ВВЕДЕНИЕ

Кластеризация (или кластерный анализ) – это задача разбиения множества объектов на группы, называемые кластерами. Внутри каждой группы должны оказаться «похожие» объекты, а объекты разных групп должны быть как можно более отличны. Главное отличие кластеризации от классификации состоит в том, что перечень групп чётко не задан и определяется в процессе работы алгоритма.

В биологии кластеризация имеет множество приложений в самых разных областях. Например, в биоинформатике с помощью неё анализируются сложные сети взаимодействующих генов, состоящие порой из сотен или даже тысяч элементов. Кластерный анализ позволяет выделить подсети, узкие места, концентраторы и другие скрытые свойства изучаемой системы, что позволяет в конечном счете узнать вклад каждого гена в формирование изучаемого феномена.

В области экологии широко применяется для выделения пространственно однородных групп организмов, сообществ и т. п. Реже методы кластерного анализа применяются для исследования сообществ во времени. Гетерогенность структуры сообществ приводит к возникновению нетривиальных методов кластерного анализа (например, метод Чекановского).

В общем стоит отметить, что исторически сложилось так, что в качестве мер близости в биологии чаще используются меры сходства, а не меры различия (расстояния).

**1** ОБЗОР ЛИТЕРАТУРЫ

Рассматривая существующие популярные платформы для разработки программного обеспечения, а также с учетом всевозможных достоинств и недостатков известных технологий, сред разработок, операционных систем, были выбраны следующие компоненты и технологии: для написания программного кода будет использоваться язык программирования C# совместно с поставляемой корпорацией Майкрософт библиотекой .NET Framework. При первоначальном знакомстве с языком программирования C# не будет лишним изучить базовые конструкции и общие правила языка [1], а также получить представление о принципах объектно-ориентированного программирования.

Далее, для более полного понимания всего, что будет происходить с программным кодом, необходимо углубиться в архитектуру и устройство библиотеки .NET Framework [2,3]. Библиотека .NET Framework состоит из двух частей: общеязыковой исполняющей среды (Common Language Runtime, CLR) и библиотеки классов Framework Class Library (FCL).

Программа для .NET Framework, написанная на любом поддерживаемом языке программирования, сначала переводится компилятором в единый для .NET промежуточный байт-код Common Intermediate Language (CIL) (ранее назывался Microsoft Intermediate Language, MSIL). В терминах .NET получается сборка. Затем код либо исполняется виртуальной машиной CLR, либо транслируется в исполняемый код для конкретного целевого процессора. Использование виртуальной машины предпочтительно, так как избавляет разработчиков от необходимости заботиться об особенностях аппаратной части. В случае использования виртуальной машины CLR, встроенный в неё JIT-компилятор «на лету» (just in time) преобразует промежуточный байт-код в машинные коды нужного процессора. Современная технология динамической компиляции позволяет достигнуть высокого уровня быстродействия. Виртуальная машина CLR также сама заботится о базовой безопасности, управлении памятью и системе исключений, избавляя разработчика от части работы.

Архитектура .NET Framework (см. рисунок 1.1) описана и опубликована в спецификации Common Language Infrastructure (CLI), разработанной Microsoft и утверждённой ISO и ECMA. В CLI описаны типы данных .NET, формат метаданных о структуре программы, система исполнения байт-кода и многое другое. Объектные классы .NET, доступные для всех поддерживаемых языков программирования, содержатся в библиотеке Framework Class Library (FCL). В FCL входят классы Windows Forms, ADO.NET, ASP.NET, Language Integrated Query, Windows Presentation Foundation, Windows Communication Foundation и другие. Ядро FCL называется Base Class Library (BCL).

Base Class Library (BCL) – стандартная библиотека классов платформы .NET Framework. Программы, написанные на любом из языков, поддерживающих платформу .NET, могут пользоваться классами и методами BCL — создавать объекты классов, вызывать их методы, наследовать необходимые классы BCL и так далее.

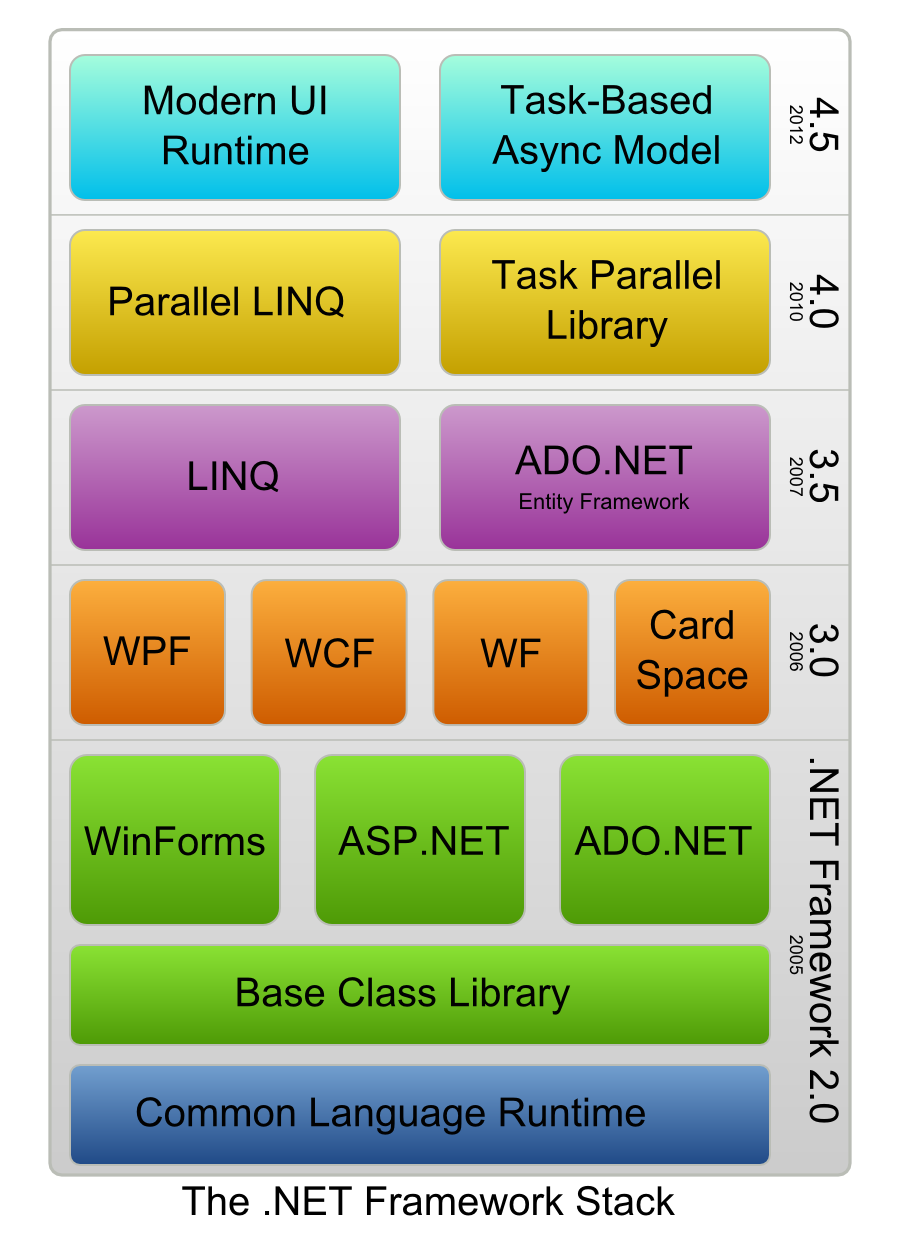


Рисунок 1.1 – Cтруктура стека платформы .NET Framework

Следует отметить следующие достоинства .NET Framework [2]:

1. Единая программная модель. В отличие от существующего подхода, когда одни функции ОС доступны через процедуры динамически подключаемых библиотек (DLL), а другие – через СОМ-объекты, весь прикладной сервис представлен общей объектно-ориентированной программной моделью.

2. Упрощенная модель программирования. CLR избавляет от работы с разными потаенными структурами, как это было с Win32 и СОМ. Так, разработчику не нужно разбираться с реестром, глобально-уникальными идентификаторами (QUID), Release, HRESUIT и так далее. CLR не просто позволяет разработчику абстрагироваться от этих концепций – их просто нет в CLR в каком бы то ни было виде.

3. Работа на нескольких платформах. При компиляции кода для .NET Framework компилятор генерирует код на общем промежуточном языке, а не традиционный код, состоящий из процессорных команд. Поскольку трансляция выполняется в период выполнения, генерируются команды конкретного процессора. Это значит, что приложение для .NET Framework может быть развернуто на любой машине, где работает версия CLR и FCL.

4. Автоматическое управление памятью. Одна из самых распространенных ошибок – небрежное отношение к освобождению этих ресурсов, что может привести к некорректному выполнению программы в непредсказуемый момент. CLR автоматически отслеживает использование ресурсов, гарантируя, что не произойдет их утечки. По сути она исключает возможность явного «освобождения» памяти.

5. Проверка безопасности типов. CLR может проверять безопасность использования типов в коде, что гарантирует корректное обращение к существующим типам. Безопасность типов также означает, что управление может передаваться только в определенные точки (точки входа методов). Невозможно указать произвольный адрес и заставить программу исполняться, начиная с этого адреса. Совокупность всех этих защитных мер избавляет от многих распространенных программных ошибок.

Большое количество информации о языке программирования C# можно найти в фактически первоисточнике – официальной спецификации, разработанной корпорацией Майкрософт [4]. В ней содержится информация о том, как должен выглядеть код, какие вещи должны поддерживаться компилятором. Спецификация – весьма строгий и сухой документ, который не может служить учебником по программированию. Она обычно содержит не слишком много примеров, не дает рекомендаций по применению тех или иных конструкций, а лишь определяет их форму и смысл. Это затрудняет её применение на практике. Потому имеет смысл при написании кода обращаться к более подходящей документации, которая в первую очередь ориентирована на программиста, который пишет на данном языке программирования. В случае с языком программирования C# это документация MSDN [5]. Достоинство MSDN заключается в том, что весь материал в нем хотя и является технической информацией, однако адаптированной на использование. Многие статьи в нем подкреплены примерами, дающими ключ к понимаю принципов использования или функционирования той или иной функции. Часто встречаются советы и рекомендации по использованию того или иного метода в своей программе.

Следует заметить, что большая часть документации по платформе .NET присутствует в переведенном на русский язык варианте, что существенно облегчает понимание.

Существует достаточно большое количество инструментов и решения для работы с картами на стороне клиента. У каждого картографического сервиса есть свои библиотеки по работе с API и отрисовкой карты. Но все они привязаны к конкретному провайдеру карт. Лишь один продукт реализовал независимость от картографических сервисов в полной мере – Lefalet – библиотека для управления геоданными на веб-карте в браузере. Leaflet задумана как библиотека, одинаково хорошо работающая и в современных браузерах, и на мобильных устройствах (iPhone/iPad, Android, Windows Phone) — очень быстрая, легковесная, с простым API, гибким и понятным ООП-кодом [6]. Кратко рассмотрим основные преимущества данной библиотеки.

API для отображения GeoJSON-слоёв был разработан так, чтобы сделать более гибким и простым в использовании GeoJSON данные. Маркеры, векторные объекты и слои из изображений плавно анимируются вместе с базовым изображением карты. Если же у вас на карте много сотен маркеров и анимация происходит не очень плавно (скажем, больше 1000 в случае Chrome), ее можно отключить опцией карты markerZoomAnimation, хотя в таком случае было бы более целесообразно использовать плагин кластеризации маркеров [7].

Доступность Leaflet-карт заметно улучшилась благодаря новой возможности управления с помощью клавиатуры, включённой по умолчанию. Она позволяет управлять картой, нажимая клавиши стрелок.

Управлять картойс помощью мышки или пальцами комфортно засчет эффекта инерции, при котором карта продолжает плавно замедляться в том же направлении, в которое вы ее дёрнули. Этот эффект достаточно гибко настраивается — можно контролировать максимальную начальную скорость, замедление и порог чувствительности, при котором инерция срабатывает.

Простой индикатор текущего масштаба в метрический и/или имперской системах измерения. Как и предполагалось, внешний вид можно полностью изменить с помощью CSS.

LeafletJS позволяет редактировать ломаные линии и полигоны с помощью простого, интуитивно понятного интерфейса. В добавок к традиционным иконкам для маркеров из изображений, используются легковесные иконки, представляющие собой обычный div блок c определённым HTML внутри, стилизованный внешним CSS.

Кроме того, с момента предпоследней версии Leaflet содержит около 45 исправлений, делающих библиотеку более стабильной и надёжной среди всех поддерживаемых браузеров и платформ. Данная библиотека полностью описана в документации и отражает актуальную информацию от версии к версии. Все классы, методы, свойства, опции и события тщательно задокументированы и сопровождены большим количеством примеров кода, что делает эту библиотеку очень популярной среди разработчиков любых систем и приложений независимо от того, какие технологии последние используют.

Существует два типа современных веб-карт: распределенные картографические веб-сервисы c интеграцией собственных данных в карты и веб-сервисы предоставляющие хранение пользовательских данных на своей стороне, их анализ и другие инструменты по работе с данными. Рассмотрим первый тип карт. К ним относятся:

1. Google Maps
2. Yahoo! Maps
3. Bing Maps
4. Яндекс.Карты

Эти сервисы выделяются широким спектром предоставляемых функциональных возможностей, хорошим географическим покрытием и удобством для конечных пользователей. К числу их сильных сторон можно отнести развитые навигацию по карте и масштабирование (zooming), наличие специализированных информационных ресурсов (карты улиц крупных городов, транспортных потоков в режиме реального времени и прочее).

Все ведущие картографические сервисы поддерживают последние версии популярных браузеров (IE, Firefox, Safari, Opera, Chrome). При этом Google Maps доступен для пользователей наибольшего числа версий браузеров, включая уже вышедшие из употребления - благодаря своей давней истории (первая версия сервиса была опубликована в 2005 г.) [8] и изначально кросс-браузерному дизайну, заложенному в его клиентское программное обеспечение.

Если говорить об источниках картографических данных, то общей чертой этих сервисов является сотрудничество со специализированными организациями, у которых они закупают данные.

Безусловно, сервисы различаются функциональностью. Например, картографическая служба Microsoft Bing Maps предоставляет возможность навигации по трехмерным картам улиц крупнейших городов США и некоторых других стран [9], в то время как Google дает уникальный инструмент для разработчиков, желающих разместить ту или иную прикладную информацию на картах, - Google Maps API.

Также эти сервисы отличаются по степени покрытия различных районов земного шара и актуальностью картографических данных. Например, карты Яндекса для территорий стран СНГ отвечают последнему требованию. Актуальность карт в Google Maps и Microsoft Bing Maps для разных регионов - 1-3 года. Отстает от всех Yahoo! Maps - его данные в среднем устарели на 4-5 лет.

Существенным минусом также является закрытие Yahoo! Maps своего API 13 сентября 2011 г. [10].

Каждый сервис имеет свои преимущества и недостатки. В зависимости от поставленной задачи можно применять разные веб-сервисы.

Современные картографические веб-сервисы, несомненно, предоставляют массу возможностей, как специалистам, так и пользователям интернета. Однако у них имеется целый ряд недочетов, связанных с применением коммерческих данных (недостаточное или отсутствующее покрытие в определенных регионах земного шара, устаревшая картографическая информация и так далее).

Именно это послужило толчком для развития альтернативных картографических веб-решений - коллективных веб-карт. Их отличительной чертой является возможность для пользователей самим создавать и обновлять данные на картах. В результате получается достаточно достоверная и совершенно бесплатная карта региона и мира в целом.

Среди наиболее ярких решений данного класса - проект OpenStreetMaps (OSM). Проект развился и на сегодняшний день превратился в достаточно массовое движение GPS-картографирования. Одна из важных особенностей OSM - развитие набора инструментов для автоматического ввода данных, полученных пользователями с помощью GPS-картографирования, в централизованное хранилище. Именно благодаря этому в системе появляются и обновляются карты различных участков земного шара.

