МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Кафедра технологий программирования

ШЕЛЕГ ВЛАДИСЛАВА МИХАЙЛОВНА

Разработка и защита веб-приложений сервис-ориентированной архитектуры, основанных на использовании REST сервисов

Курсовой проект
студента 3 курса 9 группы

Шелег Владиславы Михайловны

студентки 3 курса,
специальность
«Компьютерная безопасность»

Научный руководитель:
Войтешенко Иосиф Станиславович,
доцент кафедры ТП

Минск 2017

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет ____ФПМИ_____ Кафедра _____ТП_____

"**Утверждаю**"
Зав. кафедрой_____
"___" _____ 20__.

ЗАДАНИЕ
ПО ПОДГОТОВКЕ КУРСОВОЙ РАБОТЫ

Студенту Шелег Владиславе Михайловне курса 3 группы 9
1. Тема работы Разработка и защита веб-приложений сервис-ориентированной архитектуры, основанных на использовании REST сервисов
утверждена приказом № _____по Белгосуниверситету от "_____"_____ 20__ г.
2. Срок сдачи студентом законченной работы _____
3. Исходные данные к работе
1. Leonard Richardson, Sam Ruby. RESTful Web Services – Sebastopol, O'Reilly, 2007.
2. Jim Webber, Ian Robbinson. REST in practice – Sebastopol, O'Reilly, 2010.
3. Mark Masse. REST API Design Rulebook – Sebastopol, O'Reilly, 2011.
4. Subbu Allamaraju. RESTful Web Services Cookbook– Sebastopol, O'Reilly, 2010.
5. Другие литературные источники и электронные ресурсы по теме курсового проекта.
4. Перечень вопросов подлежащих разработке или краткое содержание работы
1) Изучение принципов работы архитектуры Rest.
2) Проектирование API для предоставления пользователям возможности получения данных о ресторанах Беларуси.
3) Проектирование Active Job для получения данных с сайта https://carte.by.
4) Реализация API.
5) Реализация Acive Job.
5. Перечень графического материала
5. Перечень графического материала
Диаграммы UML (вариантов использования, деятельности, классов, объектов, компонентов и др.) - не менее 5;
6. Консультанты по работе (с указанием относящихся к ним разделов работы)
_____
7. Дата выдачи задания "___" _____ 20__г.
8. Календарный график работы на весь период
(с указанием этапов работы и сроков их выполнения)
1) 10.09.17 - 30.09.17 — Анализ предметной области
2) 01.10.17 - 27.10.17 — Разработка требований
3) 28.10.17 - 17.11.17 — Проектирование приложения
4) 18.11.17 - 09.11.17 — Разработка приложения
5) 10.12.17 - 15.12.17 — Оформление курсового проекта и подготовка презентации
Руководитель _____ / _____ /
(подпись) (Ф.И.О.)
Задание принял к исполнению "_____" _____ 20___г.
_____
(подпись студента)

Аннотация

Шелег В.М. Разработка и защита веб-приложений сервис-ориентированной архитектуры, основанных на использовании REST сервисов: Курсовой проект / Минск: БГУ, 2017.

В работе рассматриваются основные пункты разработки REST API с использованием языка программирования Ruby, фреймворка Ruby on Rails, и библиотеки Nokogiri. Разработаны методы передачи данных о ресторанах Беларуси и их меню по запросу пользователя в удобном для дальнейшей работы формате JSON.

Анатацыя

Шэлег У.М. Распрацоўка і абарона вэб-прыкладанняў сэрвіс-арыентаванай архітэктуры, заснаваных на выкарыстанні REST сэрвісаў: Курсавы праект / Мінск: БДУ, 2017.

У працы разглядаюцца асноўныя пункты распрацоўкі REST API з выкарыстаннем мовы праграмавання Ruby, фреймворка Ruby on Rails, і бібліятэкі Nokogiri. Распрацаваны метады перадачы дадзеных аб рэстаранах Беларусі і іх меню па запыце карыстальніка ў зручным для далейшай працы фармаце JSON.

Annotation

Sheleg V.M. Development and protection of web applications for a service-oriented architecture based on the use of REST services: Course project / Minsk: BSU, 2017.

The work deals with the main points of development of the REST API are discussed using the Ruby programming language, the Ruby on Rails framework, and the Nokogiri library. Methods for transferring data about restaurants of Belarus and their menus at the request of the user in a format JSON that is convenient for further work have been developed.

РЕФЕРАТ

Курсовой проект, 20 стр., 15 рис., 9 источников.

