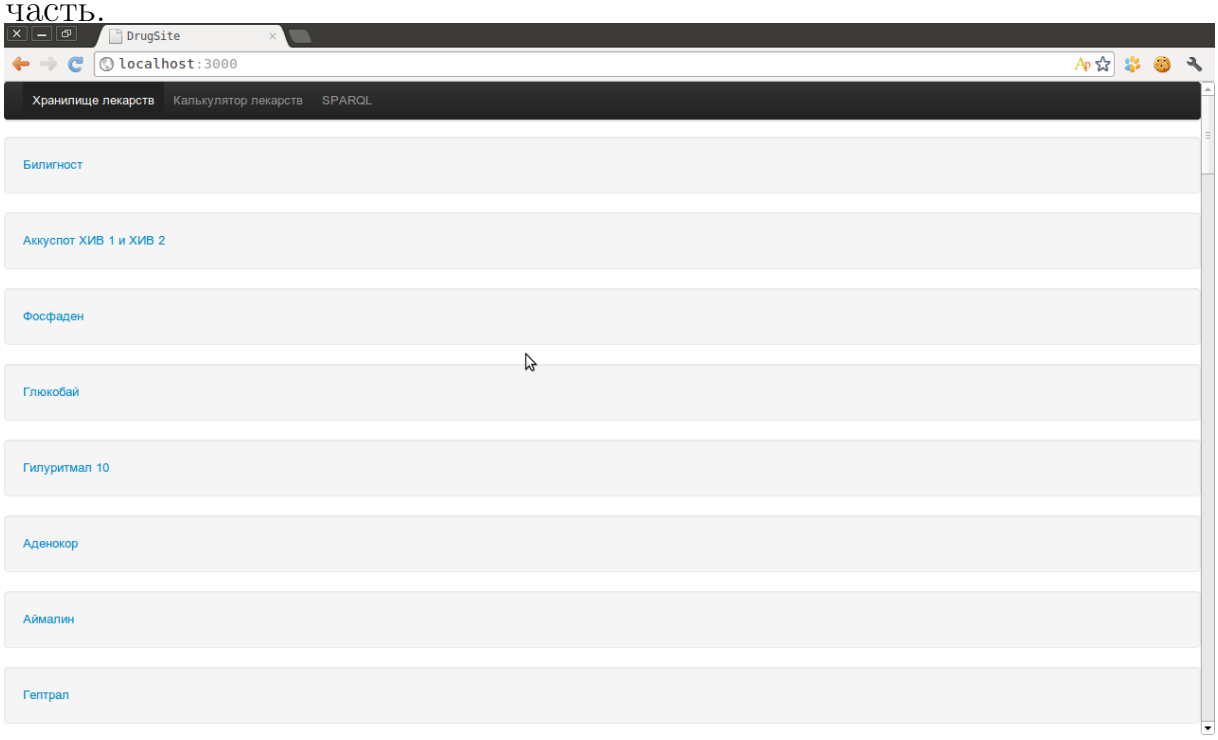
Построение web-интерфейса

Для прототипирования web-интерфейса в последнее время зачастую используются CSS-фреймворки. В моем проекте использовался Bootstrap. Его преимущества:

1. 12-столбцовая сетка
2. Jquery-плагины
3. Поддержка LESS

5.1. Главная страница

Главная страница содержит список всех лекарств и навигационную часть.



Согласно правилам оформления Bootstrap, навигация обрамляется блоками navbar, navbar-inner и container.