Популяризации данных OSM способствовало создание основателями OSM компании Cloudmade, которая разрабатывает линейку коммерческих продуктов, использующих данные OSM (а именно, картографические Web API и MobileAPI [11].

Другим примером успешной коллективной веб-карты является проект WikiMapia, созданный россиянами Александром Корякиным и Евгением Савельевым в 2006 г. [12] Это надстройка над Google Maps, в которой применяется wiki-подобный интерфейс для ввода пользовательских данных.

Теперь же рассмотрим второй тип сервисов, а именно, веб-сервисы, предоставляющие хранение пользовательских данных на своей стороне с неким инструментарием по работе с ними.

CartoDB - геопространственная база данных, работающая в «облаке», которое предполагает хранение и визуализацию данных [13] прямо на клиентской стороне в браузере (см. рисунок 1.2).

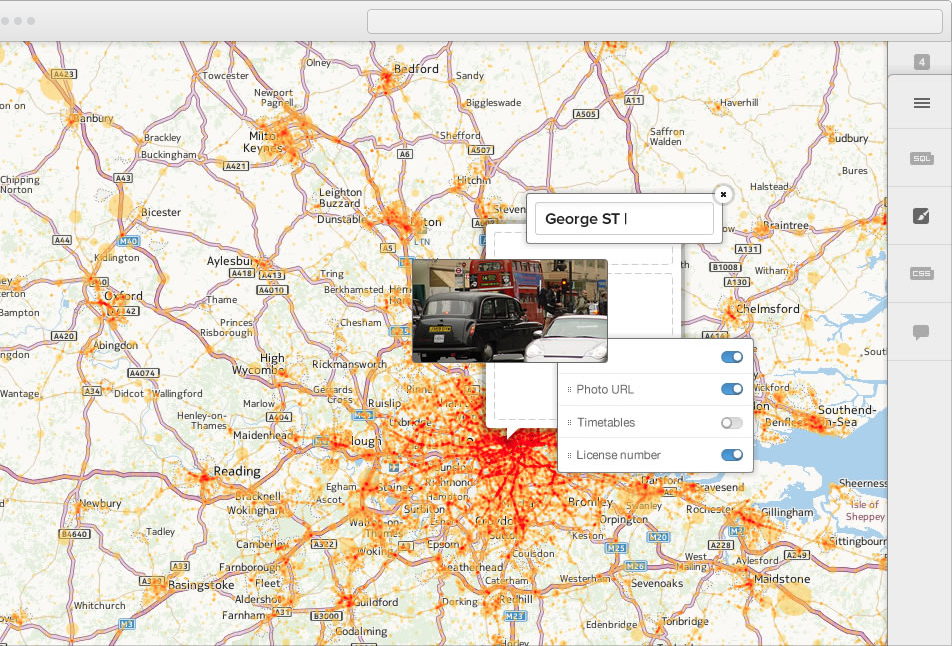


Рисунок 1.2 – Пример веб-карты и интегрированных в неё данных в веб-сервисе CartoDB

Используя CartoDB можно быстро создать карту визуализацию пользовательских данных. Вот несколько решений, которые предоставляются сервисом CartoDB:

1. Загрузка, визуализация и управление своими данными, используя инструментальную панель CartoDB.
2. Быстрое создание и настройка карты, которую Вы можете встроить или совместно использовать через общедоступный URL, используя инструмент встраивания карты.
3. Анализ и интеграция данных, которые Вы храните в хранилище CartoDB, в приложения, используя API SQL.
4. Использование библиотеки CartoDB.js для большей интеграции и усовершенствования карт CartoDB на веб-сайте или приложении.

CartoDB дает доступ к PostGIS, включая базу данных PostgreSQL, содержащую данные. Эти геопространственные базы данных не только дают полный контроль пользовательских данных, но и комплект инструментов с безграничным потенциалом для анализа этих данных. Сервис обеспечивает мощное управление данными и геопространственный аналитический бэкэнд. Среди многих функций CartoDB включает API Maps, который предоставляет спроектированные мозаики карты, чтобы создать быструю визуализацию данных, и API SQL, который позволяет запрашивать свои данные, или посредством безопасного соединения, записывает новые данные в таблицы CartoDB.

Обеспечение безопасности и безопасного хранения данных – один из самых важных вопросов. Во всех учетных записях есть возможность разместить частные таблицы, где пользователь будет единственным человеком, которому будет разрешено считывать данные. Кроме безопасности чтения и записи, CartoDB предлагает услуги резервного копирования данных на каждом уровне хостинга. Это означает, что данные будут сохранены каждый раз, когда пользователь входит в систему.

Используя технологию CDN, карты, создаваемые с CartoDB, распределены по сети по всему миру. Это гарантирует максимальную скорость и надежность при обработке больших и сложных запросов.

На основе анализа существующих программных продуктов для работы с пользовательскими геоданными можно сделать вывод, что ни один из предложенных продуктов не удовлетворяет всем требованиям для использования разработчиками С#. Поддержка выборок данных из БД реализована далеко не у всех, а если и реализована, то функциональность клиентской части приложения оставляет желать лучшего. Фильтрация данных на основе координат центра веб-карты реализована только у веб-сервисов CartoDB, однако данные хранятся в этом случае на стороне веб-сервиса. Ни о какой интеграции в существующие продукты речи идти не может. Из обзора данных средств были вынесены следующие особенности, которыми должно обладать современное и удобное средство для управления записями:

1. Интеграция в существующие программные продукты, использующие карту, как источник информации.
2. Поддержка выборок и фильтрации данных на стороне сервера.
3. Расширяемый функционал клиентской части модуля.

Для дипломной работы были выбраны картографические сервисы OSM, так как OSM – это свободный веб-картографический проект, API которого дает разнообразный инструмент для создания собственных карт. При этом не стоит беспокоится о выполнении множества лицензионных соглашений и требований.

**2** СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Реализация программного модуля в виде библиотеки позволяет разработчикам использовать API модуля для работы с базами данных. Предполагается, что в качестве хранилища пространственных данных пользователя будут использоваться MS SQL Server для хранения и управления базой данных и ORM-фреймворком EntityFramework, поставляемых компанией Microsoft. Однако это не означает, что другие решения не поддерживаются. Модуль основан на работе с IQueriable<T> [5] интерфейсом библиотеки классов .NET, который возвращает выборку из базы данных и используется как контейнер для любого типа объектов. Поэтому база данных и ORM-фреймворк могут быть сторонними.

Также следует отметить то, что разработанный модуль является достаточно гибким с точки зрения расширения функционала. А применение фильтров позволяет конечным пользователям веб-приложения, основанного на данном модуле, пользоваться только картой и при этом получать лишь то количество данных, которое необходимо отобразить на веб-карте именно в каждый момент времени.

Выбранная модель архитектуры библиотеки предоставляет широкий инструментарий для написания веб-приложений, целью которых является взаимодействие пользовательского интерфейса (веб-браузер) со всеми остальными компонентами, которые требуют визуального отображения, посредством привязки элементов управления к соответствующим действиям.

При выборе языка программирования, на котором будет реализован программный модуль, внимание было обращено на такие факторы, как удобство работы, целесообразность использования для решения поставленной задачи, наличие стандарта и документации, поддержка операционной системы Windows и системы управления реляционными базами данных (СУРБД) MS SQL Server.

Следует отметить наличие большого количества библиотек, написанных для C#, среди которых можно выбрать наиболее подходящую для реализации задач по интеграции пользовательских данных в картографические сервисы.

Для обеспечения гибкой структуры программы, весь разрабатываемый проект разбивается на отдельные блоки. При таком подходе допускается модификация любого из выбранных модулей без значительного изменения остальных.

В результате системного проектирования, программный модуль был разбит на следующие блоки:

1. Блок инфраструктуры программного модуля.
2. Блок API.
3. Блок конфигурации.
4. Блок фильтров.
5. Блок представления данных.
6. Блок геокодирования.
7. Блок взаимодействия с геосервисами.
8. Блок репозитория БД.

Структурная схема, отображающая данные блоки и связи между ними представлена на чертеже ГУИР.400201.115 С1.

**2.1** Блок инфраструктуры программного модуля.

Данный блок предоставляет базовый функционал программного модуля. А именно: базовые классы для работы с простыми объектами, взаимодействие с удаленными ресурсами посредством сети Интернет, контейнер для хранения объектов на протяжение всей работы веб-приложения, к которому подключен данный модуль. Инфраструктура модуля построена таким образом, что каждый из блоков взаимодействует с другим через блок инфраструктуры за исключением блока API, который взаимодействует только с модулем конфигурации.

**2.2** Блок API

Данный блок позволяет использовать программный модуль как в уже существующих, так и в новых проектах. В блоке API реализованы методы для работы со всеми блоками модуля с областью видимости public. При подключение программного модуля в веб-приложение разработчику доступны лишь методы данного блока. Реализовано это было для невозможности изменения базового поведения основных элементов программного модуля.

**2.3** Блок конфигурации

Задачей данного блока служит конфигурирование всех блоков модулей, создание и управление фильтрами для выборок данных с условиями из БД. Разработчик через блок API взаимодействует с данннм блоком и настраивает все остальные части модуля посредством блока инфраструктуры.

Блок конфигурации является единственной точкой доступа для настройки всех блоков модуля.

Данный блок представлен различными атрибутами и вспомогательными классами. Атрибуты обеспечивают эффективный способ связывания метаданных или декларативной информации с кодом. Это дополнительная информация о классе, свойстве или методе, может быть полезна для других объектов системы.

**2.4** Блок фильтров

Данный блок отвечает за фильтрацию данных на стороне сервера. Таким образом на запрос карты со стороны блока представления данных, который в свою очередь получил запрос от блока взаимодействия с геосервисами (веб-браузер), происходит выборка данных из хранилища (база данных), передача этой выборки в блок фильтров, отсеивание данных по определенным правилам (реализованным фильтрам), обратное кодирование данных через блок представления данных и загрузка результата

Стоит отметить, что данные загружаются асинхронно на каждый запрос от блока взаимодействия с геосервисами о том, что параметры окружения изменились.

Архитектура блока позволяет разработчикам легко и гибко применять и реализовывать свои фильтры, наследоваться от уже реализованных, изменять их поведение.

**2.5** Блок представления данных

Задачей блока представления данных является преобразование информации, исходящей с серверной стороны приложения, направляемую на сторону клиента (веб-браузер). Форматы и инструменты широко представлены в данном блоке. Разработчик может воспользоваться реализованными в модуле конвертером данных, основанный на работе с форматом данных JSON (JSON — текстовый формат обмена данными, основанный на JavaScript и обычно используемый именно с этим языком. Как и многие другие текстовые форматы, JSON легко читается людьми), а также есть возможность подключить свой собственный инструмент конвертации данных.

**2.6** Блок геокодирования

Геокодирование – это процесс, преобразующий описание местоположения (например, координаты, адрес или название места) в местоположение на поверхности Земли, выраженное в данном модуле в виде GPS-координат (см. рисунок 2.1). Геокодирование позволяет анализировать пользовательские данные и преобразовывать некоторую часть из них в пространственные данные. С помощью блока геокодирования разработчики имеют инструменты для реализации систем, которые могут быстро находить различные виды местоположений пользователей или иных объектов. Посредством геокодирования возможно нахождение местоположения разных типов, включая достопримечательности или названия из географического справочника, такие как горы, мосты и магазины; координаты на основе долготы и широты, GPS-координаты, адреса в различных стилях и форматах, включая пересечения дорог, номера домов с названиями улиц и почтовыми кодами.

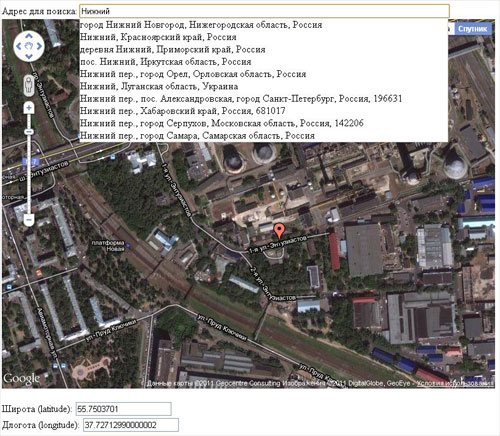


Рисунок 2.1 – Пример процесса геокодирования адреса

в сервисе Google Maps

В результате геокодирования получаются географические объекты с атрибутами, которые используются для выведения информации на веб-карту или пространственного анализа.

**2.7** Блок взаимодействия с геосервисами

Геосерис, или картографический сервис, - это веб-сервис, содержащий картографические данные, набор приложений, предоставляемых сервисом, спутниковые снимки, аэрофотоснимки. Обычно с сервисом интегрирован бизнес-справочник и карта автомобильных дорог, с поиском маршрутов по некоторым странам. Пример того, как выглядит картографический сервис в браузере, изображен на рисунке 2.2.

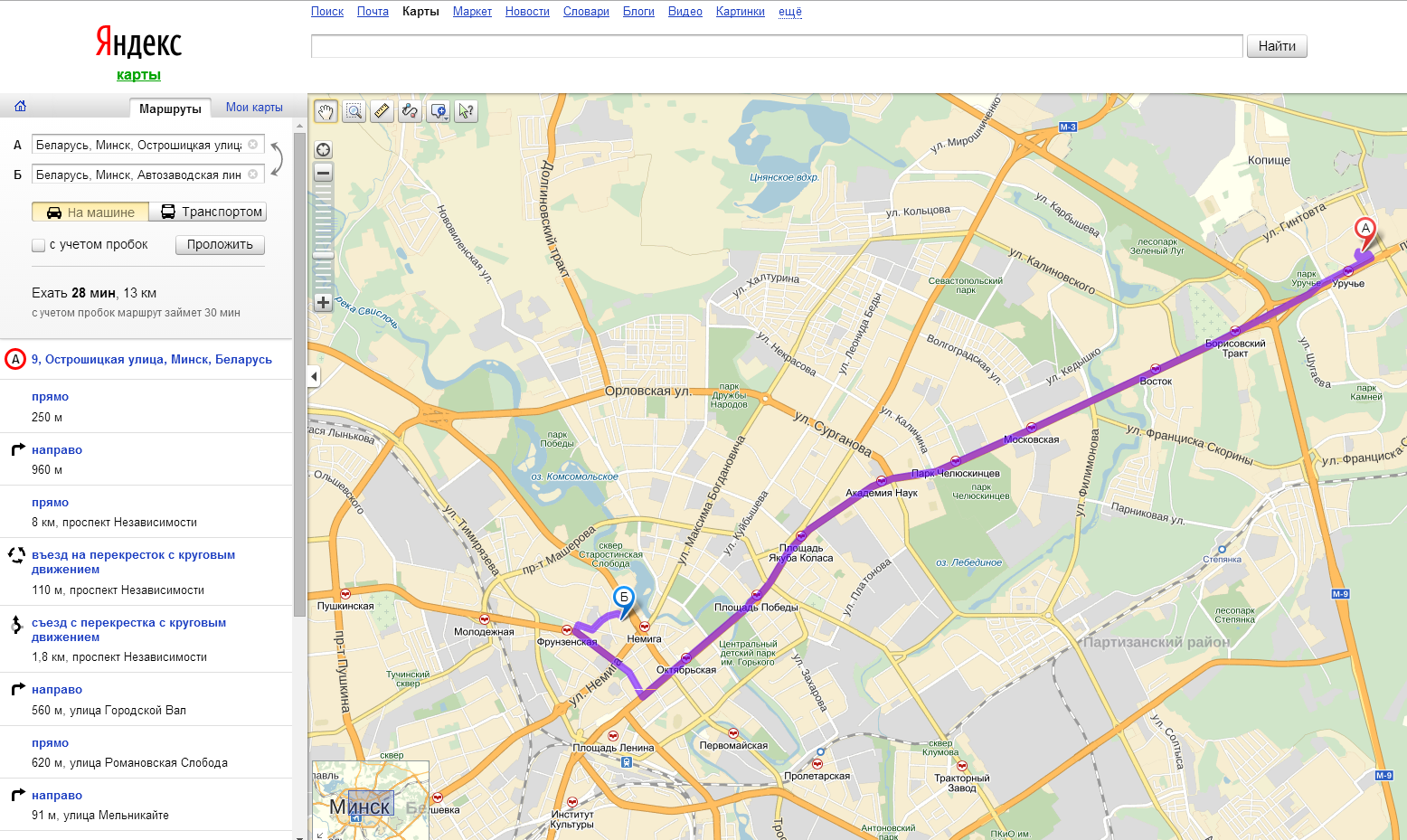


Рисунок 2.2 – Пример картографического сервиса Яндекс.Карты

Данный блок отвечает за прямое взаимодействие с API геосервисов таких, как OSM, Google Maps, Bing Maps, Яндекс.Карты и других. Разработчик, использующий разработанный модуль, может выбрать источник карт и картографических данных любого из современных провайдеров карт. Работа блока основана на использовании JS-библиотеки LeafletJS.