Разработка и защита веб-приложений сервис-ориентированной архитектуры, основанной на использовании REST сервисов

Ключевые слова: REST, Ruby, Ruby on Rails, Nokogiri, web application, API, SQLite3.

Объект исследования – веб-приложения на основе REST-архитектуры.

Цели работы – изучение основ создания веб-приложений, реализация программного обеспечения для получения данных с сети и дальнейшего их хранения; изучение возможностей библиотеки Nokogiri; реализация веб-приложения для работы с данными о ресторанах Беларуси.

Результат работы – API, позволяющее получать данные о ресторанах Беларуси в удобном для дальнейшей обработки формате; front-end часть, позволяющая продемонстрировать полученые с API данные в удобном для восприятия виде.

Область применения – проектирование и изготовление приложений по доставке еды в Беларуси, навигационных приложений для туристов, путешествующих по РБ.

Введение

В какое бы время мы ни жили, вопрос о питании в конце концов всегда встает для человека на первое места. И что мы можем сделать, если вдруг в нашей насыщенной жизни 21ого века абсолютно нет времени покупать продукты, готовить и т.д.? Мы встаем перед необходимостью найти себе еду где-то в общественном месте, перекусить на скорую руку, не отвлекаясь от своих дел больше положенного времени.

Чтобы получить нужные данные, можно воспользоваться сайтами ресторанов, находящимися в свободном доступе в сети Интернет. Однако, это также отнимает много времени, подобрать подходящий именно вам по вкусам, ценовой категории и другим критериям ресторан, делая множество запросов к разным ресурсам. Мы встаем перед необходимостью собрать все данные в одном месте, чтобы за минимальное количество шагов получить необходимую нам информацию.

Для решения этой проблемы предлагается создать API, способное предоставлять данные о множестве ресторанов Беларуси, например, их местоположение, время работы, меню, включающее в себя перечень блюд, состав и цены.

Такое API может свободно распространяться и выкладываться в сеть Интернет, предоставляя возможность работать с этой информацией другим разработчикам, заинтересованным в данной сфере.

API является максимально удобным решением этой проблемы, так как его непосредственная работа напрямую зависит лишь от запроса, сформированного пользователем. Не зависимо от того, какой язык программирования или платформу разработки использует разработчик, запрашивающий информацию, при корректно сформулированном запросе, будет получен ответ в удобном для дальнейшей обработки данных формате.

Таким образом реализованный продукт становится более доступным и полезным, не зависимо от сферы, в которой планируется его дальнейшее использование, будь то веб-приложение для визуализации этих данных, или навигационное мобильное приложение, позволяющее не только подобрать еду по вкусу, но и предлагающее оптимальный путь до места с необходимым вам меню.

1. Web API

1.1. Общие сведения

Прежде чем давать пояснения, что такое Web API, необходимо разобраться в том, что же такое API.

API (от англ. *Application programming interface*) представляет собой набор готовых подпрограмм, инструментов, предоставляемых приложением или операционной системой для дальнейшего создания внешних программных продуктов, нуждающихся в этих данных. Проще говоря, API - это своего рода интерфейс, который имеет набор функций, которые позволяют разработчикам получать доступ к определенным функциям или данным приложения, операционной системы или других служб.

Web API, как следует из названия, представляет собой API, общение с которым осуществляется через сеть Интернет. К такому API можно подключиться, используя протокол прикладного уровня передачи данных HTTP (от англ. *HyperText Transfer Protocol*). Обмен сообщениями идет по схеме «запрос-ответ», а для идентификации ресурсов используются глобальные URI (от англ. *Uniform Resource Identifier*). Данный протокол не сохраняет своего состояния, то есть каждый запрос относится к независимой транзакции, которая не связана с предыдущими парами «запрос-ответ».

1.2. Клиент-серверная архитектура

Модель клиент-сервер представляет собой распределенную структуру приложения, которая разбивает задачи или рабочие нагрузки между поставщиками услуг, называемыми серверами и заказчиками этих самых услуг, называемыми клиентами. В общем случае, клиент и сервер находятся на разных устройствах, взаимодействуя между собой через одну сеть, используя при этом некоторый протокол передачи данных (к примеру, HTTP). Программа-сервис дожидается запроса в одном из определенных форматов от программы-клиента, обрабатывает его и в зависимости от обработки, предоставляет клиенту ресурсы сервера как данные, либо как сервисные функции.

Клиент не должен волноваться о том, как сервер обрабатывает его запрос и генерирует ответ. Клиент должен только принимать ответ, полученный на основе известного протокола передачи данных.