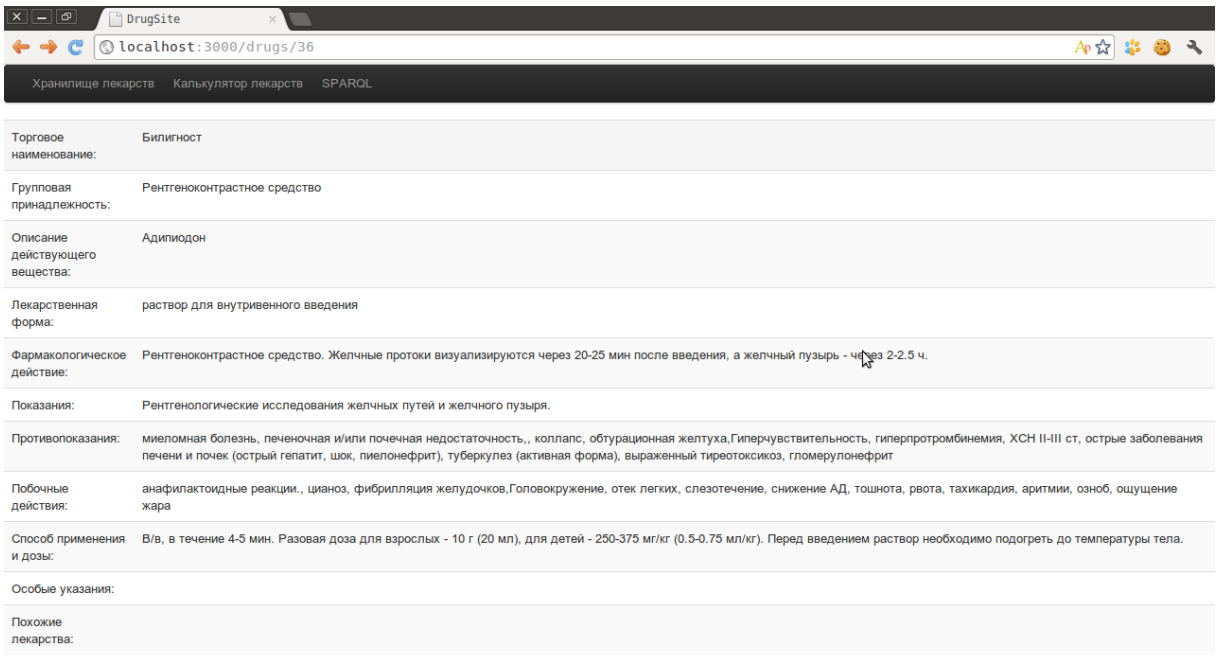
```
<div class="navbar">
  <div class="navbar-inner">
    <div class="container">
      <ul class="nav">
        <li class="active">
          <a href="/">Хранилище лекарств</a>
        </li>
        <li class="">
          <a href="/drugs/calc">Калькулятор лекарств</a>
        </li>
        <li class="">
          <a href="/sparql_form">SPARQL</a>
        </li>
      </ul>
    </div>
  </div>
</div>
```

Для выделения строки с названием лекарства, обрамляющему параграфу добавлен класс *well*.

```
<p class="well">
  <a href="/drugs/36">Билигност</a>
</p>
```

5.2. Страница препарата

На странице лекарства, помимо навигационной части, присутствует располосованная таблица со свойствами препарата.

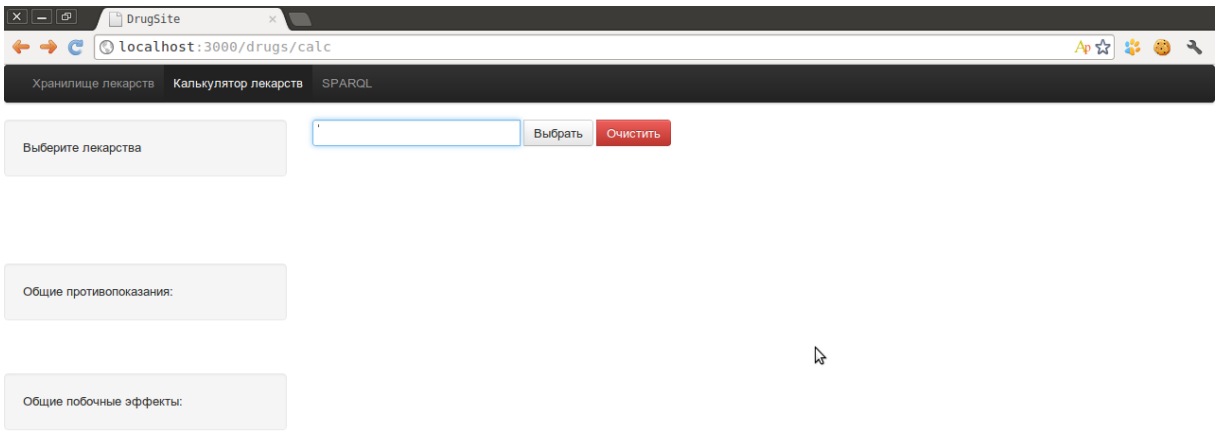


Этот эффект достигается путем добавления классов *table* и *table-striped* к таблице.