**2.8** Блок репозитория БД

Программный модуль может быть интегрирован как в уже существующее, так и в разрабатываемой веб-приложение. В базе данных приложения могут храниться записи и данные в любом формате. Данный блок позволяет производить выборки в таблицах в базе данных, транслировать фильтры, созданные в блоке фильтров в команды выборок для сервера базы данных.

Выборки проходят через модуль фильтров, там самым отсеивая те данные, которые не попадают под условия фильтров, а значит, уменьшается время на обработку запроса и выборку результата из базы данных.

**3** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В проектируемой системе главным архитектурным решением стало создание статических публичных (static public) методов для реализации поточных интерфейсов в API блоке, интерфейсов для реализации общего функционала, использование потоковых интерфейсов, абстрактных классов, множество различных классов-утилит, классов c расширяющими функционал методами (extensions methods), а также моделей, которые должны в полной мере использоваться во время разработки систем и программ по работе с геоданными.

Данный программный модуль можно охарактеризовать как middleware (связующее программное обеспечение), означающий слой программного обеспечения для обеспечения взаимодействия между различными приложениями, работающими с различными данными в БД, и браузером клиента.

Рассмотрим структурные блоки системы как реализацию той функциональности, которая описывалась выше.

**3.1** Блок инфраструктуры программного модуля

В программном модуле было реализованы несколько шаблонов проектирования, таких как: интерфейс, одиночка и декоратор. Шаблон проектирования или паттерн — повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста. Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться. На наивысшем уровне существуют архитектурные шаблоны, они охватывают собой архитектуру всей программной системы.

В сравнении с полностью самостоятельным проектированием, шаблоны обладают рядом преимуществ. Основная польза от использования шаблонов состоит в снижении сложности разработки за счёт готовых абстракций для решения целого класса проблем. Таким образом, за счёт шаблонов производится унификация деталей решений: модулей, элементов проекта, — снижается количество ошибок. Применение шаблонов концептуально равносильно использованию готовых библиотек кода. Правильно сформулированный шаблон проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова. Набор шаблонов помогает разработчику выбрать возможный, наиболее подходящий вариант проектирования. Сообразное использование паттернов проектирования дает разработчику ряд неоспоримых преимуществ.

Модель системы, построенная в терминах паттернов проектирования, фактически является структурированным выделением тех элементов и связей, которые значимы при решении поставленной задачи. Помимо этого, модель, построенная с использованием паттернов проектирования, более проста и наглядна в изучении, чем стандартная модель. Тем не менее, несмотря на простоту и наглядность, она позволяет глубоко и всесторонне проработать архитектуру разрабатываемой системы с использованием специального языка. Применение паттернов проектирования повышает устойчивость системы к изменению требований и упрощает неизбежную последующую доработку системы. Кроме того, трудно переоценить роль использования паттернов при интеграции информационных систем организации. Также следует упомянуть, что совокупность паттернов проектирования, по сути, представляет собой единый словарь проектирования, который, будучи унифицированным средством, незаменим для общения разработчиков друг другом.

Таблица 3.1 – Описание шаблонов проектирования

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Оригинальное название | Описание | Тип |
| 1 | 2 | 3 | 4 |
| Интерфейс | Interface | Общий метод для структурирования компьютерных программ для того, чтобы их было проще понять. | Общий |
| Репозиторий | Repository | Класс, который работает с источником данных | Общий |

Продолжение таблицы 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Декоратор | Wrapper/Decorator | Класс, расширяющий функциональность другого класса без использования наследования. | Структурный |

Таблица 3.2 - Свойства интерфейса IGeoEntity

|  |  |
| --- | --- |
| Свойства | Описание |
| double Latitude | Свойство типа double, которое хранит значение широты - угла между местным направлением зенита и плоскостью экватора. Так как широта отсчитывается от 0° до 90° в обе стороны от экватора, а также географическую широту точек, лежащих в северном полушарии, (северную широту) принято считать положительной, широту точек в южном полушарии — отрицательной, данное свойство может принимать значения от -90 до 90. |
| double Longitude | Свойство типа double, которое хранит значение долготы - угол между плоскостью меридиана, проходящего через данную точку, и плоскостью начального нулевого меридиана, от которого ведётся отсчёт долготы. Долготы от 0° до 180° к востоку от нулевого меридиана называют восточными, к западу — западными. Восточные долготы принято считать положительными, западные — отрицательными, соответственно, данное свойство может принимать диапазон значений от -180 до 180. |

В модуле были использованы шаблоны следующих семейств: общие, порождающие и структурные. Общая информация об использованных шаблонах представлена в таблице 3.1.

Отправной точкой модуля будет выступать статический класс FluentAPI. Он предоставляет доступ к публичным методам, так называемому API модуля, для работы со всеми блоками и инструментами.

В качестве базового объекта по работе с геоданными выступает интерфейс IGeoEntity, свойства которого описаны в таблице 3.2.

Любой класс, в котором есть широта и долгота в том или ином виде, должны реализовать данный интерфейс. Методы блока API, принимают IQueriable<T> объекты, где T – объекты классов, реализовавшие интерфейс IGeoEntity. Это позволяет выделить среди множества полей и свойств объекта только те, что прямым образом указывают на пространственную информацию.

* 1. Блок конфигурации

Данный блок представлен различными атрибутами и вспомогательными классами. Атрибуты обеспечивают эффективный способ связывания метаданных или декларативной информации с кодом. Это дополнительная информация о классе, свойстве или методе, которая на работу класса, свойства или метода не влияет, однако может быть полезна для других объектов системы.

* + 1. GeoAttribute

Данный класс наследуется от системного класса Attribute для возможности указания, какое свойство или поле класса содержит в себе пространственные значения. Поле GeoType \_propName хранит в себе значение перечисления GeoType: GeoType.Lat или GeoType.Lng.

* + 1. JsonProperty

Аналогично классу GeoAttribute был реализован атрибут JsonProperty для указания конвертеру форматов JsonConverter название свойств для создание JSON объектов.

**3.3** Блок API

Пользователь начинает свою работу с программным модулем со статического публичного класса FluentAPI. В нем содержится ряд статических методов, для работы с IQueriable<T> интерфейсом. Этот интерфейс предоставляет функциональные возможности расчета запросов к конкретному источнику данных, для которого не указан тип данных [5].

Интерфейсы взаимодействия реализованы как потоковые интерфейсы – способ реализации, в разработке программного обеспечения, объектно-ориентированного API, нацеленный на повышение читабельности исходного кода программы. Преимущества потокового интерфейса в том, что упрощается множественный вызов методов одного объекта. Обычно это реализуется использованием цепочки методов, передающих контекст вызова следующему звену. Обычно, этот контекст: определен с помощью значения, возвращаемого методом, или наследуется (в качестве нового контекста используется предыдущий), или прекращается возвращением ничего не значащего значения (void). Такой стиль косвенно полезен повышением наглядности и интуитивности кода.

**3.3.1** FluentAPI

Каждый потоковый интерфейс должен содержать конструктор интерфейса IQueriable<T>. В данном блоке конструктор создается вне программного модуля, а экземпляр объекта приходит в цепочки методов в качестве параметра. Данный статический класс является отправной точкой в выборке и фильтрации данных на серверной стороне. Методы класса, представленные в таблице 3.3, принимают IQueriable<T> объект, производят манипуляции с данными в методах-цепочках и метод-исполнитель позволяет выйти из цепочки методов и получить результат.

Таблица 3.3 – Методы класса Configurator

|  |  |
| --- | --- |
| Название и параметры | Описание |
| 1 | 2 |
| internal IQueryable<T> QueryGeoData<T>(this IQueryable<T> query) where T : IGeoEntity | Принимает объект выборки данных и возвращает его. Используется внутри модуля |

Продолжение таблицы 3.3

|  |  |
| --- | --- |
| 1 | 2 |
| internal IQueryable<T> QueryGeoData<T>(this IQueryable<T> query) where T : IGeoEntity | Принимает объект выборки данных и возвращает его. Используется внутри модуля |
| public static IEnumerable<T> ListGeoData<T>(this IQueryable<T> query) where T : IGeoEntity | Принимает объект выборки данных и возвращает его в виде списка |
| public static IEnumerable<T> GetGeoData<T>(this IQueryable<T> query) where T : IGeoEntity | Принимает объект выборки данных и возвращает его в виде интерфейса IEnumerable<T> |
| public static IQueryable<T> FilterBy<T>(this IQueryable<T> query, IFilter filter, IFilterArgs args) where T : IGeoEntity | Принимает объект выборки данных, фильтрует данные по определенному фильтру с параметрами, которые приходят вместе с данными и возвращает его в объекта выборки |
| public static IQueryable<T> FilterBy<T>(this IEnumerable<T> query, IFilter filter, IFilterArgs args) where T : IGeoEntity | Принимает объект интерфейса IEnumerable данных, фильтрует данные по определенному фильтру с параметрами, которые приходят вместе с данными и возвращает его в объекта выборки |
| public static string Append(this string current, string key, string value) | Дописывает в передаваемую строку фрагмент другой строки |
| public static object GetAttributeValue(Type objectType, string propertyName, Type attributeType, string attributePropertyName) | Возвращает значение атрибута по определенному типу и свойству |

**3.4** Блок фильтров

Блок фильтров служит для отсеивания той части информации, которая не попадает под определенные условия или ограничения, накладываемые на получаемые данные от модуля репозитория.

В блоке содержится интерфейс IFilter, абстарктный класс Filter, который частично реализует интерфейс IFilter, а также интерфейс IFilterArgs для создания на его основе классов по передаче данных для обработки последних в соответствующем фильтре. Преимущество данной реализации в том, что разработчик легко может создавать свои собственные фильтры, основываться на уже реализованном BoundsFilter, который фильтрует данные по условию, находятся ли точки, загружаемые из БД в области экрана или же они за областью видимости. Свойства и методы интерфейса IFilter представлены в таблице 3.4.

Таблица 3.4 – Интерфейс IFilter

|  |  |
| --- | --- |
| Название и параметры | Описание |
| string Name | Название фильтра для последующей идентификации |
| IQueryable<T> ApplyFilter<T>(IQueryable<T> query, IFilterArgs args) where T : IGeoEntity | Метод применения фильтра, который будет вызываться в цепочке методов блока API |

Чтобы у разработчика была возможность переопределять и реализовывать свои методы ApplyFilter, класс Filter сделан абстрактным c виртуальными и абстрактными методами. Разработчику необходимо создать свой класс и унаследоваться от абстрактного класса Filter, реализовать метод ApplyFilter и по необходимости переопределить свойство Name. Члены класса Filter представлены в таблице 3.5.

Интерфейс IFilterArgs содержит словарь объектов Dictionary<string, object> Params для хранения объектов для последующей передачи данных в экземпляр класса, унаследованного от абстрактного класса Filter. Примером реализации интерфейса IFilterArgs служит класс BoundsFilter, поля и методы которого представлены в таблице 3.6.

Таблица 3.5 – Члены абстрактного класса Filter

|  |  |
| --- | --- |
| Название и параметры | Описание |
| virtual string Name | Название фильтра для последующей идентификации. Без переопределения название будет соответствовать названию класса. |
| public abstract IQueryable<T> ApplyFilter<T>(IQueryable<T> query, IFilterArgs args) where T: IGeoEntity | Метод применения фильтра, который должен быть реализован в классе-наследнике и который будет вызываться в цепочке методов выборки данных из блока репозитория БД |

Таблица 3.6 – Класс BoundsFilter

|  |  |
| --- | --- |
| Название и параметры | Описание |
| Поля: | |
| private LatLng northEast | Поле, хранящее координаты северо-восточной точки на карте |
| private LatLng southWest | Поле, хранящее координаты юго-западной точки на карте |
| Свойства: | |
| public Dictionary<string, object> Params | Словарь объектов с ключом типа string, для идентификации объектов в словаре. |

**3.5** Блок репозитория БД

Для удобства управления данными реализован репозиторий БД. Это набор интерфейсов, классов и их методов для удобной и быстрой работы с источниками данных, таких, например, как базы данных.

Были выделены основные методы по работе с объектами:

1. Добавление
2. Удаление
3. Редактирование
4. Получение по ID
5. Получение списока всех объектов

Получившаяся архитектура блока репозитория представлена на рисунке 3.1.

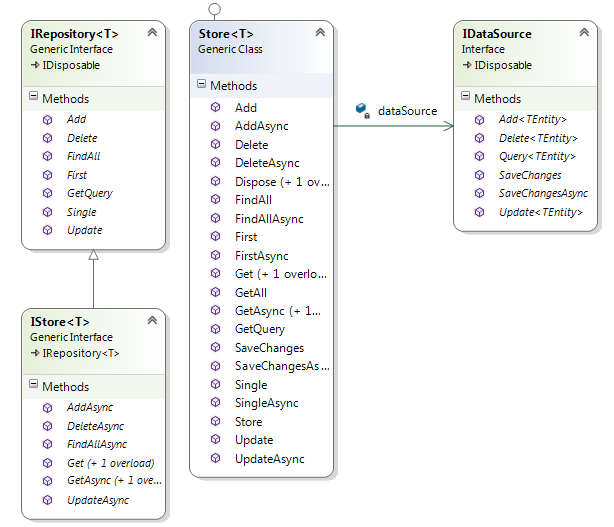


Рисунок 3.1 – Диаграмма классов блока репозитория БД

**3.5.1** IDataSource

Классы, реализовавшие данный интерфейс, могут предоставлять данные для репозиториев, фильтров и других модулей. Стоит отметить, что данные классы универсальные, а значит смогут работать со всеми объектами, реализовавшими интерфейс IEntity, который хранит поле Id типа Guid.

Таблица 3.7 – Методы интерфейса IDataSource

|  |  |
| --- | --- |
| Название и параметры | Описание |
| 1 | 2 |
| IQueryable<TEntity> Query<TEntity>() where TEntity : class, IEntity; | Получение объекта выборки |

Продолжение таблицы 3.7

|  |  |
| --- | --- |
| 1 | 2 |
| void Add<TEntity>(TEntity value) where TEntity : class, IEntity; | Добавление объекта |
| void Update<TEntity>(TEntity value) where TEntity : class, IEntity; | Изменение объекта |
| void Delete<TEntity>(TEntity value) where TEntity : class, IEntity; | Удаление объекта |
| void SaveChanges(); | Сохранение изменений в источнике данных |
| Task SaveChangesAsync(); | Асинхронное сохранение изменений в источнике данных |

**3.5.2** IRepository

Репозиторий или хранилище - место, где хранятся и поддерживаются объекты, доступные через исчтоник данных.