Плюсом данной модели можно считать то, что клиент не должен быть связан непосредственно с работой сервера, то есть отсутствует дублирование кода. Так же требования к устройству, на котором расположен клиент, существенно снижены, в связи с тем, что все вычислительные операции выполняет сервер. В данной модели проще обеспечить безопасное сохранение данных, так как нет необходимости максимально защищать каждое устройство, которое взаимодействует с ними. Есть необходимость защитить устройство с сервером, а так же организовать контроль полномочий, предоставляемых клиентам, чтобы разрешать доступ к данным только пользователям, имеющим необходимые права доступа.

Однако не существует идеальных моделей. Таким образом, если работа сервера будет нарушена любым из известных способов (техническое обслуживание, поломка, превышение количества клиентов, для которых сервер может обеспечить высокую производительность), то с большой вероятностью и все клиенты, зависящие от него, станут временно неработоспособными. Такая система требует наличия специалиста, отвечающего за ее работоспособность. Выходит, что данная модель требует достаточно материальных вложений, чтобы гарантировать правильную работу без сбоев.

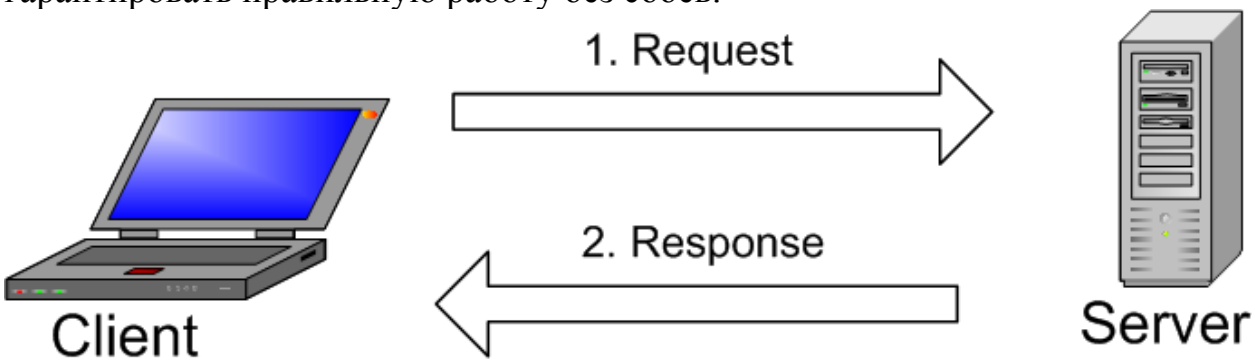

Рисунок 1.1 Принцип работы клиент-серверного приложения

1.3. RESTful API

REST (от англ. *Representational State Transfer*) – архитектурный стиль, предназначенный для обеспечения взаимодействия между компонентами распределенного приложения в сети. Для того, чтобы называться RESTful сервисом, приложение должно выполнять 6 требований:

1. Модель клиент-сервер
2. Отсутствие состояния
3. Кэширование
4. Единообразие интерфейса
5. Слоевая архитектура
6. Код по требованию (не обязательно)

Web API, удовлетворяющий архитектурным ограничениям REST, называется RESTful API. RESTful API, базирующееся на HTTP-запросах, удовлетворяет следующим требованиям:

1. базовый URL-адрес, например http://api.example.com/resources
2. стандартизированный адрес для доступа к ресурсам
3. реализованы стандартные HTTP-методы (GET, POST, DELETE)