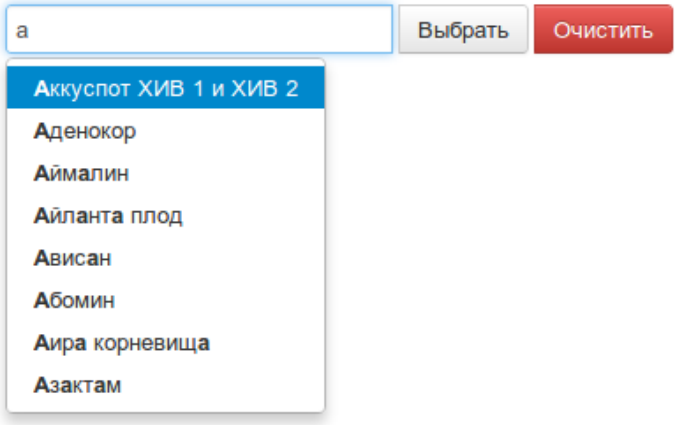
```
<table class="table table-striped">
```

5.3. Калькулятор лекарств

На этой странице присутствует форма с текстовым полем для ввода названия лекарства и два блока, в которую выводится полученная информация.



Вводимое название лекарства дополняется автоматически.



Для данного функционала использовался плагин *typeahead*. Чтобы его использовать, надо передать в функцию typehead массив возможных названий и идентификатор текстового поля.

```
$(".select_drugs #drugs").typeahead({
  source: <%= raw @drug_names.inspect %>
});
```

После выбора названия препарата, осуществляется ajax-запрос к серверу. Отправляется список выбранных лекарств, возвращается список общих противопоказаний и побочных эффектов. Для этого написаны две

javascript-функции:

```
selected_drugs = [];
var getContraIndications = function(drugs){
  $.get("<%= effects_drugs_path %>", {
    type: 'contraindications',
    drugs: selected_drugs
  }, function (response) {
    $(".contra-indications").text(response);
  });
};

var getToxicities = function(drugs){
  $.get("<%= effects_drugs_path %>", {
    type: 'toxicities',
    drugs: selected_drugs
  }, function (response) {
    $(".toxicities").text(response);
  });
};

$(".select_drugs .btn").click(function(e){
  e.preventDefault();
  var drug_name = $(".select_drugs #drugs").val();
  if (drug_name == "") {
    return false;
  };
  selected_drugs.push(drug_name);
  $(".selected_drugs").append("<li>" + drug_name + "</li>")
  $(".select_drugs #drugs").val("")
  getContraIndications(selected_drugs);
  getToxicities(selected_drugs);
});
```

После нажатия на кнопку *Очистить* удаляется содержимое списка выбранных лекарств и блоков с информацией о противопоказаниях и побочных эффектах:

```
$(".select_drugs .clear_selected_drugs").click(function(e){
  selected_drugs = [];
```

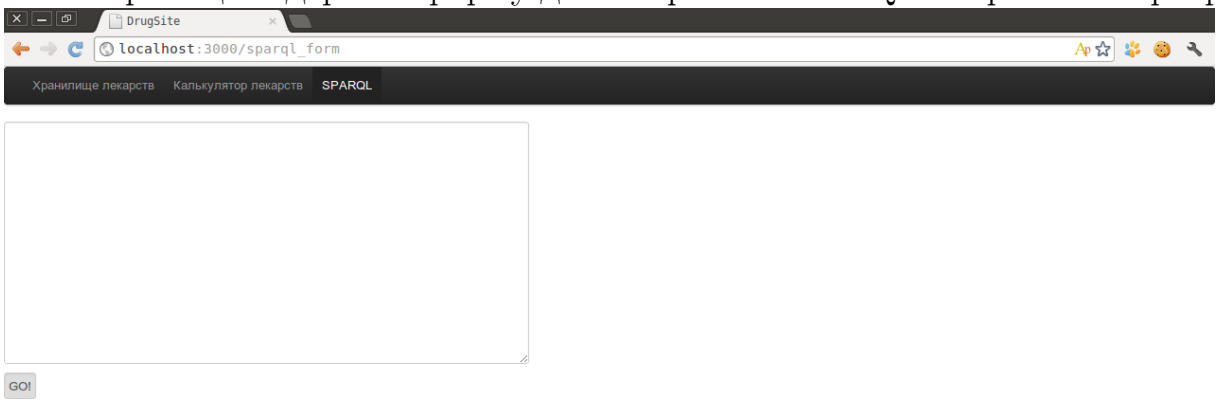
```
  $(".selected_drugs").html("");
  $(".contra-indications").html("");
  $(".toxicities").html("")
  e.preventDefault();
});
```