Таблица 3.8 – Методы интерфейса IRepository

|  |  |
| --- | --- |
| Название и параметры | Описание |
| IQueryable<T> GetQuery() | Получение объекта выборки |
| T First(Expression<Func<T, bool>> predicate) | Получение первого объекта выборки |
| T Single(Expression<Func<T, bool>> predicate) | Получение единственного выборки |
| IEnumerable<T> FindAll(Expression<Func<T, bool>> predicate = null) | Получение IEnumberable выборки |
| void Add(T entity) | Добавление объекта |
| void Update(T entity) | Изменение объекта |
| void Delete(T entity); | Удаление объекта |

**3.5.3** Store

В блоке репозиториев широко применяется паттерн проектирования *интерфейс*. Это означает, что инициализация классов, реализующих определенные интерфейсы, будет происходит также по интерфейсу. Пример этому – класс Store<T>, который позволяет управлять источником данных. Класс реализует интерфейс IStore<T>. В данном случае не важно, что за тип базы данных хранит данные пользователей, или какой ORM-фрейворк используется для получения этих данных. Члены класса Store представлены в таблице 3.9.

Таблица 3.9 – Члены класса Store

|  |  |
| --- | --- |
| Название и параметры | Описание |
| 1 | 2 |
| Поля: | |
| private IDataSource dataSource; | Интерфейс, который будет указывать на класс, реализующий IDataSource, как источник данных |
| Методы: | |
| public void Dispose() | Словарь объектов с ключем типа string, для идентификации объектов в словаре. |
| IQueryable<T> GetQuery() | Реализация интерфейса IRepository<T> |
| T First(Expression<Func<T, bool>> predicate) | Реализация интерфейса IRepository<T> |
| T Single(Expression<Func<T, bool>> predicate) | Реализация интерфейса IRepository<T> |
| IEnumerable<T> FindAll(Expression<Func<T, bool>> predicate = null) | Реализация интерфейса IRepository<T> |
| void Add(T entity) | Реализация интерфейса IRepository<T> |
| void Update(T entity) | Реализация интерфейса IRepository<T> |
| void Delete(T entity); | Реализация интерфейса IRepository<T> |
| public virtual Task<T> FirstAsync(Expression<Func<T, bool>> predicate) | Реализация интерфейса IRepository<T> |
| public virtual Task<T> SingleAsync(Expression<Func<T, bool>> predicate) | Реализация интерфейса IStore<T> |

Продолжение таблицы 3.9

|  |  |
| --- | --- |
| 1 | 2 |
| public virtual Task<IEnumerable<T>> FindAllAsync(Expression<Func<T, bool>> predicate = null) | Реализация интерфейса IStore <T> |
| public virtual IEnumerable<T> GetAll() | Реализация интерфейса IStore <T> |
| public virtual Task UpdateAsync(T entity) | Реализация интерфейса IStore <T> |
| public Task AddAsync(T entity) | Реализация интерфейса IStore <T> |
| public virtual Task UpdateAsync(T entity) | Реализация интерфейса IRepository<T> |
| public Task DeleteAsync(T entity) | Реализация интерфейса IRepository<T> |
| public virtual T Get(Guid id) | Реализация интерфейса IStore <T> |
| public virtual Task<T> GetAsync(Guid id) | Реализация интерфейса IStore <T> |
| public virtual T[] Get(params Guid[] ids) | Реализация интерфейса IStore <T> |
| public virtual Task<T[]> GetAsync(params Guid[] ids) | Реализация интерфейса IRepository<T> |
| public void SaveChanges() | Реализация интерфейса IRepository<T> |
| public Task SaveChangesAsync() | Реализация интерфейса IRepository<T> |

**3.6** Блок геокодирования

Сервис, который дает возможность преобразовать описание местоположения в виде адреса или название места в местоположение на поверхности Земли, выраженное в данном блоке в виде LatLng класса. В качестве поставщика данных и преобразований был выбран сервис компании Google.

**3.6.1** GeocodingService

Класс GeocodingService является начальной точкой во взаимодействии приложения разработчика и сервиса по геокодированию. Класс содержит в себе статическое неизменяемой поле ApiUrl, которое хранит сетевой адрес сервиса, а также метод GetResponse, который принимает в качестве параметров GeocodingRequest объект с данными, необходимыми для геокодирования (см. таблицу 3.10), и возвращает объект класса GeocodingResponse.

Для удобства все необходимые поля были помечены атрибутом JsonProperty для передачи объектов между серверной частью и клиентской используя JSON формат и блок представления данных. Тем самым при передаче JSON объект будет иметь определенный набор полей и значений.

**3.6.2** GeocodingRequest

Таблица 3.10 – Класс GeocodingRequest

|  |  |
| --- | --- |
| Название и параметры | Описание |
| Свойства: | |
| public string Address | Адрес, который необходимо геокодировать |
| public string LatitudeLongitude | Текстовое значение широты и долготы, которое будет использоваться для нахождения ближайшего объекта с удобочитаемым адресом |
| public string Bounds | Границы карты выраженные в виде северо-восточного и юго-западного угла |
| public string Region | Двухбуквенный код региона |
| public string Language | Язык, в котором необходимо вернуть ответ |
| public string Sensor | Это свойство указывает, отправлен запрос с устройства с поддержкой локационного сенсора, или же с устройства без него |
| Методы: | |
| internal Uri ToUri() | Преобразует объект GeocodingRequest в строку для последующей отправки |

Единственный параметр, который передается в метод GetResponse класса сервиса, это объект запроса GeocodingRequest, свойства и методы которого указаны в таблице 3.10.

**3.6.3** GeocodingResponse

Класс является компонентом класса GeocodingService. Содержит в себе данные, которые пришли в качестве ответа от сервиса геокодирования. Свойства класса представлены в таблице 3.11.

Ответ приходит после формирования запроса и вызова метода GetResponse класса GeocodingService. Ответ по умолчанию приходит в JSON формате. Пример ответа на запрос с параметрами latlng=53.714224,27.561452 представлен на рисунке 3.3.

Ответ с помощью класса JsonSerializer преобразуется в объект класса GeocodingResponse с соответствующими полями.

Таким образом, взаимодействие между программным модулем и сервисом геокодирования можно представить в виде связи клиент-сервер посредством REST протокола над HTTP, которая показано на рисунке 3.2.

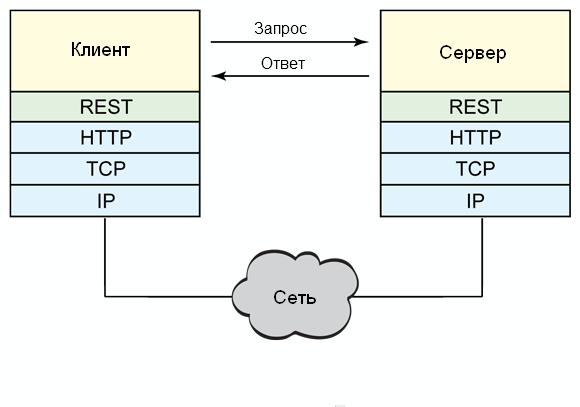


Рисунок 3.2 - REST сервис процесса геокодирования

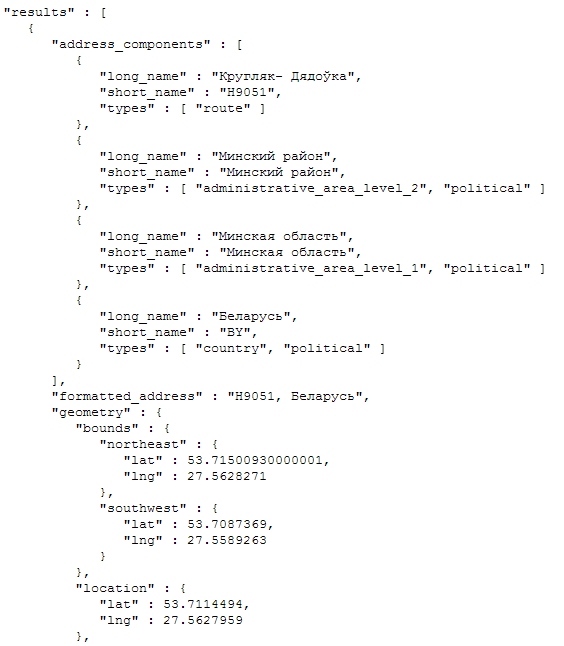


Рисунок 3.3 – Ответ от сервиса Google Geocoding

Таблица 3.11 – Класс GeocodingResponse

|  |  |
| --- | --- |
| Название и параметры | Описание |
| Свойства: | |
| public ServiceResponseStatus Status | Свойство, хранящее перечисление ServiceResponseStatus |
| public GeocodingResult[] Results | Массива объектов класса GeocodingResult |

**3.6.4** ServiceResponseStatus

Является компонентом класса GeocodingResponse. Содержит в себе значения статуса ответа от геосервиса. Возможные значения перечисления указаны в таблице 3.12.

Таблица 3.12 – Класс ServiceResponseStatus

|  |  |
| --- | --- |
| Название и параметры | Описание |
| Unknown | Неизвестная результат |
| Ok | Ошибок при геокодирование не произошло |
| InvalidRequest | Неправильно сформированный запрос |
| ZeroResults | Нет результатов по текущему запросу |
| OverQueryLimit | Лимит запросов истек |
| RequestDenied | Запрос был отклонен |

**3.6.5** GeocodingResult

Объект содержит в себе детали декодированного запроса. Члены класса представлены в таблице 3.13.

Таблица 3.13 – Класс GeocodingResult

|  |  |
| --- | --- |
| Название и параметры | Описание |
| public AddressType[] Types | Типы адресов, которые вернулись вместе с ответом от сервиса геокодирования |
| public string FormattedAddress | Удобочитаемый адрес, который пришел вместе с ответом |
| public AddressComponent[] Components | Компоненты адреса, такие как номер дома, |
| public LatLng Geometry | Пространственные данные пришедшие от сервиса |

**3.6.6** Http и HttpGetResponse

Статический класс Http и класс HttpGetResponse используются для создания веб-запроса к сервису, получение ответа, выборки потока ответа и десериализации ответа и преобразование его в объект. Поля и методы класс HttpGetResponse представлены в таблице 3.14.

Таблица 3.14 – Класс HttpGetResponse

|  |  |
| --- | --- |
| Название и параметры | Описание |
| Поля: | |
| private Uri requestUri | Поле для хранения текущего рабочего веб-адреса |
| Методы: | |
| public string AsString() | Чтение ответа от сервиса и и представление ответа в виде строки |
| public T As<T>() where T : class | Преобразование строкового ответа в объект типа T |

**3.7** Блок представления данных

Блок содержит несколько классов для работы с геоданными, их преобразованием в строку, JSON формат, обратное преобразование из строки и JSON формата. В качестве формата передачи данных был выбран именно формат JSON по нескольким причинам. Кратко перечислим их.

Во-первых, JSON (англ. JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript и обычно используемый именно с этим языком. Как и многие другие текстовые форматы, JSON легко читается людьми.

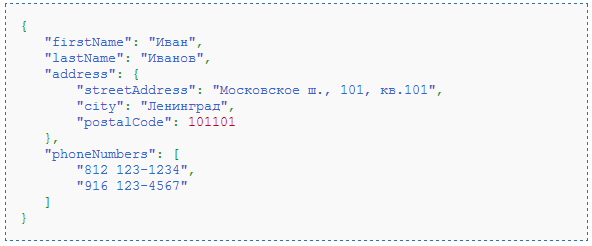


Рисунок 3.4 – JSON структура

Для сравнения, на рисунке 3.4 и рисунке 3.5 можно сравнить формат JSON и XML, количество символов для передачи на сторону клиента, удобочитаемость для человека. По всем позициям лидирует JSON. Он более прост и JavaScript понимает его без каких-либо преобразований в качестве объекта для всевозможных действий.

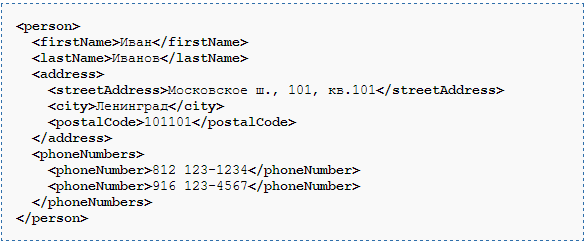


Рисунок 3.5 – XML структура

Во-вторых, несмотря на происхождение от JavaScript, формат считается независимым от языка и может использоваться практически с любым языком программирования. Для С# существует готовые решения для создания и обработки данных в формате JSON, в том числе сериализация и десериализация данных с помощью класса JavascriptSerializer. Однако программный модуль выступает в роли связующего программного обеспечения, поэтому процесс сериализации и десериализации был немного изменен, а именно: на свойства Latitude и Longitude распространяется атрибут JsonProperty c параметрами «lat» и «lng» соотвественно, чтобы объекты, содержащие данные поля, однозначно трактовали пространственные данные в виде поля lat и lng.

И в-третьих, JSON-текст представляет собой (в закодированном виде) одну из двух структур: набор пар «ключ:значение». В различных языках это реализовано как объект, запись, структура, словарь, хэш-таблица, список с ключом или ассоциативный массив. Ключом может быть только строка, значением — любая форма. Или же как упорядоченный набор значений. Во многих языках это реализовано как массив, вектор, список или последовательность.

Это универсальные структуры данных: как правило, любой современный язык программирования поддерживает их в той или иной форме. Они легли в основу JSON, так как он используется для обмена данными между различными языками программирования. В качестве значений в JSON используются структуры объекта, массива, строки и значения.

Объект — это неупорядоченное множество пар ключ:значение, заключённое в фигурные скобки «{ }». Ключ описывается строкой, между ним и значением стоит символ «:». Пары ключ-значение отделяются друг от друга запятыми. Массив (одномерный) — это упорядоченное множество значений. Массив заключается в квадратные скобки «[ ]». Значения разделяются запятыми. Значение может быть строкой в двойных кавычках, числом, объектом, массивом, одним из литералов: true, false или null. Таким образом, структуры могут быть вложены друг в друга.

Строка — это упорядоченное множество из нуля или более символов юникода, заключенное в двойные кавычки. Символы могут быть указаны с использованием escape-последовательностей, начинающихся с обратной косой черты «\».

**3.7.1** LatLng

В качестве модели хранения непосредственно самих пространственных данных выступает класс LatLng, свойства и методы которого указаны в таблице 3.15.

Таблица 3.15 - Класс LatLng

|  |  |
| --- | --- |
| Название и параметры | Описание |
| 1 | 2 |
| Свойства: | |
| public double Latitude | Широта, хранящаяся в double формате |
| public double Longitude | Долгота, хранящаяся в double формате |
| Методы: | |
| public static bool TryParse(string s, out LatLng result) | Метод, который вызывается объектом класса LatLngConverter для попытки распознать значения свойств Latitude и Longitude. |

Продолжение таблицы 3.15

|  |  |
| --- | --- |
| 1 | 2 |
| internal virtual JsonObjectBuilder ToJsonObjectBuilder() | Преобразование объекта данного класса в объект класса JsonObjectBuilder |
| public string Render() | «Отрисовка» объекта в виде строки |

**3.7.2**  LatLngConverter

Для передачи LatLng объекта на клиент был написан TypeConverter для преобразования свойств класса LatLng в строки, для последующей сериализации и десериализации в/из формат JSON. Методы LatLngConverter класса представлены в таблице 3.16.