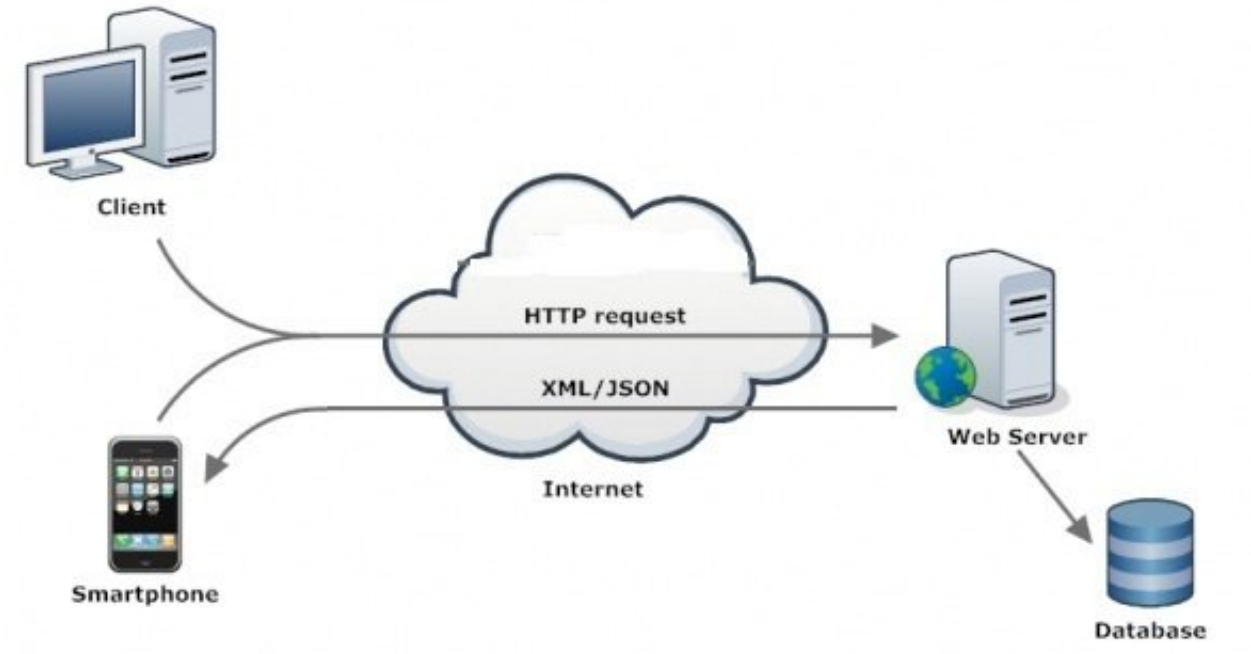

Рисунок 1.2 Принцип работы REST-приложения

Для RESTful Web API нет «официального» стандарта. Это связано с тем, что REST является архитектурным стилем, а не протоколом. REST не является стандартом сам по себе, но реализации RESTful используют стандарты, такие как HTTP, URI и JSON. Многие разработчики также описывают свои API как RESTful, даже несмотря на то, что эти API фактически не выполняют все архитектурные ограничения, описанные выше (особенно унифицированное ограничение интерфейса).

1.4. RESTful API HTTP-методы

HTTP методы	URL(Uniform Resource Locator)	
	`https://api.example.com/resources/`	**`https://api.example.com/resources/item17`**
GET	Сведения обо всех ресурсах данного API.	Сведения о ресурсе, доступном под названием item17.
POST	Создание новой записи. URI присваивается автоматически.	Не используется.
DELETE	Удаление всех ресурсов.	Удаление ресурса, доступного под названием item17.

Метод GET считается безопасным, так как его вызов не влечет никакого изменения данных.

2 Задача проекта

Задача проекта заключается в решении следующих заданий:

- разработка структуры базы данных, необходимой для дальнейшей работы, и заполнение ее с использованием информацию о ресторанах и их меню с сайта https://carte.by/;
- реализация API для предоставления информации о ресторанах по запросу пользователя;
- реализация front-end части, работающей на основе данных API.

Анализируя поставленные задачи, стал вопрос выбора технических средств для реализации проекта. Решено было выбрать фреймворк Ruby on Rails, являющийся открытым программным обеспечением. Данный фреймворк реализует архитектурный шаблон MVC (от англ. *Model-View-Controller*) для веб-приложений, позволяет обеспечить интеграцию с веб-сервером и базой данных [3]. Для хранения данных была выбрана компактная встраиваемая реляционная база данных SQLite, как позволяющая уменьшить накладные расходы и время отклика. Так же SQLite позволяет быстрое оперирование данными, что существенно уменьшает время отклика API на запрос [4].

3. Особенности реализации API

3.1. Структура компонентов ПО

Файл/папка	Назначени
app/	Содержит контроллеры, модели, задания и т.п. приложения.
bin/	Содержит Rails скрипты, обеспечивающие работу приложения.
config/	Конфигурации маршрутов, базы данных и т.п.
db/	Содержит текущую схему БД, а также миграции.
lib/	Внешние модули приложения.
log/	Файлы логов приложения.
test/	Аппарат тестирования.
tmp/	Временные файлы.
vendor/	Место для кода внешних разработчиков.
.gitignore	Файл, сообщающий git, какие файлы ему следует игнорировать.
config.ru	Конфигурация для сервера, использующегося для запуска приложения.
Gemfile Gemfile.lock	Файлы, показывающие зависимость приложения от гемов Ruby.
Rakefile	Файл находит и загружает задачи, которые могут быть запущены в Ruby console.
README.md	Вводный мануал приложения.