Кнопки оформлены стилями *btn* и *btn-danger*

```
<button class="btn" href="#">Выбрать</button>
<button class="btn btn-danger clear_selected_drugs" href="#">Очистить</button>
```

5.4. SPARQL форма

Страница содержит форму для отправки SPARQL-запросов к серверу.



Заключение

В данной работе представлены и проанализированы все этапы современной разработки RDF-хранилища:

1. Построение онтологии
2. Распознавание и подготовка данных для RDF-хранилища.
3. Заполнение RDF-хранилища.
4. Написание SPARQL-запросов, реализация SPARQL-точки.
5. Построение пользовательского интерфейса.

В реализованном RDF-хранилище представлено более 20000 лекарств, подготовлена система создания связей с другими RDF-хранилищами. Пользователи сайта могут получать информацию о лекарствах, недоступную на других ресурсах: суммарные противопоказания и побочные эффекты для группы лекарств, похожие препараты.

Данная система может использоваться как информационный ресурс для врачей и пациентов. Благодаря SPARQL-точке, хранилище может использоваться и как основа для семантических ризонеров.

Листинг

Парсинг Webapteka

```
require 'rubygems'
require 'nokogiri'
require 'open-uri'
require 'iconv'
require 'yaml'

drugs = []
i = 0
(i*5000 + 1).upto((i+1)*5000) do |i|
  s = open("http://www.webapteka.ru/drugbase/name#{i}.html")
  doc = Nokogiri::HTML(s, nil, "koi8-r")
  table = doc
    .xpath("//div[@style='_width:100%; padding:5px; clear:left']")
    .first
  next unless table
  table.css("span").first.remove
  values = []
  names = []
  drug = {}
  drug["link"] = "http://www.webapteka.ru/drugbase/name#{i}.html"
  table.xpath(".//div[@style='margin-left:20px']").each do |el|
    values << el.text
    el.remove
  end
  table.css("b").each do |a|
    names << a.text
  end

  names.each_with_index do |name, index|
    drug[name.gsub(/:$/, "")] = values[index]
  end
  drugs << drug
end
File.open("drugs#{i}.yml", "w") do |file|
```

```
  file.write drugs.to_yaml
end
```

Настройка RDF-репозитория

```
require 'do_sqlite3'

repository = RDF::DataObjects::Repository.new "sqlite3:base.db"
Spira.add_repository!(:default, repository)

class ActiveRecord::Base
  def self.sparql_query(query)
    repo = Spira.repositories[:default]
    @@repository ||= RDF::Repository.new do |graph|
      repo.each do |st|
          graph << [st.subject, st.predicate, st.object]
      end
    end
    SPARQL.execute(query, @@repository)
  end
end
```