Таблица 3.16 - Класс LatLngConverter

|  |  |
| --- | --- |
| Название и параметры | Описание |
| public override bool CanConvertFrom (ITypeDescriptorContext context, Type sourceType) | Метод определяет, можно ли получить верные данные у переданного контекста по определенному типу |
| public override object ConvertFrom (ITypeDescriptorContext context,  CultureInfo culture, object value) | Реализация этого метода вызывает метод TryParse класса LatLng |

**3.7.3**  JsonConverter

Непосредственно самой конвертацией в формат JSON занимается класс JsonConverter. Методы данного класса представлены в таблице 3.17

Таблица 3.17 Члены класса JsonConverter

|  |  |
| --- | --- |
| Название и параметры | Описание |
| 1 | 2 |
| Поля: | |
| private readonly JsonObjectBuilder builder | Объект сериализации и десериализации данных |
| Методы: | |

Продолжение таблицы 3.17

|  |  |
| --- | --- |
| 1 | 2 |
| public virtual string Serialize(object obj) | Чтение ответа от сервиса и и представление ответа в виде строки |
| public virtual T Deserialize<T>(string json) | Преобразование строкового ответа в объект. |

Объект данного класса может быть использован в приложении как для прямого преобразования (из объекта в JSON), так и для обратного (из JSON в объект заданного типа).

**3.7.4** CollectionContext

Данный класс был реализован, как декоратор данных, переданных с серверной части в клиентскую. Представляет собой обертку или контейнер для хранения массива объектов, подлежащих преобразованию для дальнейшей работе с ними на клиентской части.Члены класса представлены в таблице 3.18.

Класс имеет атрибуты DataContract и DataMember, которые указывают, что данный класс может подлежать преобразованию JSON преобразователями. Свойства, не помеченные атрибутом DataMember, не могут быть преобразованы в JSON формат. Данный класс является универсальным, потому может содержать объекты любого типа, являясь контейнером. Также стоит отметить, что данный класс предоставляет свойство TotalCount типа int для хранения в нем общего количества объектов. Это удобно, например, тогда, когда нужно реализовать фильтр, показав при этом общее количество объектов в системе.

Таблица 3.18 – Члены класса CollectionContext

|  |  |
| --- | --- |
| Название и параметры | Описание |
| 1 | 2 |
| Поля: | |
| private T[] items | Поле для хранения массива объектов |
| Свойства: | |
| public int TotalCount | Общее число объектов. Может не совпадать с количеством объектов в массиве items |

Продолжение таблицы 3.18

|  |  |
| --- | --- |
| 1 | 2 |
| public bool IsGeneric | Указывает, является ли тип, который хранится в массиве items, общим |
| public IDictionary<string, string> Context | Контекст, в который могут быть добавлены любые данные с определенными ключами |
| public IEnumerable<T> Items | Акссесор и модификатор доступа к массиву объектов, хранящимся в данном объекте-контейнере |

**3.7.5** GeoActionResult

Имея ActionResult класс, можно легко контролировать процесс получение результата, который попадет на клиентскую часть. Для этих целей и был реализован GeoActionResult, унаследованный от системного ActionResult и использующий JsonConverter для сериализации данных и отправки данных в блок взаимодействия с геосервисами. Члены класса представлены в таблице 3.19.

Таблица 3.19 – Члены класса GeoActionResult

|  |  |
| --- | --- |
| Название и параметры | Описание |
| Поля: | |
| private readonly JsonConverter jsonConverter | Объект класса JsonConverter для преобразования списка объектов в JSON формат |
| Методы: | |
| public override void ExecuteResult(ControllerContext context) | Переопределенный метод ExecuteResult у класса JsonResult |

**3.7.6** JsonObjectBuilder

Иногда стандартного JSON-конвертера не хватает, потому что информацию в объекте нужно сперва обработать перед тем, как преобразовывать в JSON. Для того и был реализован класс JsonObjectBuilder, методы и поля которого представлены в таблице 3.20.

Таблица 3.20 – Члены класса JsonObjectBuilder

|  |  |
| --- | --- |
| Название и параметры | Описание |
| Поля: | |
| protected Dictionary<string, object> \_ObjectValues | Инкапсулированное поле для хранения ключей и значений необходимых для построения JSON объекта |
| internal Dictionary<string, object> Values | Аксессор и модификатор для поля \_ObjectValues |
| Методы: | |
| protected JsonObjectBuilder AppendValue(string key, object value) | Дописывает объект к уже существующему объекту в JSON формате |
| public JsonObjectBuilder Append(string key, string value) | Дописывает строку к уже существующему объекту в JSON формате. |
| public JsonObjectBuilder Append(string key, int value) | Дописывает число к уже существующему объекту в JSON формате |
| public JsonObjectBuilder Append(string key, int? value) | Дописывает число, которое может быть не проинициализированным, к уже существующему объекту в JSON формате |
| public JsonObjectBuilder Append(string key, double value) | Дописывает число в формате double к уже существующему объекту в JSON формате |
| public JsonObjectBuilder Append(string key, double? value) | Дописывает число в формате double, которое может быть не проинициализированным, к уже существующему объекту в JSON формате |
| public JsonObjectBuilder Append(string key, bool value) | Дописывает булево значение к уже существующему объекту в JSON формате |
| public JsonObjectBuilder Append(string key, bool? value) | Дописывает булево значение, которое может быть не проинициализированным, к уже существующему объекту в JSON формате |
| public JsonObjectBuilder Append(object obj) | Дописывает объект к уже существующему объекту в JSON формате |
| public string Render() | Отрисовка объекта в виде строки |

**3.8** Блок взаимодействия с геосервисами

Данный блок содержит клиентскую логику по отображению карты, кластеризацию маркеров (точек на карте). Блок представлен скриптом, написанным на JavaScript.

Работа блока основана на использовании JS-библиотеки LeafletJS. В данном блоке была реализована функция *F(),* которая содержит в себе всю логику по работе с объектами на карте.

Таблица 3.21 – Члены функции F()

|  |  |
| --- | --- |
| Название и параметры | Описание |
| Поля: | |
| var map | Объект L.Map библиотеки LeafletJS |
| var mapquest | Объект инициализации карты |
| var objects | Объекты, которые подлежат кластеризации |
| Функции: | |
| initRegistration() | Регистрация события создания новой записи с помощью карты |
| cancelRegistration() | Отмена регистрации события создания новой записи с помощью карты |
| getObjects() | Получение объектов, которых необходимо нанести на карту. |
| insertObject() | Добавление объекта с помощью AJAX вызова на сервер |
| removeObject() | Удаление объекта с помощью AJAX вызова |
| onMapClick(e) | Обработчик события нажатия левой кнопки мыши на карту в момент нанесения на неё нового объекта |

В функции getObjects выполняется AJAX запрос на сервер по определенному адресу, который возвращает список объектов, необходимых отобразить на карте. После получения списка объектов, последние добавляются в слой кластеризации и на основе библиотеки Leaflet.markerClusterer проходит этап кластеризации. После выполнения алгоритма, точки в цикле добавляются на карту.

**4** РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

В программном модуле использовалась клиентская часть, как инструмент обработки пользовательской информации на карте. Рассмотрим подробней алгоритм кластеризации, а также как это работает в связке с серверной частью и библиотекой Leaflet.

Основные требования, которые предъявлялись к кластеризатору:

* Возможность подключения без глубокой подготовки данных на сервере, а именно, используя только уже имеющиеся географические координаты объектов.
* Простой протокол подготовки и передачи данных (JSON via REST)
* Быстрый клиентский кластеризатор. Кластер — несколько объектов, расположенных рядом. При клике мышкой по кластеру, он «раскрывается», то есть увеличивает масштаб пока объекты будут на расстоянии, большем чем расстояние кластеризации.
* Отображение до 10000 точек с использованием кластеризатора.
* Шаблонизация на клиенте двух инфоокон — группового кластера и самого объекта.
* Возможность использования неограниченного количества типов меток (иконок) для разных типов объектов.
* Возможность использования нескольких размеров иконок для разного диапазона зума.

Кластеризатор, в отношении географических карт, означает визуальную группировку нанесенных меток на карте. К примеру, в задаче необходимо разместить на карте несколько сотен маркеров. Подключается карта, система получает координаты маркеров и размещает на карте через отдельный слой (далее - тайл карты). Маркеры размещены, однако если их достаточно много, несколько тысяч в пределах одного города, и пользователь отдалит карту на несколько делений, на карте будет отображаться сильное нагромождение маркеров в одной точке. Во избежание этого и используется кластеризация маркеров.

Непосредственно сам алгоритм кластера не зависит от Leaflet, впрочем, как и от других провайдеров карт. Разработано достаточно много вариантов алгоритма кластеризатора меток на карте, но все они в различной степени уже интегрированный в библиотеку работы с картами.

Непосредственно выполнение добавления объектов на карту, их кластеризацию и «отрисовку» решено было перенести на сторону клиента., так как выполнение кластеризации на сервере дополнительно нагружало последний, что негативно сказалось на запросах пользователей в целом (см. рисунок 4.1). Сегодня компьютерные машины достаточно быстродейственные, и даже кластеризовать 10 000 маркеров для современного веб-браузера уже не составит проблемы.



Рисунок 4.1 – Кластеризация на серверной и клиентской стороне

Вся область карты разбивается на прямоугольники фиксированного размера, все метки, попадающие в заданный прямоугольник при данном масштабе заменяются одной меткой с указанием количества исходных данных (объектов выбранных серверной частью). В начале задается стили для отображения окна с картой, затем подключаем необходимые javascript-библиотеки: API Google Maps v3, jQuery и MarkerClusterer плагин для библиотеки Leaflet.

В функции initialize задается начальные параметры веб-карты: координаты центра, уровень масштаба и тип карты. После этого используя функцию getObjects, загружаются исходные данные о маркерах, передаваемые с сервера по адресу, определенному в функции getObjects в массив markers. После этого, кластеризатору через метод new MarkerClusterer, передается указатель на карту и массив меток.

Также, был вынесен объявление объекта карты map в глобальные переменные. У библиотеки MarkerClusterer есть дополнительные параметры, которые позволяют управлять порядком кластеризации. Основные из них: gridSize — размер ячейки сетки в пикселях, maxZoom — максимальный масштаб для кластеризации, styles — позволяет задавать стиль отображения кластеров.

Серверная часть состоит из двух запросов: запрос на данные карты и запрос об объекте. В первый запрос передается ссылка с параметрами для выборки меток. Например, cityobjects/?bounds=56,5875;27,5789. То есть выбираем только те объекты, которые помещаются с данными параметрами карты в её область, тем самым используя на сервере фильтр

В ответ отдается скрипт с JSON-параметрами карты (объекты в контейнере, их общее количество и любая другая информация, которая необходима разработчику для отображения на карте). Данные точки содержат: id объекта, геокоординаты, тип иконки и строку всплывающей подсказки. На 10 000 точек в несжатом состоянии размер ответа — 1,2Мб, но при сжатии gzip — всего 350 кб. Часто повторяющиеся запросы хорошо кэшируются сервером.

При наведении на одиночный маркер всплывает инфоокно с данными об объекте, которые заполняют инфоокно во время «отрисовки» на карте. Ajax-запрос содержит id объекта. В ответ отдается json. Названия полей json должно совпадать с тэгами в шаблоне инфоокна, последние подменяются полученным содержанием. Пример показан на рисунке 4.2.

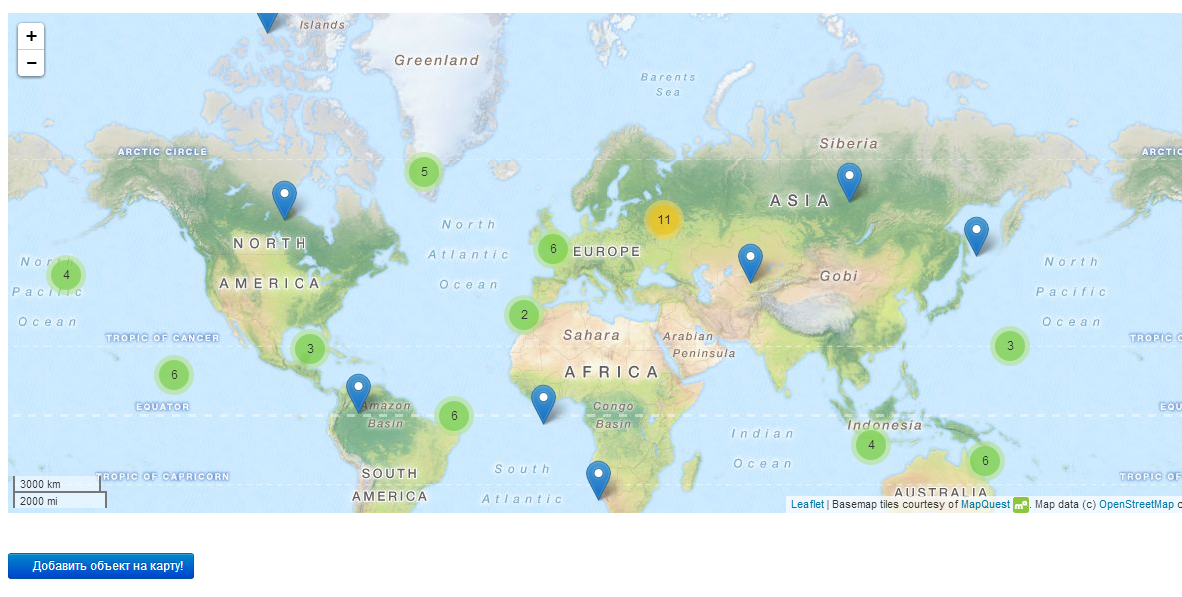


Рисунок 4.2 – Пример применения кластеризатора

Полученные данные обрабатываются следующим образом:

1. Вычисляются глобальные пиксельные координаты на нулевом масштабе для каждой точки.

2. Исходя из глобальных координат вычисляется id тайла зуме, и точки помещаются в массив с соответствующим id тайла. В дальнейшем, на любом масштабе можно легко вычислить Id необходимых тайлов и выбрать все точки для них. Скорость выборки точек — для 10,000 точек не более 100/350мс. Время парсинга 10000 точек — 100/650 мс.

В результате был выбран кластеризатор MarkerClusterer, как плагин к Leaflet. Во-первых, инфраструктура работы как с кластеризатором, так и с библиотекой Leaflet одинакова, так что разработчик будет уже знаком с синтаксисом библиотеки и кластеризатора изначально. А во-вторых, в нем способ кластеризации асинхронно-гибридный: при запросе данных для тайла происходит следующее: тайл разбивается на 4 подтайла (сетка) и каждый подтайл кластеризуется по расстоянию, но сам кластер располагается не в середине ячейки, а в центре области «свернутых» точек, полученные точки и кластеры собираются обратно в тайл, и снова кластеризуются. На этом этапе точек уже немного, полученные точки «прореживаются» с учетом граничащих тайлов (если они уже были кластеризованы). То есть если в граничащем тайле на границе есть кластер или маркер, перекрывающий маркер на другом тайте, то эти маркеры объединяются в кластер. Это позволяет избежать размещения иконок кластеров вдоль границы сетки кластеризации. Для вывода иконок используются спрайты. Количество типов иконок не ограничено. В примерах используется 5 типов иконок для одиночных маркеров и иконок кластера.

Таблица 4.1 – Тайминг карты с 10 000 точками и кластеризацией на машине 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Браузер | Тайминг | | | | Объектов в DOM |
| Выборка точек | Кластеризация | Нанесение точек | Общее время |
| Intel Core Quad Q9550 2.8 GHz ОЗУ 4Gb Windows-7 64bit | | | | | |
| Google Chrome 35.0 | 80мс | 73мс | 19мс | 172мс | 126 |
| Firefox 29.01 | 30мс | 84мс | 46мс | 160мс | 126 |
| Internet Explorer 11 | 117 мс | 225мс | 240мс | 582мс | 126 |

По клику на маркер происходит следующее: если маркер является кластером — кластер масштабируется до «раскрытия».

Для вывода маркеров использовался стандартный механизм библиотеки Leaflet. Была написана модель, в которой хранятся и обрабатываются данные, кластеризуются тайлы и отрисовываются канвасы, а также несколько контроллеров, которые подгружают необходимый API и выводят маркеры.

В таблицах 4.1, 4.2 и 4.3 представлены тайминги выборки, кластеризации и «отрисовки» на карту на различных компьютерных машинах.