В самом начале работы возникает вопрос, как правильно реализовать хранение данных и работу с ними в будущем. Проанализировав информацию, с которой надо взаимодействовать, решено создать модели Region, City, Restaurant, Category, Dish.

```
class Restaurant < ApplicationRecord
  belongs_to :city
  has_many :categories

  def restaurant_params
    params.require(:restaurant).permit(:name, :address, :worktime, :phone, :average,
                                       :bookable, :orderable, :images, :logo)
  end
end
```

Рисунок 3.1 Код модели Restaurant.

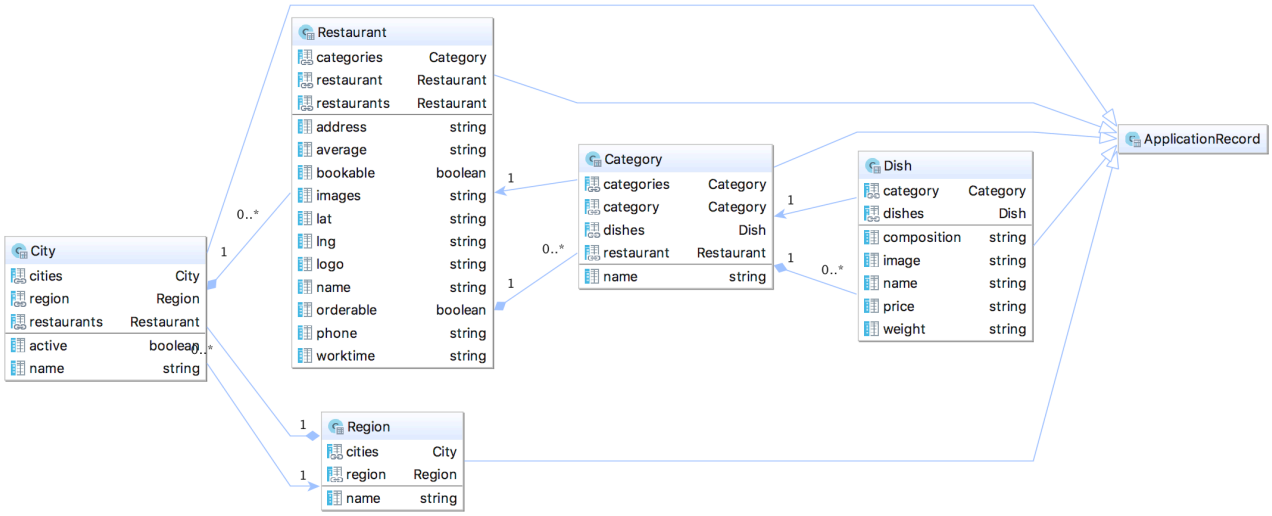

Рисунок 3.2 Диаграмма моделей проекта.

Специфика приложений на RoR в том, что работа с базой данных осуществляется не напрямую, а посредством миграций. Миграции позволяют работать не с «чистым» языком SQL, а использовать Ruby DSL для описания изменений в базах данных, а также работать с различными версиями БД, то есть разработчик имеет возможность «откатить» все изменения своей базы данных до нужного ему этапа.

Для дальнейшей работы была спроектирована следующая база данных, включающая в себя связь между таблицами внутри нее, для более удобного обращения к связанным между собой данным:

```
ActiveRecord::Schema.define(version: 20171206223137) do

  create_table "categories", force: :cascade do |t|
    t.string "name"
    t.integer "restaurant_id"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index ["restaurant_id"], name: "index_categories_on_restaurant_id"
  end

  create_table "cities", force: :cascade do |t|
    t.string "name"
    t.boolean "active"
    t.integer "region_id"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index ["region_id"], name: "index_cities_on_region_id"
  end

  create_table "dishes", force: :cascade do |t|
    t.string "name"
    t.string "weight"
    t.string "price"
    t.string "composition"
    t.integer "category_id"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.string "image"
    t.index ["category_id"], name: "index_dishes_on_category_id"
  end

  create_table "regions", force: :cascade do |t|
    t.string "name"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
  end

  create_table "restaurants", force: :cascade do |t|
    t.string "name"
    t.string "address"
    t.string "worktime"
    t.string "phone"
    t.string "average"
    t.boolean "bookable"
    t.boolean "orderable"
    t.integer "city_id"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.string "images"
    t.string "logo"
    t.index ["city_id"], name: "index_restaurants_on_city_id"
  end
```

Рисунок 3.3 Схема БД приложения.

На данном этапе, имея связи между элементами базы данных, необходимо указать взаимосвязь ресурсов.

```
resources :regions do
  resources :cities do
    resources :restaurants do
      resources :categories do
        resources :dishes
      end
    end
  end
end
```

Рисунок 3.4 Иерархия ресурсов приложения.

