Содержание

Введение

Достижения в области биологических наук наряду с нарастанием объемов доступной для использования информации повышают необходимость в интеграции разрозненных источников данных. К ним относятся и интернет, и ведомственные, и корпоративные системы:

1. Медстатистика
2. Результаты клинических испытаний
3. Электронная история болезни
4. Фармацевтика
5. Разработка лекарственных средств

В настоящее время в России разработано и используется большое количество разрозненных медицинских информационных систем, разнообразных баз данных с описанием лекарственных препаратов, результатами научных исследований; написано множество научных трудов и статей в области здравоохранения (медицина, фармацевтика, медицинское страхование и др.), хранящихся в специализированных электронных библиотеках в различных форматах. Однако эффективные механизмы извлечения из таких источников знаний, хранения и предоставления к ним широкого доступа отсутствуют.

Требуется фундаментальный сдвиг от единичных попыток интеграции к единой функциональной области. Для решения этой проблемы разработан стандарт публикации данных Linked Data. Одним из важнейших достоинств этой технологии является ее открытость - возможность объединения в общую семантическую сеть распределенных семантических хранилищ,

созданных различными организациями (органы управления здравоохранением, ВУЗы, НИИ, МО) и профессиональными сообществами (ассоциации кардиологов, анестезиологов, медицинских IT-специалистов и др.) на основе единых открытых стандартов. Как показывает международный опыт, это позволяет системе саморазвиваться, постоянно пополняя количество доступных знаний и повышая их качество.

Межресурсные ссылки дают исследователям возможность перемещаться между источниками данных и открывать связи, которые не были замечены ранее. Существуют универсальные инструменты, такие как семантические веб-браузеры и поисковые движки, которые могут использоваться для задач представления и поиска данных.

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

Семантическая сеть

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.

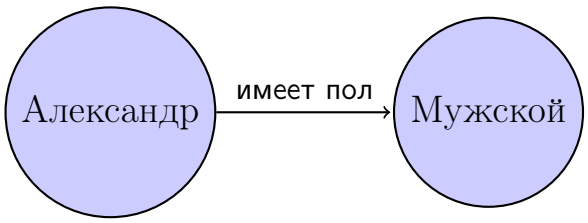
В RDF вся информация представляется в виде триплетов: субъект, предикат, объект. Триплеты по форме похожи на простое предложение. Например:

Субъект: Александр

Предикат: Имеет пол

Объект: Мужской

Триплет может быть выражен в виде графа

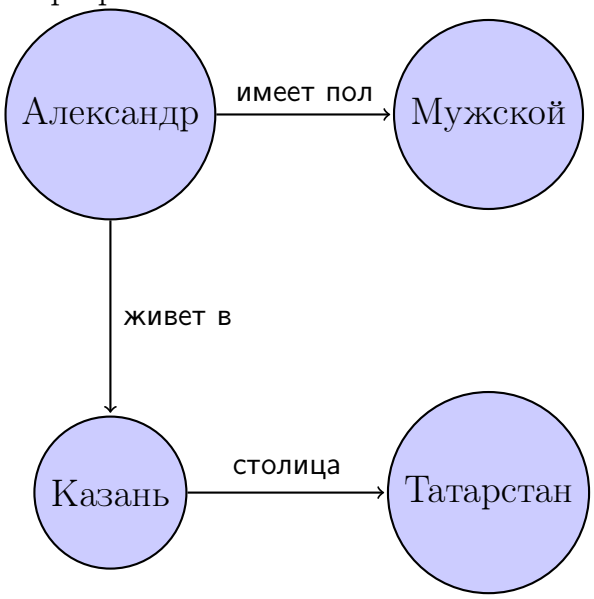


Субъекты и объекты могут быть представлены URI, либо литералом. URI - это уникальный идентификатор, который обозначает сущность: например URI для собаки может быть таким *'http://example.ru/animals/dog'*. Литерал - это просто строка, например 'Jack Nickolson', с возможными добавлениями, указывающими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

2.2. 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 указы-

вать, 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.'*

2.3. RDF хранилище

Требование к строгому определению структуры базы данных характерны для СУБД, ситуация с RDF принципиально другая, ведь RDF был разработан максимально безструктурным. Как отмечалось ранее, это имеет свои преимущества в условиях доступа к произвольным источникам данных в Semantic Web, а также ситуациях, когда данные имеют неизвестные, постоянно меняющиеся структуры. Однако, отсутствие определенной структуры приводит к трудностям при хранении большого количества триплетов и быстрого выполнения запросов. Современные RDF-хранилища могут хранить на порядок меньше данных чем СУБД.