Таблица 4.2 – Тайминг карты с 10 000 точками и кластеризацией на машине 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Браузер | Тайминг | | | | Объектов в DOM |
| Выборка точек | Кластеризация | Нанесение точек | Общее время |
| Intel Core i5 3.2 GHz ОЗУ 8Gb Windows-8 64bit | | | | | |
| Google Chrome 35.0 | 63мс | 48мс | 25мс | 111мс | 238 |
| Firefox 29.01 | 45мс | 82мс | 36мс | 163мс | 238 |
| Internet Explorer 11 | 389мс | 293мс | 267мс | 949мс | 238 |

Таблица 4.3 – Тайминг карты с 10 000 точками и кластеризацией на машине 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Браузер | Тайминг | | | | Объектов в DOM |
| Выборка точек | Кластеризация | Нанесение точек | Общее время |
| Intel Core i7 3.2 GHz ОЗУ 4Gb Windows-8 64bit | | | | | |
| Google Chrome 35.0 | 63мс | 49мс | 23мс | 135мс | 238 |
| Firefox 29.01 | 45мс | 90мс | 37мс | 172мс | 238 |
| Internet Explorer 11 | 221мс | 123мс | 297мс | 641мс | 238 |

**5** ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Испытания проводятся с целью проверки и подтверждения соответствия фактической программной реализации модуля требованиям, и оценки готовности программного модуля к передаче в постоянную эксплуатацию.

Тестирование – деятельность, выполняемая для оценки и улучшения качества программного обеспечения. Эта деятельность, в общем случае, базируется на обнаружении дефектов и проблем в программных системах.

Тестирование программных систем состоит из динамической верификации поведения программ на конечном (ограниченном) наборе тестов, выбранных соответствующим образом из обычно выполняемых действий прикладной области и обеспечивающих проверку соответствия ожидаемому поведению системы.

Основным подходом при тестировании программного обеспечения является тестирование «черного ящика». При данном подходе неизвестно внутреннее устройство программы. Взаимодействие с программой следующее: введение данных, нажатие кнопок.

Так как обычно полное тестирование невозможно или труднореализуемо, задача тестирования подобрать данные и последовательность действий, на которых с наибольшей вероятностью могут проявиться ошибки тестируемой программы.

При разработке программного обеспечения проводятся большое количество одинаковых тестов при проведении приемочного и регрессионного тестирования. В связи с этим, автоматизация тестирования может позволить сэкономить множество человеко-часов. При проведении автоматизированного тестирования задача – разработка набора тестов.

Так как модуль реализован в виде .NET библиотки, то требования к техническим и программным средствам вычислительной техники аналогичны требованиям фреймворка .NET версии 4.5.

**5.1** Механизмы платформы .NET

При разработке .NET-приложений можно воспользоваться платформой для создания легковесного и гибкого инструмента, который будет запускать разработанные тесты. В данном случае имеет смысл быстрое создание набора автоматизированных тестов, которые будут заданы прямо в коде. Для быстрой модификации и добавления тестов необходимо создать и использовать сборку с различными методами, которые будут выполнять с пользовательским интерфейсом программы необходимые тестировщику действия.

При автоматизированном тестировании пользовательского интерфейса Windows-приложений возможны два решения. Первое из них основано на механизме отражения (Reflection).

Механизм отражения позволяет получать объекты, которые описывают сборки, модули и типы. Отражение можно использовать для динамического создания экземпляра типа, привязки типа к существующему объекту, а также получения типа из существующего объекта и вызова его методов или доступа к его полям и свойствам.

При использовании механизма отражения необходимо загрузить сборку, получить тип формы, создать экземпляр и запустить приложение с данной формой в новом потоке. Так метод GetAttributeValue класса FluentAPI получает значение атрибута определенного объекта. Так как в программном модуле атрибуты – это достаточно частый механизм связывания данных и метаданных, то разработчики могут добавлять метаданные в свой код благодаря атрибутам. С точки зрения разработчика оба типа имеют одинаковый синтаксис. Атрибуты в коде являются по сути сообщениями компилятору для создания метаданных. В CIL метаданные (такие как например, модификаторы наследования, модификаторы области видимости), а также почти все, что не является кодом операции или потоками, также помечаются как атрибуты.

**5.2** Внешнее тестирование

Внешнее тестирование проводится для проверки соответствия программного модуля предъявляемым требованиям. Тесты выполняются по готовым сценариям, каждый из которых содержит последовательность шагов и описание ожидаемого состояния системы при их выполнении. Сценарий является успешным, если все наблюдаемые результаты соответствуют описанным.

Для программного модуля были разработано веб-приложение, работающее на разработанном программном модуле, и следующие сценарии:

1. Создание нового объекта на карте. Целью сценария является проверка возможности создание нового объекта. Необходимые действия и их результат приведены в таблице 5.1.

Таблица 5.1 – Сценарий создания объекта в демо приложении

|  |  |  |
| --- | --- | --- |
| № | Действие | Результат |
| 1 | Нажать на ссылку «Создать новый объект» | Система отображает всплываюзее окно с кнопкой начать |
| 2 | Нажать на кнопку «Создать» для непосредственного нанесения маркера на веб-карту. | После нажатия системы считывает, в каком месте указал пользователь точку на карте, сохраняет позицию, выраженную в широте и долготе |
| 3 | Нажать на кнопку «Отправить» | Система собирает данные из формы, которую заполняет пользователь. Если некоторые требуемые данные будут непроинициализированы, система отобразит валидационное сообщение о том, что не все поля заполнены. Если все данные были заполнены, системы выполнит AJAX POST запрос на сервер для создания нового объекта в базе данных. Если на серверной стороне произойдут какие-то ошибки, на веб-карте отобразится всплывающее окно о том, что произошли ошибки в системе |

1. Изменение уже существующего объекта на карте. Целью сценария является проверка возможности изменение уже созданного объекта. Необходимые действия и их результат приведены в таблице 5.2.

Таблица 5.2 – Сценарий изменения объекта в демо приложении

|  |  |  |
| --- | --- | --- |
| № | Действие | Результат |
| 1 | 2 | 3 |
| 1 | Навести курсор на маркер | Навести курсор на маркер |
| 2 | Изменить поля формы | - |

Продолжение таблицы 5.2

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 3 | Нажать на кнопку «Отправить» | Система собирает данные из формы, которую заполняет пользователь. Если некоторые требуемые данные будут непроинициализированы, система отобразит валидационное сообщение о том, что не все поля заполнены. Если все данные были заполнены, системы выполнит AJAX POST запрос на сервер для создания нового объекта в базе данных. Если на серверной стороне произойдут какие-то ошибки, на веб-карте отобразится всплывающее окно о том, что произошли ошибки в системе |

1. Удаление уже существующего объекта на карте. Целью сценария является проверка возможности удаления уже существующего объекта. Необходимые действия и их результат приведены в таблице 5.3.

Таблица 5.3 – Сценарий удаления объекта в демо приложении

|  |  |  |
| --- | --- | --- |
| № | Действие | Результат |
| 1 | Навести курсор на маркер | Система должна отобразить всплывающее окно с кнопкой «Удалить объект» |
| 2 | Нажать на кнопку удаления | Система должна сделать AJAX DELETE запрос на сервер с целью удаления объекта |
| 3 | Подтвердить необходимость удаления | Карта перерисовывается, чтобы добавить новый маркер в группу кластеров. |

**6** РУКОВОДСТВО ПРОГРАММИСТА

**6.1** Подключение модуля к проекту

Для работы приложения необходимо соблюдение следующих требований к программному обеспечению:

1. Необходима установка .NET-фреймворка версии 4.5.
2. Установленный экземпляр Visual Studio 2012 и права администратора.
3. Установленная и подключенная библиотека по работе с картами Leaflet версии не ниже 0.4.
4. Веб-браузер с включенным JavaScript.

Подключаемые файлы программного модуля представляют собой библиотечный файл формата dll и файл сценариев Jscript формата js. В Visual Studio необходимо создать новый веб-проект. Добавить необходимые для работы сборки и модули.

Необходимо скопировать Frontmaps.dll в папку с локальными сборками, модулями и инструментами, а затем подключить к проекту как новую сборку из локальной папки как это показано на рисунке 6.1 и 6.2.

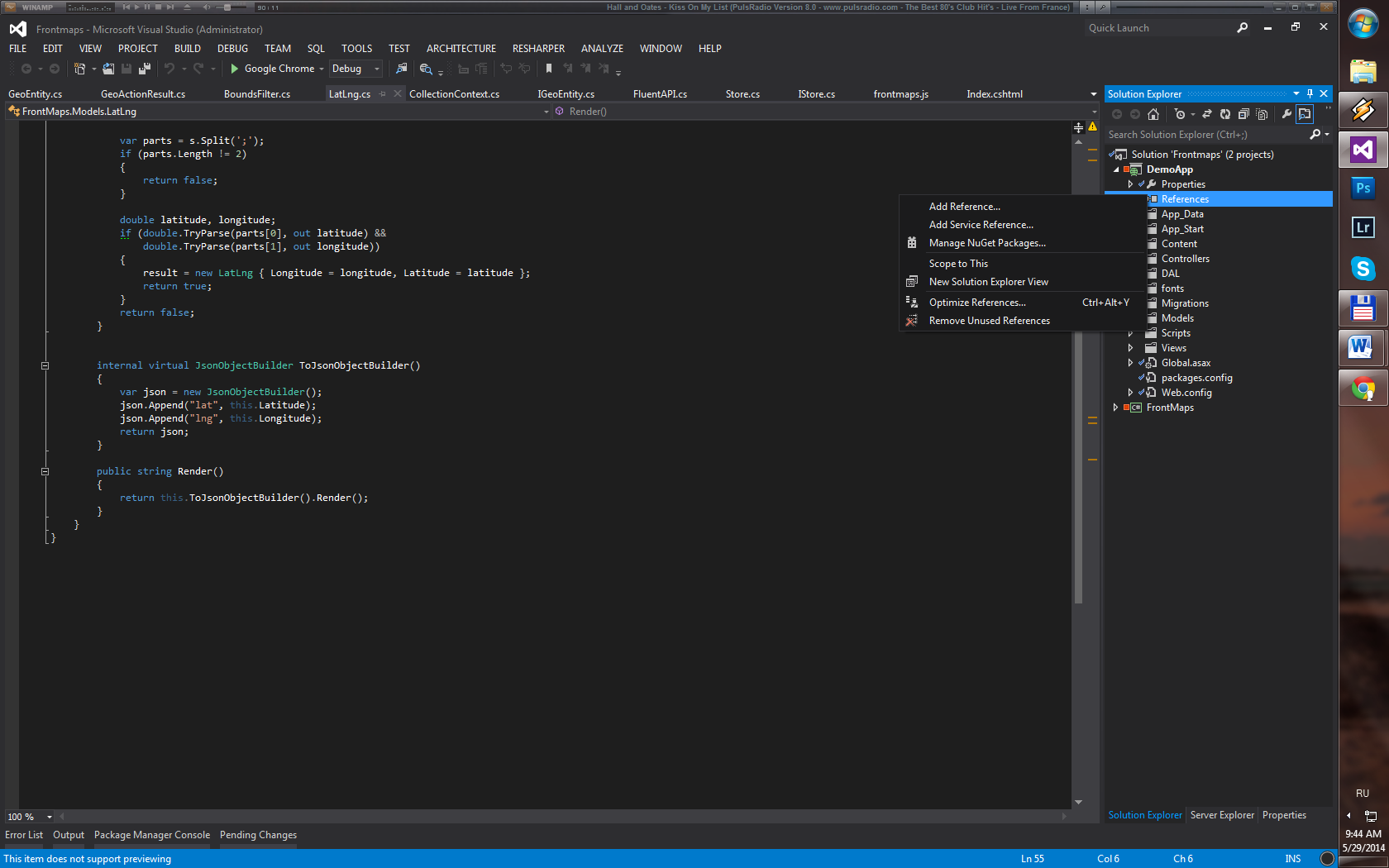


Рисунок 6.1 – Подключение модуля к проекту

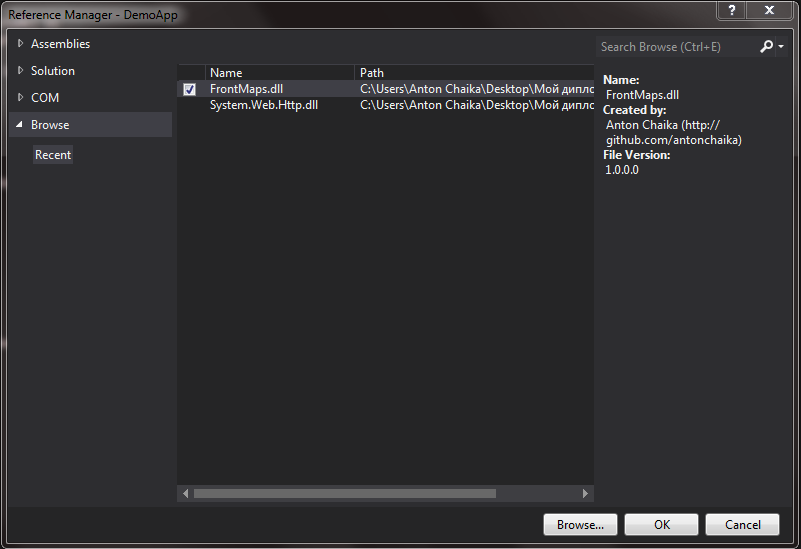


Рисунок 6.2 – Выбор модуля для подключения

**6.2** Использование репозитория IStore<T>

Данный интерфейс уже реализован в модуле в виде класса Store<T>. Однако, разработчик вправе реализовать собственный класс, реализовать интерфейс IDataSource в качестве источника данных и IQueriable<T> для динамических выборок. Система может не ограничиваться одним классом, реализовавшим IDataSource. Можно динамически переключаться между репозиториями и, соответственно, между источниками данных. Данный интерфейс позволяет реализовать методы для таких методов:

1. Добавление.
2. Удаление.
3. Редактирование.
4. Получение по ID.
5. Получение списока всех объектов.

Используя конструкции вида:

Object a = new Object ();

IStore<Object> store = new Store< Object >();

/\*…\*/

store.AddItem(a);

store.ChangeItem(a);

store.DeleteItem(a.ID);

store.SaveChanges();

При этом с данным классом нет необходимости каждый раз выбирать нужную коллекцию объектов из списка и можно избежать написания нескольких классов и использовать всего один как единый репозиторий и доступ к данным. Тем самым код становится намного понятней и тесты могут проводиться независимо от остальных модулей в системе.

**6.3** Работа с сервисом геокодирования

Работа с модулем геокодирования сводится к созданию запроса к сервису геокодирования, передаче этому запросу необходимых параметров и получение ответа от сервиса в виде объекта класса GeocodingResponse:

var request = new GeocodingRequest();

request.Address = ‘sample address’;

request.Sensor = false;

var response = new GeocodingService().GetResponse(request);

if (response.Status == ServiceResponseStatus.Ok)

{

/\*ok status\*/

}

**6.4** Работа с библиотекой Leaflet

Leaflet содержит удобный синтаксис по работе с маркерами, инфоокнами и другими инструментами для отображения пространственных данных. Несколько улучшений API, позволяющих писать более простой и краткий код (в стиле jQuery):

L.marker([53.5, -27.5])

.addTo(map)

.bindPopup('Hello world!')

.openPopup();

Методы теперь принимают объекты LatLng, LatLngBounds, Point и Bounds в форме простых массивов — теперь нет необходимости создавать их явно:

map.panTo([50, 30]); // то же самое, что:

map.panTo(new L.LatLng(50, 30));

Такие методы класса Map, как addLayer, addControl и openPopup, получили свои аналоги с другой стороны:

marker.addTo(map); // аналогично map.addLayer(marker)

control.addTo(map); // map.addControl(control)

popup.openOn(map); // map.openPopup(popup)

Учитывая то, что в Leaflet все методы, которые не возвращают какое-то явное значение, возвращают сам объект (this), это позволяет более удобно вызывать методы «по цепочке».