3.2. Веб-парсинг

На следующем этапе возникает вопрос, как же реализовать заполнение базы данных, чтобы в дальнейшем не возникло проблем с совместимостью данных из БД и моделей, реализованных для работы с ними? Возможности Ruby on Rails предлагают для реализации этой задачи фреймворк Active Job, позволяющий запускать фоновые процессы в порядке условий их добавления в очередь.

Для начала необходимо получить саму HTML-страницу с интересующими данными. На Ruby реализовано множество гемов, подходящих для выполнения поставленной задачи. В данном случае был выбран гем HTTParty, как наиболее простой в использовании.

```
def perform
  page = Nokogiri::HTML(HTTParty.get('https://carte.by/'))
```

Рисунок 3.4 Получение HTML-кода страницы https://carte.by/.

Получив страницу, необходимо с помощью гема Nokogiri для парсинга HTML и XML объектов получить информацию с данной страницы, сохранив ее для дальнейшей обработки в структуре Hash. Так как Active Job работает непосредственно в проекте приложения, имеется доступ к моделям RoR, что облегчает хранение данных.

```
regions = Hash.new{|hsh,key| hsh[key] = [] }
page_rest.css(".vr-region").each do |region|
  region_name = region.css(".vr-region-name").text.strip
  region_model = Region.new({name: region_name})

  region.css(".vr-region-cities a").each do |city|
    href = city["href"]
    name = city.text

    regions[region_model].push [City.new({name: name, active: 1}), href]
  end
  region.css(".vr-region-cities span").each do |city|
    href = nil
    name = city.text

    regions[region_model].push [City.new({name: name, active: 0}), href]
  end
end
```

Рисунок 3.5 Фрагмент парсинга данных HTML-кода. Получение списка городов, доступных для каждого региона.

3.3. Логика API

Для того, чтобы в дальнейшем можно было реализовать методы, доступные пользователю, необходимо написать контроллер для каждой модели, которая реализована. Для того, чтобы реализовать GET-методы, необходимые для деятельности API, нужно реализовать методы, стандартно называющиеся index и show, для получения информации о доступных городах, ресторанах и их меню. Так же пользователю может быть необходимо предоставить возможность получения списка всех блюд ресторана без разделения их на категории.

```
class CategoriesController < ApplicationController
  def index
    categories = Category.where(restaurant_id: params[:restaurant_id])
    render :json => categories.to_json(:except => [:created_at, :updated_at])
  end

  def show
    category = Category.find(params[:id])
    render :json => category.to_json(:except => [:created_at, :updated_at])
  end

  def all
    dishes = []
    categories = Category.where(restaurant_id: params[:restaurant_id])
    categories.each do |category|
      dishes.push category.dishes
    end
    render :json => dishes.to_json(:except => [:created_at, :updated_at])
  end
end
```

Рисунок 3.6 Реализация GET-методов для модели Categoria.

4. Front-end разработка

4.1. AJAX

AJAX (от англ. *Asynchronous Javascript and XML)* – подход к построению интерактивных пользовательских интерфейсов веб-приложений. Особенностью его является «фоновый» обмен данными между браузером и веб-сервером. В результате, при обновлении какого-либо блока данных на веб-странице не происходит полной перезагрузки, что делает веб-приложения быстрее и удобнее. В классической модели веб-приложения обновление данных происходит по следующему алгоритму:

- Пользователь заходит на веб-страницу и нажимает на какой-либо элемент, провоцирующий обновление данных.
- Браузер формирует запрос и отправляет его серверу.
- Сервер генерирует абсолютно новую веб-страницу, отправляет ее браузеру. После получения данного ответа браузер перезагружает страницу.

При использовании же AJAX этот устоявшийся алгоритм несколько изменяется:

- Пользователь заходит на веб-страницу и нажимает на какой-либо элемент, провоцирующий обновление данных.
- Скрипт, написанный на языке Javascript, определяет, какая именно информация нужна для обновления страницы.
- Браузер генерирует соответствующий запрос и отправляет его на веб-сервер.
- Сервер возвращает ответ на запрос. В качестве формата для передачи данных могу быть использованы фрагменты простого текста, HTML-кода, JSON и XML.
- Скрипт вносит изменения только в выбранный пользователем блок данных.

На основе этих данных уже можно сделать вывод о некоторых существенных плюсах такого подхода, например:

- Экономия трафика.
- Уменьшение нагрузки на сервер.
- Ускорение реакции интерфейса.