RDF-хранилище может быть спроецировано в СУБД, оптимизированной для RDF-модели с помощью индексов и других тактик. SPARQL запросы при этом транслируются в SQL. В самой простой реализации, RDF представляется с помощью одной таблицы. Если пользоваться правилом нормализации, то появляются еще таблицы, хранящие URI и литералы, а триплеты используют внешние ключи, указывающие на записи в других таблицах.

К несчастью, гибкость RDF представляет собой барьер на пути создания сложных, выразительных представлений. Легкость изменения информации в RDF приводит к сверх-трудностям при создании эффективных

схем хранения. Это можно назвать основным отличием RDF от других представлений.

	Рекомендованное использование	Структура хранимой информации	Запросы
RDF	Произвольное представление знаний	Триплеты	Неизвестный уровень предсказуемости
Реляционные	Поддержка приложений, база данных	Таблицы, предопределенне структуры	Более предсказуемые запросы, оптимизация запросов
Ключ/Значение	Поддержка приложений	Пары ключ-значение	Неизвестный уровень предсказуемости

Таким образом при построении RDF-хранилища необходимо предусмотреть будущие проблемы, связанные с большим количеством триплетов, изменчивостью структуры данных. При этом интерфейс приложения должен быть понятен пользователю.

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:

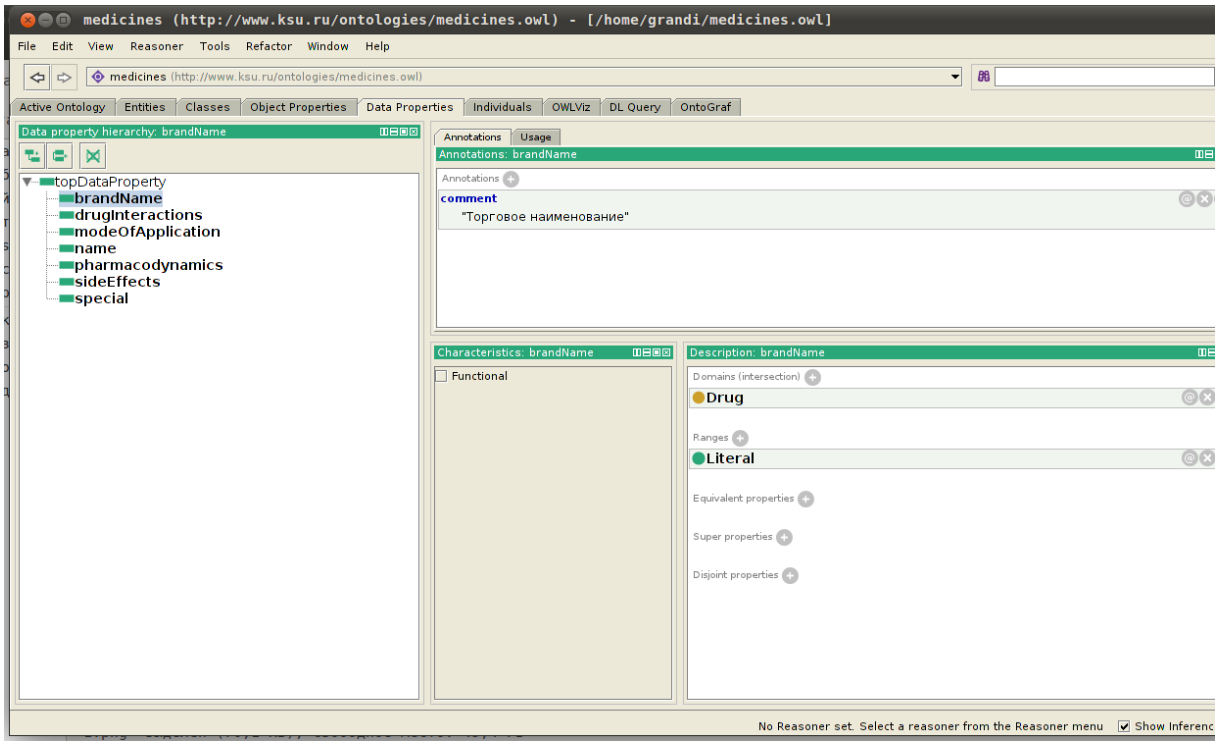
1. имеет международное название
2. имеет торговое наименование
3. принадлежит нескольким категориям
4. имеет множество форм применения
5. имеет фармакологию
6. имеет множество показаний
7. имеет множество противопоказаний
8. имеет множество побочных эффектов
9. имеет дозировку