Каждый класс в Leaflet имеет свой class factory (с таким же именем, но начинающийся с маленькой буквы), что позволяет создавать объекты без оператора new:

L.map('map').fitWorld(); // то же самое, что:

(new L.Map('map')).fitWorld();

Методы для работы с событиями on и off принимают несколько типов событий в одной строке (с пробелами между типами):

map.on('click dblclick moveend', doStuff);

Также можно передавать объекты с парами событий/слушателей:

marker.on({

click: onMarkerClick,

dragend: onMarkerDragEnd

});

Кроме этого, если ничего не передавать в метод off кроме названия события, объект отпишет всех слушателей, которые были подписаны на данное событие:

map.off('click');

Также удобно работать и с кластеризатором. Для того, чтобы проинициализировать кластеризатор, необходимо создать с помощью объекта L библиотек и Leaflet объект markerClusterGroup. Задать необходимые параметры и переопределить параметры и настройки по умолчанию. Пример такой инициализации представлен ниже.

var markers = L.markerClusterGroup({

maxClusterRadius: 120,

iconCreateFunction: function (cluster) {

var markers = cluster.getAllChildMarkers();

var n = 0;

for (var i = 0; i < markers.length; i++) {

n += markers[i].number;

}

return L.divIcon({ html: n, className: 'mycluster', iconSize: L.point(40, 40) });

},

//Отключение настроек по умолчанию

spiderfyOnMaxZoom: false, showCoverageOnHover: false, zoomToBoundsOnClick: false

});

**7** ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ ПРОГРАММНОГО МОДУЛЯ ДЛЯ ИНТЕГРАЦИИ ПОЛЬЗОВАТЕЛЬСКИХ ДАННЫХ С ПРИВЯЗКОЙ К GPS КООРДИНАТАМ В КАРТОГРАФИЧЕСКИЕ СЕРВИСЫ

**7.1** Характеристика программного продукта

Модуль для интеграции пользовательских данных в картографические сервисы адаптирован для работы, как с существующими данными, так и с новыми. Данный модуль позволяет эффективней использовать поисковые запросы, фильтрацию и добавление новых записей, привязанных к географическим координатам посредством добавления записи прямо на карте.

Разработка и внедрение данного модуля позволят:

− снизить трудоемкость работы по управлению записями базы данных через пользовательский интерфейс

− сократить время на поиск записи в БД

− сократить время на создание новой записи

Экономическая целесообразность инвестиций в разработку и использование программного продукта осуществляется на основе расчета и оценки следующих показателей [14]:

− чистая дисконтированная стоимость (*ЧДД*)

− срок окупаемости инвестиций (*ТОК*)

− рентабельность инвестиций (*Ри*)

В результате разработки данного программного модуля снизится трудоемкость выполнения задачи регистрации новых, фильтрации существующих объектов, управление данными с привязкой к географическим координатам, что и будет являться результатом от внедрения программного продукта.

**7.2** Расчет затрат и отпускной цены программного средства

1. Основная заработная плата исполнителей проекта определяется по формуле:

, (7.1)

где *n* – количество исполнителей, занятых разработкой ПС;

*Tчi* – часовая тарифная ставка *i*-го исполнителя (руб.);

*Фэi* – эффективный фонд рабочего времени *i*-го исполнителя (дней);

*ТЧ* – количество часов работы в день (ч);

*К* – коэффициент премирования (1,35).

В настоящий момент тарифная ставка 1-го разряда на предприятии составляет 1000000 руб.

Расчет основной заработной платы представлен в таблице 7.1.

Таблица 7.1 **–** Расчет основной заработной платы

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Исполнитель | Раз-ряд | Тарифный коэффи-циент | Месячная тарифная ставка, руб. | Часовая тарифная ставка, руб. | Плано-вый фонд рабочего времени, дн. | Заработная плата, руб. |
| Программист 1-ой категории | 14 | 3,25 | 3250000 | 19345 | 40 | 8357140 |
| Программист 2-й категории | 8 | 2,17 | 2170000 | 12920 | 50 | 6975000 |
| Основная заработная плата |  |  |  |  |  | 15332140 |

2. Дополнительная заработная плата исполнителей проекта определяется по формуле:

, (7.2)

где *Нд* – норматив дополнительной заработной платы (20%).

Дополнительная заработная плата составит:

руб. (7.3)

3. Отчисления в фонд социальной защиты населения и на обязательное страхование (*Зсз*) определяются в соответствии с действующими законодательными актами по формуле:

, (7.4)

где *Нсз* – норматив отчислений в фонд социальной защиты населения и на обязательное страхование (34+0,6%).

Отчисления составят:

руб. (7.5)

4. Расходы по статье «Машинное время» (*РМ*) включают оплату машинного времени, необходимого для разработки и отладки ПС, и определяются по формуле:

, (7.6)

где *Цм* – цена одного машино-часа;

*Тч* – количество часов работы в день;

*Ср* – длительность проекта.

Стоимость машино-часа на предприятии составляет 14000 руб. Разработка проекта займет 90 дней. Определим затраты по статье «Машинное время»:

руб. (7.7)

5. Затраты по статье «Накладные расходы» (*РН*), связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных (экспериментальных) производств, а также с расходами на общехозяйственные нужды, определяются по формуле:

, (7.8)

где *Нрн* – норматив накладных расходов (45%).

Накладные расходы составят:

руб. (7.9)

Общая сумма расходов по всем статьям сметы (*Сп*) на программное обеспечение рассчитывается по формуле:

(7.10)

и составит:

руб. (7.11)

Кроме того, организация-разработчик осуществляет затраты на сопровождение и адаптацию ПС (*РСА*), которые определяются по нормативу (*НРСА*):

, (7.12)

где *Нрса* – норматив расходов на сопровождение и адаптацию (20%).

руб. (7.13)

Общая сумма расходов на разработку (с затратами на сопровождение и адаптацию) как полная себестоимость ПС (*СП*) определяется по формуле:

(7.14)

Прибыль ПС рассчитывается по формуле:

, (7.15)

где *Ппс* – прибыль от реализации ПС заказчику (руб.);

*УР*  – уровень рентабельности ПС (20%);

*СП* – себестоимость ПС (руб.).

руб. (7.16)

Прогнозируемая отпускная цена ПС составит:

руб. (7.17)

**7.3** Расчет стоимостной оценки результата

Результатом (*Р*) в сфере использования программного продукта является прирост чистой прибыли и амортизационных отчислений.

**7.3.1** Расчет прироста чистой прибыли

Прирост прибыли за счет экономии расходов на заработную плату в результате снижения трудоемкости выполнения работ, выполняемых каталогизаторами.

Экономия затрат на заработную плату при использовании ПС в расчете на объем выполняемых работ определяется по формуле:

(7.18)

где *Nп* – плановый объем работ;

*tс* – трудоемкость выполнения работы до внедрения программного средства, нормо-час;

*tн* – трудоемкость выполнения работы после внедрения программного средства, нормо-час;

*Тс* – часовая тарифная ставка, соответствующая разряду выполняемых работ до внедрения программного продукта (10000 руб.);

*Тн* – часовая тарифная ставка, соответствующая разряду выполняемых работ после внедрения программного продукта (10000 руб.);

*Кпр* – коэффициент премий (1,35);

*Нд* – норматив дополнительной заработной платы (15%);

*Нно* – ставка отчислений от заработной платы, включаемых в себестоимость (34,6%).

До внедрения программного продукта трудоемкость создания записи на каталогизируемый документ составляла 1,9 человека-часа, после внедрения программы – 0,7 человека-часа. В год каталогизаторами создается около 1000 записей.

Экономия на заработной плате и начисления на заработную плату составит:

руб. (7.19)

Прирост чистой прибыли (*ΔПч*) определяется по формуле:

, (7.20)

где *Нп* – ставка налога на прибыль (18%).

Таким образом, прирост чистой прибыли составит:

руб. (7.21)

**7.3.2** Расчет прироста амортизационных отчислений

Расчет амортизационных отчислений осуществляется по формуле:

(7.22)

где *З* – затраты на разработку программы, руб.;

*НА* – норма амортизации программного продукта (17%).

Амортизационные отчисления составят:

руб. (7.23)

**7.4** Расчет показателей эффективности использования программного продукта

Для расчета показателей экономической эффективности использования программного средства необходимо полученные суммы результата (прироста чистой прибыли) и затрат (капитальных вложений) по годам привести к единому времени – расчетному году (за расчетный год принят 2014 год) путем умножения результатов и затрат за каждый год на коэффициент приведения (*αt*), который рассчитывается по формуле:

, (7.24)

где *Ен* – норматив приведения разновременных затрат и результатов (25%);

*tp* – расчетный год, *tp* = 1;

*t* – номер года, результаты и затраты которого приводятся к расчетному (2014 – 1, 2015 – 2, 2016 – 3, 2017 – 4).

Коэффициенты приведения для первых четырех лет эксплуатации ПС составят:

|  |  |
| --- | --- |
| (2014 год); |  |
| (2015 год); | (7.25) |
| (2016 год); |
| (2017 год); |  |

Результаты расчета показателей эффективности приведены в таблице 7.2.

Проект планируется внедрить в организации во второй половине 2014 года, поэтому в 2014 году организация может получить половину прибыли ( руб.).

Рассчитаем рентабельность инвестиций в разработку и внедрение программного продукта (*РИ*) по формуле:

, (7.26)

где *Пчср* – среднегодовая величина чистой прибыли за расчетный период, которая определяется по формуле:

, (7.27)

где *Пчt* – чистая прибыль, полученная в году t.

Чистая прибыль и рентабельность инвестиций составят:

руб. (7.28)

%. (7.29)

Таблица 7.2 − Расчет экономического эффекта от использования нового ПС

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Наименование показателей | Ед. изм | Усл.  обоз. | По годам производства | | | |
| 1-й | 2-й | 3-й | 4-й |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Результат |  |  |  |  |  |  |
| 1. Прирост чистой прибыли | руб | *∆Пч* | 10281151 | 20562303 | 20562303 | 20562303 |
| 2. Прирост амортизации-онных отчислений | руб | *∆А* | 11261437 | 11261437 | 11261437 | 11261437 |
| 2. Прирост результата | руб | *∆Рt* | 21542588 | 31823740 | 31823740 | 31823740 |
| 3.Коэффициент дисконтирова-ния | руб | *αt* | 1 | 0,8 | 0,64 | 0,512 |
| 4. Результат с учетом фактора времени | руб |  | 21542588 | 25458992 | 20367194 | 16293755 |
| Затраты (инвестиции) |  |  |  |  |  |  |
| 5. Инвестиции в разработку программного продукта | руб | *Иоб* | 66243750 |  |  |  |
| 6. Инвестиции с учетом фактора времени | руб |  | 66243750 |  |  |  |
| 7. Чистый дисконтированный доход по годам (п.4 - п.6) | руб | *ЧДДt* | -44701161 | 25458992 | 20367194 | 16293755 |

Продолжение таблицы 7.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8. ЧДД нарастающим итогом | руб | *ЧДД* | -44701161 | -19242169 | 1125025 | 17418780 |

В результате технико-экономического обоснования применения программного средства были получены следующие значения показателей их эффективности:

1. Чистый дисконтированный доход за четыре года работы программы составит 17418780 руб.

2. Затраты на разработку программного продукта окупятся на второй год его использования.

3. Рентабельность инвестиций составляет 27,2%.

Таким образом, применение программного продукта является эффективным и инвестиции в его разработку целесообразно осуществлять.

**8** ОХРАНА ТРУДА. ОБЕСПЕЧЕНИЕ ПОЖАРНОЙ БЕЗОПАСНОСТИ

НА ООО «ОМЕРТЕКС»

Целью дипломного проектирования была разработка программного модуля для интеграции пользовательских данных в картографические сервисы. Разработка выполнялась в интересах предприятия, поэтому рассмотрим пожарную безопасность на ООО «Омертекс».

Пожарная безопасность на предприятии обеспечивается системой предотвращения пожара и системой пожарной защиты. Помещение, в котором размещены ПЭВМ, по категориям пожарной опасности относится к категории «В» (твердые горючие и трудногорючие вещества и материалы (в том числе пыли и волокна), вещества и материалы, способные при взаимодействии с водой, кислородом воздуха или друг с другом взрываться и гореть, при условии, что помещения, в которых они имеются в наличии или обращаются, не относятся к категории А или Б). Обычно в нем находится большое количество возможных источников возгорания, как например: кабельные линии, используемые для питания ПЭВМ от сети переменного тока напряжением 220В, различные электронные устройства, которые при отказе систем охлаждения могут привести к короткому замыканию, оборудование мебелью из горючих материалов; носитель информации - бумага [15].

Ответственность за нарушение требований пожарной безопасности в организациях несут персонально их руководители. Ущерб, причиненный пожарами, подлежит возмещению в порядке, устанавливаемом законодательством [16].

ООО «Омертекс» находится в административном здании, работники которого проинструктированы о мерах пожарной безопасности в административных зданиях [17].

При проектировании зданий было предусмотрено безопасная эвакуация людей на случай возникновения пожара. При возникновении пожара люди должны покинуть здание в течение минимального времени, которое определяется кратчайшим расстоянием от места их нахождения до выхода наружу.

План эвакуации для сотрудников предприятия изображен на рисунке 8.1. Число эвакуационных выходов из зданий, помещений и с каждого этажа зданий равно двум. Эвакуационные выходы располагаются рассредоточено. При этом лифты и другие механические средства транспортирования людей при расчетах не учитывают, так как их использование запрещается в случае пожарной тревоги. Ширина участков путей эвакуации составляет 1 м, а дверей на путях эвакуации 0.8м. Ширина наружных дверей лестничных клеток должна быть не менее ширины марша лестницы, высота прохода на путях эвакуации - не менее 2 м. При проектировании зданий и сооружений для эвакуации людей предусмотрен следующий вид лестничных клеток и лестниц: незадымляемые лестничные клетки (сообщающиеся с наружной воздушной зоной или оборудованные техническими устройствами для подпора воздуха).