4.2. Реализация Front-end

Исходя из возможностей API и вероятных потребностей пользователя, необходимо реализовать демонстрацию регионов, просмотра городов, доступных в данном регионе, а далее визуализировать информацию о ресторанах, соответствующих выбранному городу.

```
function insertValues() {
    var regions = document.getElementById("regions");
    var select = document.getElementById("cities");
    var id = regions.options[regions.selectedIndex].id;

    var request = new XMLHttpRequest();
    request.open('Get', 'http://localhost:3000/regions/' + id + "/cities");

    request.send();


    request.onreadystatechange = function () {
        if (request.readyState === 4) {
            if (request.status === 200) {
                var cities = JSON.parse(request.responseText);
                select.innerHTML = null;
                cities.forEach(function (city) {
                    var txtVal = city.name,
                        newOption = document.createElement("OPTION"),
                        newOptionVal = document.createTextNode(txtVal);
                    if (city.active == false) {
                        newOption.disabled = true;
                    }
                    newOption.appendChild(newOptionVal);
                    newOption.setAttribute("id", String(city.id));
                    select.insertBefore(newOption, select.lastChild);
                });

            } else {
                data.innerHTML = 'Произошла ошибка при запросе: ' + request.status + ' ' + request.statusText;
            }
        }
    }
```

Рисунок 4.1 Фрагмент кода получения и визуализации списка городов, доступных для данного региона.

```
var newCard = document.createElement("div");
newCard.className = "card";
var newRow = document.createElement("div");
newRow.className = "row";
var newCol1 = document.createElement("div");
newCol1.className = "col-md-4";
var img = new Image;
img.src = "http://iphex-india.com/assets/images/nologo.png";
if (restaurant.logo)
    img.src = restaurant.logo;
img.setAttribute("height", "200");
img.setAttribute("width", "200");
newCol1.appendChild(img);
var newCol2 = document.createElement("div");
newCol2.className = "col-md-8";
var newCardBlock = document.createElement("div");
newCardBlock.className = "card-block px-5";
var h5 = document.createElement("h5");
h5.className = "card-title center";
var link = document.createElement("a");
link.setAttribute("href", 'http://localhost:3000/regions/' + region_id + "/cities/"
+ city_id + "/restaurants/" + restaurant.id);
var textLink = document.createTextNode(restaurant.name);
link.appendChild(textLink);
h5.appendChild(link);
newCol2.appendChild(h5);
if (restaurant.orderable == true) {
    var spanD = document.createElement("span");
    spanD.className = "badge badge-success";
    var textD = document.createTextNode("Доставка");
    spanD.appendChild(textD)
    newCol2.appendChild(spanD);
}
newRow.appendChild(newCol1);
newRow.appendChild(newCol2);
newCard.appendChild(newRow);
rests.insertBefore(newCard, rests.lastChild);
```

Рисунок 4.2 Фрагмент кода получения и визуализации списка ресторанов, доступных в выбранном городе.

5. Результаты работы

В результате работы было реализовано API, посылающее ответы в формате JSON в следующем виде:


```
{
    "id": 1,
    "name": "Минск",
    "active": true,
    "region_id": 1
},
{
    "id": 2,
    "name": "Солигорск",
    "active": true,
    "region_id": 1
},
{
    "id": 3,
    "name": "Борисов",
    "active": true,
    "region_id": 1
},
{
    "id": 4,
    "name": "Жодино",
    "active": true,
    "region_id": 1
},
```


Рисунок 5.1 Фрагмент JSON, полученного по запросу http://localhost:3000/regions/1/cities.


```
{
    "id": 22,
    "name": "Аура",
    "address": "ул. Смоленская, 15а",
    "worktime": "{\"Пн\"=>\"11:00 - 07:00\", \"Вт\"=>\"11:00 - 07:00\", \"Ср\"=>\"11:00 - 07:00\", \"Чт\"=>\"11:00 - 07
        :00\", \"Пт\"=>\"11:00 - 07:00\", \"Сб\"=>\"11:00 - 07:00\", \"Вс\"=>\"11:00 - 07:00\"}",
    "phone": "[\"tel:+375295550505\", \"tel:+375445550505\", \"tel:+375173970000\", \"tel:+375445260000\"]",
    "average": "32 руб.",
    "bookable": true,
    "orderable": false,
    "city_id": 1,
    "images": "[]",
    "logo": "",
    "lng": "53.892362",
    "lat": null
}
```


Рисунок 5.2 Фрагмент JSON, полученного по запросу http://localhost:3000/regions/1/cities/1/restaurants/22.

Для предоставления пользователю возможности выбора региона и города для дальнейшей работы, был выбран элемент выпадающий список.