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.

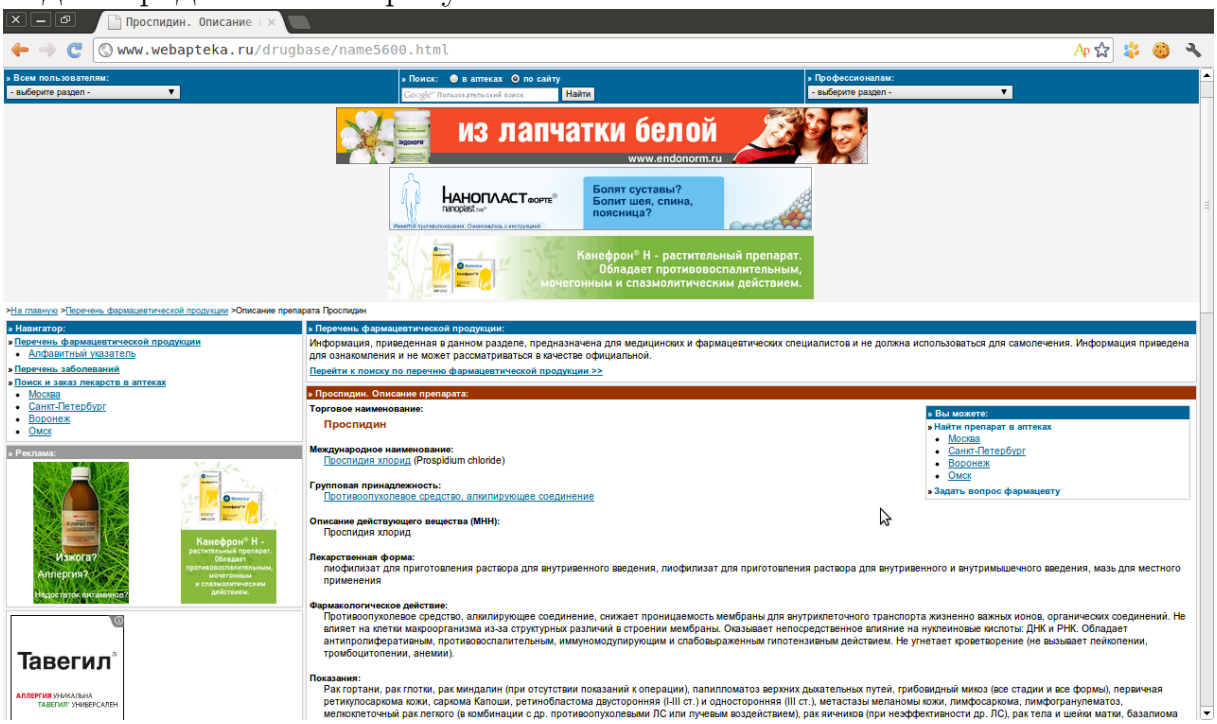
```
include Spira::Resource
base_uri "http://osmanov.me/drugs/"
type URI.new("http://osmanov.me/drugs")
property :intern_title,
  :predicate => URI.new("http://osmanov.me/intern_title"),
  :type => String
property :brandName,
  :predicate => FOAF.name,
  :type => String
has_many :drugCategories,
  :predicate => 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

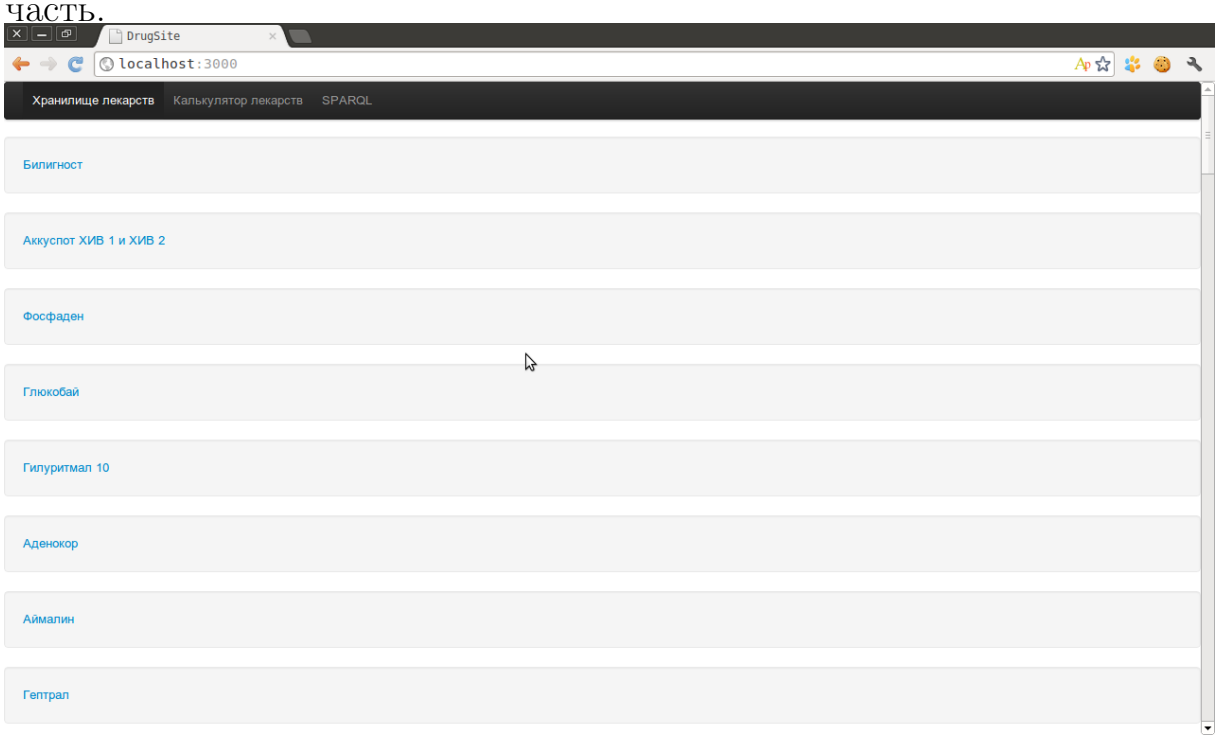