Заполнение RDF-хранилища

```
drugs = YAML::load_file(Rails.root.join("db/drugs.yml"))

drugCategories = {}
drugParts = {}

drugID = 1
drugPartID = 1
drugCategoryID = 1
drugNames = {}
drugs[0,100].each do |drug|
  #имена
  uniq_name = drug["Международное наименование"]
  next unless uniq_name
  english_name = uniq_name.match(/\((.*)\)/)[1]
  drug_in_base = Drug.for(drugID); drugID += 1
```

```
drug_in_base.intern_title = english_name
drug_in_base.brandName = drug["Торговое наименование"]

drugNames[drug["Торговое наименование"]] = drugID

#группа
groupName = drug["Групповая принадлежность"]
if groupName
  if drugCategories[groupName]
    drugCategory = DrugCategory.for(drugCategories[groupName])
  else
    drugCategory = DrugCategory.for(drugCategoryID)
    drugCategories[drug["Групповая принадлежность"]] = drugCategoryID
    drugCategoryID += 1
  end
  drugCategory.name = groupName
  drugCategory.save!
  drug_in_base.drugCategories.merge([drugCategory])
end

#ингридиенты
drugPartName = drug["Описание действующеговещества "]
if drugPartName
  if drugParts[drugPartName]
    drugPart = DrugPart.for(drugParts[drugPartName])
  else
    drugPart = DrugPart.for(drugPartID)
    drugParts[drug["Описание действующеговещества "]] = drugPartID
    drugPartID += 1
  end
  drugPart.name = drugPartName
  drugPart.save!
  drug_in_base.drugParts.merge([drugPart])
end
if drug["Лекарственная форма"]
  drug_in_base.dosageForms.merge(drug["Лекарственная форма"].split(","))
end
drug_in_base.pharmacology = drug["Фармакологическое действие"]
if drug["Показания"]
```

```
    drug_in_base.indications.merge(drug["Показания"].split(","))
  end
  if drug["Противопоказания"]
    drug_in_base.contraindications
      .merge(drug["Противопоказания"].split(/[\,\.]/))
  end
  if drug["Побочные действия"]
    drug_in_base.sideEffects.merge(drug["Побочные действия"].split(","))
  end
  drug_in_base.dosage = drug["Способ применениядозы  "]



  drug_in_base.save!
end

drugID = 1
drugInteractionId = 1
drugs[0,100].each do |drug|
  vz = drug["Взаимодействие:"]
  next unless vz
  vz = vz.split(".")
  drugNames.each do |name, id|
    vz.each do |text|
      if text[name]
        di = DrugInteraction.for(drugInteractionId)
        di.drug = Drug.for(id)
        di.description = text
        di.save!
        drug.interactions.merge(di)
        drug.save!
        drugInteractionId += 1
      end
    end
  end
  drugID += 1
end
```

Контроллеры

DrugsController

```
class DrugsController < ApplicationController

  def index
    @drugs = Drug.all
  end

  def show
    @drug = Drug.for(params[:id])
    respond_to do |format|
      format.html
      format.rdf {render :inline => @drug.to_rdf}
    end
  end

  def calc
    @drug_names = Drug.drug_names
  end

  def effects
    drugs = params["drugs"]
    if params["type"] == "contraindications"
      result = Drug.contraindications_of(drugs)
    end
    if params["type"] == "toxicities"
      result = Drug.toxicity_of(drugs)
    end
    render :text => result
  end
end
```

SparqlController

```
class SparqlController < ApplicationController

  def sparql
```

```
    result = Drug.sparql_query(params[:query])
    render :xml => result.to_xml
  end

  def show

  end

end
```