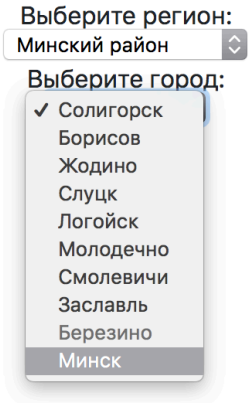

Рисунок 5.3 Выпадающий список, содержащий города, доступные для Минского района.

Для визуализации карточек-ресторанов был использован Bootstrap. **Bootstrap** – свободно распространяющийся набор инструментов для создания сайтов и веб-приложений. Включает в себя HTML- и CSS-шаблоны оформления для веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения.

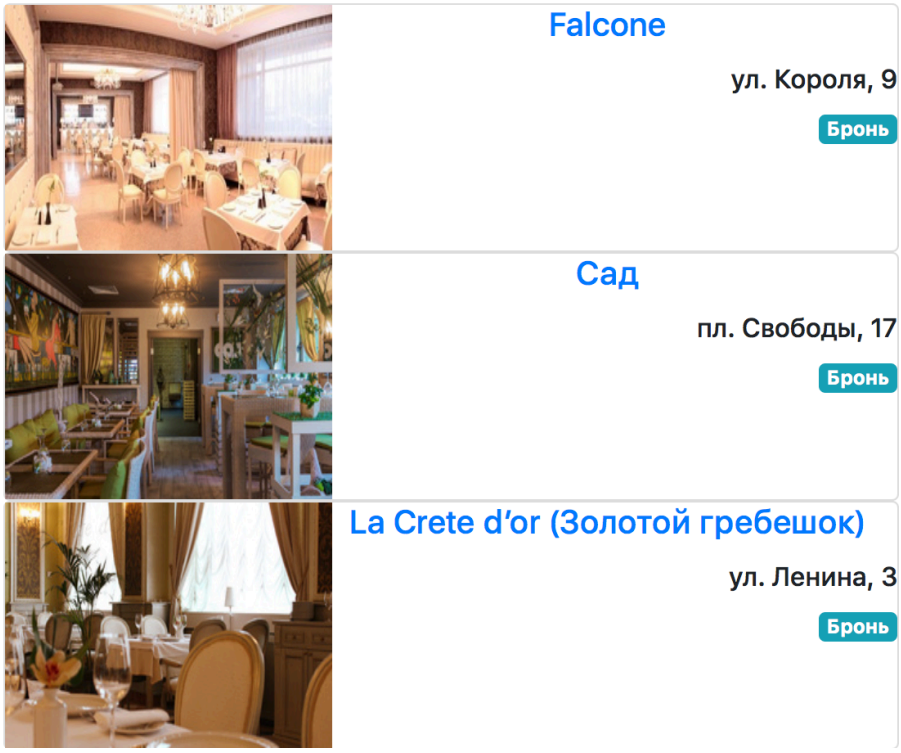

Рисунок 5.4 Фрагмент списка ресторанов, доступных в городе Минске.

Заключение

В процессе реализации курсового проекта были изучены основы работы с фреймворком Ruby on Rails, библиотеками Nokogiri и HTTParty, реализовано API, позволяющее осуществлять доступ к базе данных, содержимое которой хранит подробную информацию о ресторанах Беларуси.

Реализовано простейшая демонстрация получения данных от API на стороне клиента, произведено ознакомление с таким подходом к созданию веб-приложений, как AJAX. Также было продемонстрировано, каким образом можно использовать сторонние источники, такие как Bootstrap, в своем приложении, для упрощения реализации.

В дальнейшем планируется вплотную заняться обеспечением безопасности уже написанного приложения, к примеру, реализовать HTTP-методы POST и DELETE в API, позволяющие непосредственное изменение содержимого БД аутентифицированным пользователям с необходимыми правами доступами.

Список литературы

- 1. REST-сервис [Электронный ресурс] – https://en.wikipedia.org/wiki/Representational_state_transfer
- 2. HTTP-протокол передачи данных [Электронный ресурс] – https://ru.wikipedia.org/wiki/HTTP
- 3. Ruby on Rails Documentation [Electronic resourse] – http://api.rubyonrails.org
- 4. Database Speed Comparison [Electronic resourse] – https://sqlite.org/speed.html
- 5. Leonard Richardson, Sam Ruby. RESTful Web Services – Sebastopol, O'Reilly, 2007.
- 6. Jim Webber, Ian Robbinson. REST in practice – Sebastopol, O'Reilly, 2010.
- 7. Mark Masse. REST API Design Rulebook – Sebastopol, O'Reilly, 2011.
- 8. Subbu Allamaraju. RESTful Web Services Cookbook – Sebastopol, O'Reilly, 2010.