Построение web-интерфейса

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.

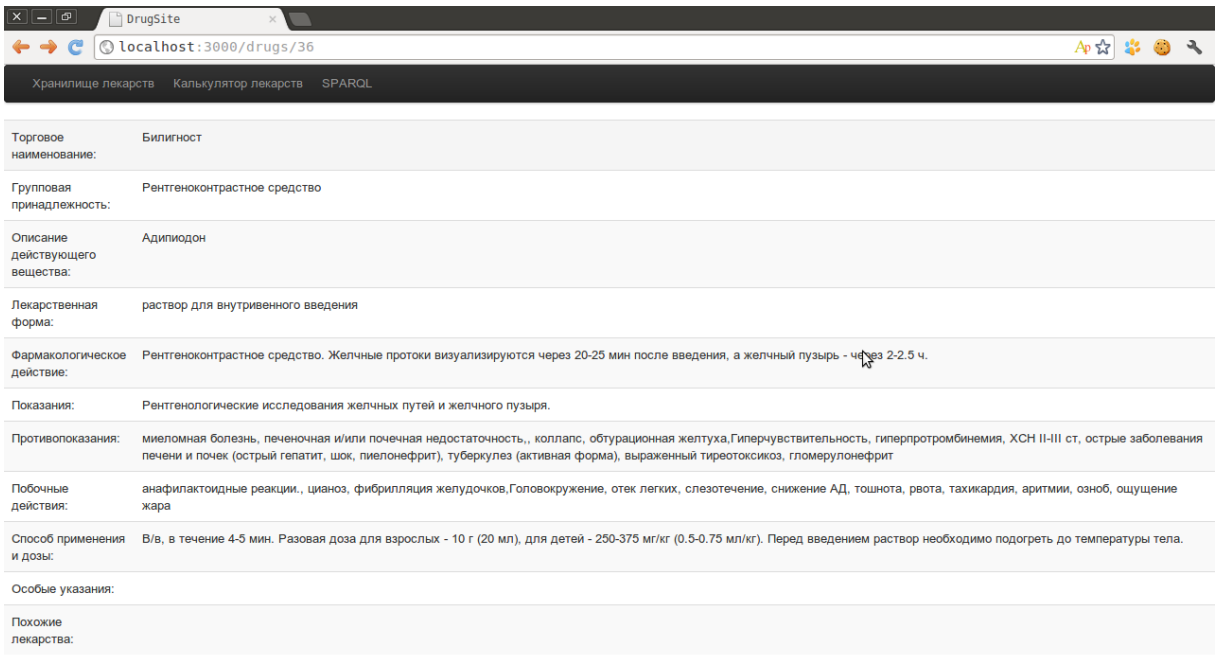
```
<div class="navbar">
  <div class="navbar-inner">
    <div class="container">
      <ul class="nav">
        <li class="active">
                <a href="/">Хранилище лекарств</a>
        </li>
        <li class="">
                <a href="/drugs/calc">Калькулятор лекарств</a>
        </li>
   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