Модели

Drug

```
class Drug < ActiveRecord::Base
  include Spira::Resource
  base_uri "http://osmanov.me/drugs/"
  type URI.new("http://osmanov.me/drugs")
  property :intern_title,
    :predicate => URI.new("http://osmanov.me/intern_title"),
    :type => String
  property :brandName,
    :predicate => FOAF.name,
    :type => String
  has_many :drugCategories,
    :predicate => URI.new("http://osmanov.me/has_drug_category"),
    :type => :DrugCategory
  has_many :dosageForms,
    :predicate => URI.new("http://osmanov.me/has_dosage_form"),
    :type => String
  property :pharmacology,
    :predicate => URI.new("http://osmanov.me/has_pharmacology"),
    :type => String
  has_many :indications,
    :predicate => URI.new("http://osmanov.me/has_indication"),
    :type => String
  has_many :contraindications,
    :predicate => URI.new("http://osmanov.me/has_contraindication"),
    :type => String
```

```
has_many :sideEffects,
  :predicate => URI.new("http://osmanov.me/has_side_effects"),
  :type => String
property :dosage,
  :predicate => URI.new("http://osmanov.me/dosage"),
  :type => String
has_many :interactions,
  :predicate => URI.new("http://osmanov.me/has_interactions"),
  :type => :DrugInteraction
has_many :drugParts,
  :predicate => URI.new("http://osmanov.me/drugComponent"),
  :type => :DrugPart
property :special,
  :predicate => URI.new("http://osmanov.me/special"),
  :type => String
def similar
  query = "
    SELECT ?drug
    WHERE {
      ?drug <http://osmanov.me/drugComponent> ?drugPart .
      <#{uri}> <http://osmanov.me/drugComponent> ?drugPart
      FILTER (?drug != <#{uri}>)
    }
  "

  self.class.sparql_query(query).map{|d| d[:drug].as(Drug)}
end

def self.all
  query = "
    SELECT DISTINCT ?drug
    WHERE {
      ?drug <#{RDF.type}> <#{@type}>
    }
  "

  self.sparql_query(query).map{|d| d[:drug].as(Drug)}
end
```

```
def id
  uri.to_s.split("/")[-1]
end

def self.drug_names
  query = "
    SELECT DISTINCT ?drugname
    WHERE {
      ?drug <#{FOAF.name}> ?drugname
    }
  "

  self.sparql_query(query).map{|d| d[:drugname].to_s}
end

def self.contraindications_of(drugs)
  names_query = drugs.map do |drugname|
    "{ ?drug <#{FOAF.name}> '#{drugname}' }"
  end.join(" UNION ")
  query = "
      SELECT ?effects
      WHERE {
        #{names_query} .
        ?drug <http://osmanov.me/has_contraindication> ?effects
      }
    "
  sparql_query(query).map(&:effects).join(",")
end

def self.toxicity_of(drugs)
  names_query = drugs.map do |drugname|
    "{ ?drug <#{FOAF.name}> '#{drugname}' }"
  end.join(" UNION ")
  query = "
      SELECT ?toxicity
      WHERE {
        #{names_query} .
        ?drug <http://osmanov.me/has_toxicity> ?toxicity
      }
```

```
      "
    sparql_query(query).map(&:toxicity).join(",")
  end
end
```

DrugCategory

```
class DrugCategory < ActiveRecord::Base
  include Spira::Resource
  base_uri "http://osmanov.me/drug_categories/"
  property :name, :predicate => DC.title
  type URI.new("http://osmanov.me/drug_parts")
end
```

DrugInteraction

```
class DrugInteraction < ActiveRecord::Base
  include Spira::Resource
  base_uri "http://osmanov.me/drug_interactions/"
  type URI.new("http://osmanov.me/drug_interactions")

  property :drug,
    :predicate => URI.new("http://osmanov.me/interaction_with"),
    :type => :Drug
  property :description,
    :predicate => URI.new("http://osmanov.me/description"),
    :type => String

end
```

DrugPart

```
require 'sxp'
require 'sparql/grammar'

class DrugPart < ActiveRecord::Base
  include Spira::Resource
  base_uri "http://osmanov.me/drug_parts/"
  property :name, :predicate => DC.title
  type URI.new("http://osmanov.me/drug_parts")
```

```
  def self.all
    sparql_query "SELECT * WHERE { ?s ?p ?o }"
  end
end
```

Представления

Основной слой

```
<!DOCTYPE html>
<html>
<head>
  <title>DrugSite</title>
  <%= stylesheet_link_tag    "application", :media => "all" %>
  <%= javascript_include_tag "application" %>
  <%= csrf_meta_tags %>
</head>
<body>
<div class="navbar">
  <div class="navbar-inner">
    <div class="container">
      <ul class="nav">
        <li class="<%= current_page?(root_path) ? 'active' : '' %>">
          <%= link_to "Хранилище лекарств", root_path %>
        </li>
        <li class="<%= current_page?(calc_drugs_path) ? 'active' : '' %>">
          <%= link_to "Калькулятор лекарств", calc_drugs_path %>
        </li>
        <li class="<%= current_page?(sparql_form_path) ? 'active' : '' %>">
          <%= link_to "SPARQL", sparql_form_path %>
        </li>
      </ul>
    </div>
  </div>
</div>
<%= yield %>

</body>
```

```
</html>
```

Список лекарств

```
<% @drugs.each do |drug| %>
  <p class="well">
    <%= link_to drug.brandName, drug_path(drug) %>
  </p>
<% end -%>
```