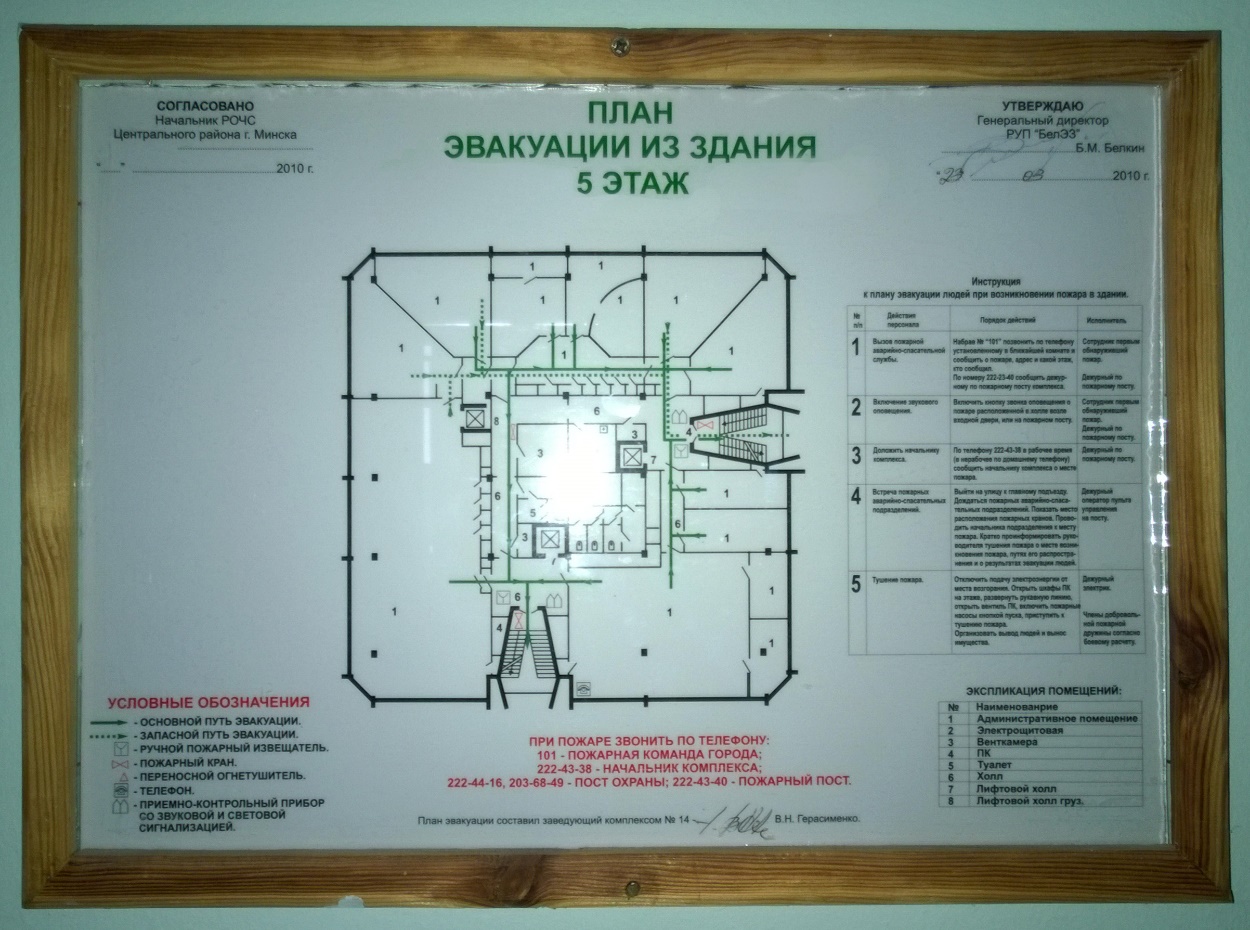


Рисунок 8.1 – План эвакуации в здании (5 этаж)

Пожарная сигнализация применяется для своевременного оповещения о времени и месте пожара и принятия мер по его ликвидации [18]. Системы пожарной сигнализации состоят из пожарных извещателей (датчиков), линий связи, приемной станции, откуда сигнал о пожаре может передаваться в помещения пожарных команд.

На предприятии установлены извещатели пожарные дымовые оптические точечные двухпроводные модели ИП 212-5МУ в общем количестве 22 штук (см. рисунок 8.2 и рисунок 8.3).



Рисунок 8.2 – ИП 212-5МУ



Рисунок 8.3 – ИП 212-5МУ

Сигналы с пожарных извещателей заведены на пульт управления дежурного по зданию. Индикация светодиодами на каждое помещение на каждый этаж.

Предприятие не оборудовано автоматической системой пожаротушения, поэтому во всех помещениях предусмотрено расположение первичных средств пожаротушения – огнетушителей.

На предприятии используются углекислотные огнетушители типа ОУ-2, ОУ-3, ОУ-5 и порошковые огнетушители типа ОП-4 в общем количестве 14 штук (см. рисунок 8.4 и рисунок 8.5).



Рисунок 8.4 – Огнетушитель ОУ-3



Рисунок 8.5 – Огнетушитель ОП-4

Углекислотные огнетушители состоят из баллона с углекислотой, запорно-пускового вентиля, сифонной трубки, гибкого металлического шланга, диффузора (раструба-снегообразователя), рукоятки и предохранителя. Запорный вентиль имеет предохранительное устройство в виде мембраны, которое срабатывает при повышении давления в баллоне сверх допустимого. Газ в баллоне находится под давлением порядка 70 атмосфер (6 – 7 МПа) в жидком состоянии. Огнетушители приводятся в действие при вращении запорного вентиля против часовой стрелки. При открытии вентиля углекислый газ выходит наружу в виде снега. При повышении окружающей температуры давление в баллоне может достигать 180 – 210 атмосфер (180 — 210⋅105 Па). Время действия углекислотных огнетушителей до 60 с, а дальность – до 2м. [17]

Порошковые огнетушители заряжены огнетушащим порошком и закачаны газом (воздух, азот, углекислота) до давления 16 атм. Предназначены для тушения пожаров класса А, В, С или ВС в зависимости от типа применяемого порошка, а также электроустановок, находящихся под напряжением до 1000 В. Снабжены запорными устройствами, обеспечивающими свободное открытие и закрытие простым движением руки. Манометр установленный на головке огнетушителя и показывающий степень его работоспособности, является большим преимуществом перед огнетушителями со встроенным источником давления. Эксплуатируется при температуре от -40 до +50 0С.

Таким образом, пожарная безопасность на предприятии обеспечивается комплексом мероприятий, предотвращающих возникновение пожара и системой пожарной защиты, обеспечивающей успешную борьбу с возникшим пожаром или последствиями взрыва.

ЗАКЛЮЧЕНИЕ

В ходе дипломного проектирования был разработан программный модуль для интеграции пользовательских данных с привязкой к GPS координатам в картографические сервисы. Реализованный модуль позволяет работать с пользовательскими данными более эффективно, не передавая их при этом третьим лицам. К преимуществам данного модуля над аналогами можно отнести:

1. Гибкость при использовании фильтров, в том числе и создание собственных фильтров предварительной обработки данных.
2. Единый формат передачи данных на карту, что позволяет использовать любое количество данных модулей, связанных с различными хранилищами пользовательской информацией, поставляющий геоданные.
3. Скорость обработки запросов достаточно высокая за счет уменьшения количества информации, отдаваемой пользователям и кластеризации пространственных данных на клиентской стороне.
4. Набор сервисов, моделей и интерфейсов для создания дополнительного функционала, легко расширяющего текущие возможности программного модуля.
5. Полностью описанный интерфейс программирования приложения (API). Документация позволит использовать данный модуль сразу же после ознакомления с ним.

Серверная часть программного модуля работает на базе .NET платформы. Клиентская часть основана на базе современной и свободно распространяемой библиотеки Leaflet, которая поддерживает большое количество картографических провайдеров карт. Обе части модуля реализованы так, что обновлять их можно независимо друг от друга, и при этом пользователи будут всегда работать с самой актуальной версией платформы и библиотеки.

СПИСОК ЛИТЕРАТУРЫ

[1] Шилдт, Г. Полный справочник по С#.: / Г. Шилдт – М.: Издательский дом «Вильямc», 2004. – 752 с.

[2] Рихтер Дж. Программирование на платформе Microsoft .NET Framework – [2-е изд., испр.] – М.: Издательско-торговый дом «Русская Редакция», 2003. – 512 с.

[3] .NET Framework. Свободная энциклопедия [Электронный ресурс]. – Электронные данные. – Режим доступа: http://ru.wikipedia.org/wiki/.NET\_Framework.

[4] C# Version 5.0 Specification [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.microsoft.com/en-us/download/details.aspx?id=7029.

[5] Библиотека MSDN [Электронный ресурс]. – Электронные данные. – Режим доступа: http://msdn.microsoft.com/ru-ru/library/default.aspx.

[6] LeafletJS Library [Электронный ресурс]. – Электронные данные. – Режим доступа: http://leafletjs.com.

[7] Leaflet 0.4 — новая версия открытой JS-библиотеки для интерактивных карт [Электронный ресурс]. – Электронные данные. – Режим доступа: http://habrahabr.ru/post/148744.

[8] Карты Google Maps [Электронный ресурс]. – Электронные данные. – Режим доступа: http://maps. google.com.

[9] Сайт картографического веб-сервиса компании Microsoft [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://www.bing.com/maps/>.

[10] Карты Yahoo! Maps для разработчиков [Электронный ресурс]. – Электронные данные. – Режим доступа: http://developer. yahoo.com/maps/.

[11] Карты OpenStreetMaps [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://www.openstreetmap.org/>.

[12] Карты Wikimapia [Электронный ресурс]. – Электронные данные. – Режим доступа: http://wikimapia.com

[13] Сервис CartoDB [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://cartodb.com>

[14] Носенко А.А. Методические указания по технико-экономическому обоснованию дипломных проектов инженерного профиля / А. А. Носенко. – Мн. : БГУИР, 2011. – 28 с.

[15] Категорирование помещений, зданий и наружных установок по взрывопожарной и пожарной опасности [Электронный ресурс]. – Электронные данные. – Режим доступа: http://mchs.gov.by/\_modules/\_cfiles/

files/TKP\_474-2013.pdf.

[16] Закон Республики Беларусь от 15 июня 1993 года «О пожарной безопасности» [Электронный ресурс]. – Электронные данные. – Режим доступа: http://pravo.by/main.aspx?guid=3871&p0=v19302403&p2={NRPA}.

[17] Общие правила пожарной безопасности Республики Беларусь для общественных зданий и сооружений [Электронный ресурс]. – Электронные данные. – Режим доступа: http://mchs.gov.by/\_modules/\_cfiles/files/ppb\_1.04-2002\_1033.pdf.

[18] Правила пожарной безопасности Республики Беларусь при эксплуатации технических средств противопожарной защиты [Электронный ресурс]. – Электронные данные. – Режим доступа: http://mchs.gov.by/

\_modules/\_cfiles/files/ppb\_1.02-94.pdf.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код

1. Store.cs.

public class Store<T> : IStore<T> where T : class, IEntity

{

private IDataSource dataSource;

public Store(IDataSource dataSource)

{

this.dataSource = dataSource;

}

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

protected virtual void Dispose(bool disposing)

{

if (disposing)

{

if (dataSource != null)

{

dataSource.Dispose();

dataSource = null;

}

}

}

public virtual IQueryable<T> GetQuery()

{

return dataSource.Query<T>();

}

public virtual T First(Expression<Func<T, bool>> predicate)

{

return GetQuery().FirstOrDefault(predicate);

}

public virtual Task<T> FirstAsync(Expression<Func<T, bool>> predicate)

{

return new Task<T>(() => First(predicate));

}

public virtual Task<T> SingleAsync(Expression<Func<T, bool>> predicate)

{

return new Task<T>(() => Single(predicate));

}

public virtual T Single(Expression<Func<T, bool>> predicate)

{

return GetQuery().SingleOrDefault(predicate);

}

public virtual IEnumerable<T> FindAll(Expression<Func<T, bool>> predicate)

{

return GetQuery().Where(predicate).ToList();

}

public virtual Task<IEnumerable<T>> FindAllAsync(Expression<Func<T, bool>> predicate = null)

{

return new Task<IEnumerable<T>>(() => FindAll(predicate));

}

public virtual IEnumerable<T> GetAll()

{

return GetQuery().ToList();

}

public virtual void Add(T entity)

{

dataSource.Add(entity);

}

public Task AddAsync(T entity)

{

#if NET40

return Task.Factory.StartNew(() => Add(entity));

#else

return Task.Run(() => Add(entity));

#endif

}

public virtual void Update(T entity)

{

dataSource.Update(entity);

}

public virtual Task UpdateAsync(T entity)

{

#if NET40

return Task.Factory.StartNew(() => Update(entity));

#else

return Task.Run(() => Add(entity));

#endif

}

public virtual void Delete(T entity)

{

dataSource.Delete(entity);

}

public Task DeleteAsync(T entity)

{

#if NET40

return Task.Factory.StartNew(() => Delete(entity));

#else

return Task.Run(() => Add(entity));

#endif

}

public virtual T Get(Guid id)

{

return GetQuery().FirstOrDefault(x => x.ID == id);

}

public virtual Task<T> GetAsync(Guid id)

{

return new Task<T>(() => Get(id));

}

public virtual T[] Get(params Guid[] ids)

{

return GetQuery().Where(x => ids.Contains(x.ID)).ToArray();

}

public virtual Task<T[]> GetAsync(params Guid[] ids)

{

return new Task<T[]>(() => Get(ids));

}

public void SaveChanges()

{

this.dataSource.SaveChanges();

}

public Task SaveChangesAsync()

{

return this.dataSource.SaveChangesAsync();

}

}

1. FluentAPI.cs

public static class FluentAPI

{

public static IQueryable<T> QueryDataSource<T>(IDataSource dataSource) where T : class, IGeoEntity

{

return dataSource.Query<T>();

}

internal static IQueryable<T> QueryGeoData<T>(this IQueryable<T> query) where T : IGeoEntity

{

return query;

}

public static CollectionContext<T> CollectGeoData<T>(this IQueryable<T> query) where T : IGeoEntity

{

return new CollectionContext<T>(query) { TotalCount = query.Count() };

}

public static IList<T> ListGeoData<T>(this IQueryable<T> query) where T : IGeoEntity

{

return query.ToList();

}

public static IEnumerable<T> GetGeoData<T>(this IQueryable<T> query) where T : IGeoEntity

{

return query.ToList();

}

public static IQueryable<T> FilterBy<T, TR>(this IQueryable<T> query, params object[] context) where T : IGeoEntity

{

var type = typeof (TR);

return query;

}

public static IQueryable<T> FilterBy<T>(this IQueryable<T> query, IFilter filter, IFilterArgs args) where T : IGeoEntity

{

return filter.ApplyFilter(query, args);

}

public static IQueryable<T> FilterBy<T>(this IEnumerable<T> query, IFilter filter, IFilterArgs args) where T : IGeoEntity

{

return filter.ApplyFilter(query.AsQueryable(), args);

}

public static string Append(this string current, string key, string value)

{

return String.IsNullOrEmpty(value) ? current : current + key + value + "&";

}

public static object GetAttributeValue(Type objectType, string propertyName, Type attributeType, string attributePropertyName)

{

var propertyInfo = objectType.GetProperty(propertyName);

if (propertyInfo != null)

{

if (Attribute.IsDefined(propertyInfo, attributeType))

{

var attributeInstance = Attribute.GetCustomAttribute(propertyInfo, attributeType);

if (attributeInstance != null)

{

foreach (PropertyInfo info in attributeType.GetProperties())

{

if (info.CanRead &&

String.Compare(info.Name, attributePropertyName,

StringComparison.InvariantCultureIgnoreCase) == 0)

{

return info.GetValue(attributeInstance, null);

}

}

}

}

}

return null;

}

}

1. BoundsFilter.cs

public sealed class BoundsFilter : Filter

{

private const double bounds = 0.05;

public override IQueryable<T> ApplyFilter<T>(IQueryable<T> query, IFilterArgs args)

{

var northEast = args.Params["northEast"] as LatLng;

var southWest = args.Params["southWest"] as LatLng;

if (northEast != null && southWest != null)

query = query.Where(x =>

x.Latitude + bounds > southWest.Latitude &&

x.Latitude - bounds < northEast.Latitude &&

Math.Abs(x.Longitude) + bounds > Math.Abs(northEast.Longitude) &&

Math.Abs(x.Longitude) - bounds < Math.Abs(southWest.Longitude));

return query;

}

public void GetData()

{

}

}

1. LatLng.cs

[TypeConverter(typeof(LatLngConverter))]

public class LatLng

{

public LatLng() { }

public LatLng(double lat, double lng)

{

this.Latitude = lat;

this.Longitude = lng;

}

public double Latitude { get; set; }

public double Longitude { get; set; }

public static bool TryParse(string s, out LatLng result)

{

result = null;

var parts = s.Split(';');

if (parts.Length != 2)

{

return false;

}

double latitude, longitude;

if (double.TryParse(parts[0], out latitude) &&

double.TryParse(parts[1], out longitude))

{

result = new LatLng { Longitude = longitude, Latitude = latitude };

return true;

}

return false;

}

internal virtual JsonObjectBuilder ToJsonObjectBuilder()

{

var json = new JsonObjectBuilder();

json.Append("lat", this.Latitude);

json.Append("lng", this.Longitude);

return json;

}

public string Render()

{

return this.ToJsonObjectBuilder().Render();

}

}

ПРИЛОЖЕНИЕ Б

(обязательное)

Спецификация

ПРИЛОЖЕНИЕ В

(обязательное)

Ведомость документов