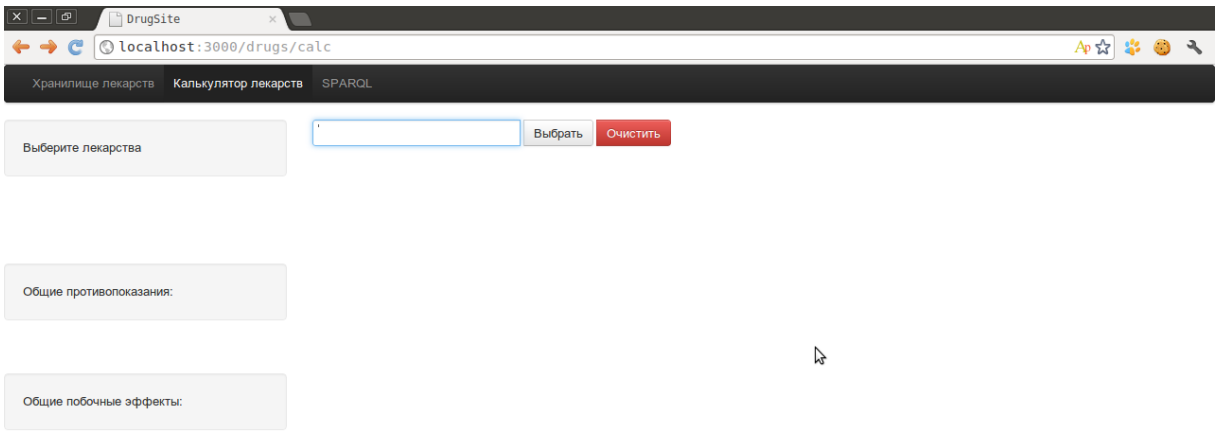


Этот эффект достигается путем добавления классов *table* и *table-striped* к таблице.

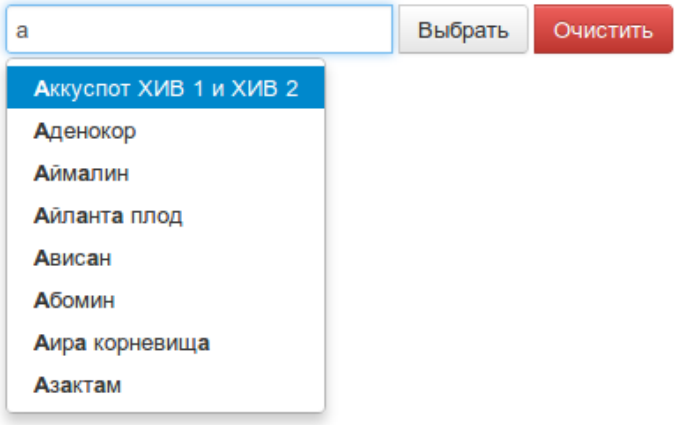
```
<table class="table table-striped">
```

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.