Страница лекарства

```
<table class="table table-striped">
  <tr>
    <td>Торговое наименование: </td>
    <td><%= @drug.brandName %></td>
  </tr>
  <tr>
    <td>Групповая принадлежность: </td>
    <td><%= @drug.drugCategories.map(&:name).join(",") %> </td>
  </tr>
  <tr>
    <td>Описание действующеговещества : </td>
    <td><%= @drug.drugParts.map(&:name).join(",") %> </td>
  </tr>
  <tr>
    <td>Лекарственная форма: </td>
    <td><%= @drug.dosageForms.to_a.join(",") %> </td>
  </tr>
  <tr>
    <td>Фармакологическое действие: </td>
    <td><%= @drug.pharmacology %> </td>
  </tr>
  <tr>
    <td>Показания: </td>
    <td><%= @drug.indications.to_a.join(",") %> </td>
  </tr>
  <tr>
    <td>Противопоказания: </td>
    <td><%= @drug.contraindications.to_a.join(",") %> </td>
```

```
  </tr>
  <tr>
    <td>Побочные действия: </td>
    <td><%= @drug.sideEffects.to_a.join(",") %> </td>
  </tr>
  <tr>
    <td>Способ применениядозы  : </td>
    <td><%= @drug.dosage %> </td>
  </tr>
  <tr>
    <td>Особые указания: </td>
    <td><%= @drug.special %> </td>
  </tr>
  <tr>
    <td>Похожие лекарства:</td>
    <td>
      <ul>
        <% @drug.similar.each do |sim_drug| %>
          <li>
            <%= link_to sim_drug.brandName, drug_path(sim_drug) %>
          </li>
        <% end -%>
      </ul>
    </td>
  </tr>
</table>
```

Калькулятор лекарств

```
<script type="text/javascript">
  $(function(){
    selected_drugs = [];
    var getContraIndications = function(drugs){
      $.get("<%= effects_drugs_path %>", {
        type: 'contraindications',
        drugs: selected_drugs
      }, function (response) {
        $(".contra-indications").text(response);
      });
```

```
    };

    var getToxicities = function(drugs){
      $.get("<%= effects_drugs_path %>", {
        type: 'toxicities',
        drugs: selected_drugs
      }, function (response) {
        $(".toxicities").text(response);
      });
    };
    $(".select_drugs #drugs").typeahead({
      source: <%= raw @drug_names.inspect %>
    });
    $(".select_drugs .btn").click(function(e){
      e.preventDefault();
      var drug_name = $(".select_drugs #drugs").val();
      if (drug_name == "") {
        return false;
      };
      selected_drugs.push(drug_name);
      $(".selected_drugs").append("<li>" + drug_name + "</li>")
      $(".select_drugs #drugs").val("")
      getContraIndications(selected_drugs);
      getToxicities(selected_drugs);
    });
    $(".select_drugs .clear_selected_drugs").click(function(e){
      selected_drugs = [];
      $(".selected_drugs").html("");
      $(".contra-indications").html("");
      $(".toxicities").html("")
      e.preventDefault();
    });
  });
</script>
<div class="row-fluid">
  <div class="span3">
    <div class="well">Выберителекарства

    </div>
```

```
  </div>
  <div class="span6">
    <%= form_tag effects_drugs_path,
      :class => "select_drugs form-horizontal" do %>
      <fieldset>
        <div class="input-append">
          <%= text_field_tag :drugs %>
          <button class="btn" href="#">Выбрать</button>
          <button class="btn btn-danger clear_selected_drugs" href="#">Очистить


          </button>
        </div>
      </fieldset>
    <% end -%>
    <div class="row-fluid">
      <div class="span6">
        <ul class="selected_drugs">
        </ul>
      </div>
      <div class="span6">
      </div>
    </div>
  </div>
</div>
<br>
<br>
<br>
<br>
<div class="row-fluid">
  <div class="span3">
    <div class="well">Общиепротивопоказания
      :
    </div>
  </div>
  <div class="span6">
    <div class="contra-indications">
    </div>
  </div>
</div>
```

```
<br>
<br>
<div class="row-fluid">
  <div class="span3">
    <div class="well">Общиепобочныеэффекты
        :
    </div>
  </div>
  <div class="span6">
    <div class="toxicities">
    </div>
  </div>
</div>
```