Таблица 1. Среднее время хеширования случайных данных размером 512кб

Алгоритм	Node.js	Python	Go
MD5	2.23 мс	0.98 мс	0.75 мс
SHA256	2.39 мс	1.82 мс	0.38 мс
SHA512	2.12 мс	2.14 мс	0.49 мс

Таблица 2. Среднее время генерации случайного ключа размером 1024 бита

Алгоритм	Node.js	Python	Go
RSA	320.5 мс	580.5 мс	60.7 мс
DSA	N/A	674.8 мс	N/A

Особенности применения криптографических алгоритмов на практике.

В соответствии с результатами исследования, при выборе конкретного алгоритма следует в первую очередь оценить, насколько критичными являются скорость и уровень защищенности алгоритма для данной задачи. Наилучшим вариантом является применение алгоритмов, реализованных на уровне ОС или оборудования. Помимо этого, следует обратить внимание на то, что гораздо безопаснее применять алгоритмы, реализованные «из коробки», – таким образом можно минимизировать вероятность появления ошибок и увеличить устойчивость к попыткам взлома.

Список использованных источников:

1. Node.js Crypto Documentation [Electronic resource] / Node.js v5.9.1 Documentation. – Node.js Foundation, 2016. – Mode of access: https://nodejs.org/api/crypto.html. – Date of access: 26.03.2016.
2. go/crypto documentation [Electronic resource] / The Go Programming Language. – Build version go1.6, 2016. – Mode of access: https://golang.org/pkg/crypto/. – Date of access: 27.03.2016.
3. Python hashlib documentation [Electronic resources] / Python. – Python Software Foundation, 2016. – Mode of access: https://docs.python.org/3/library/hashlib.html. – Date of access: 26.03.2016.
4. Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер. – М. : Триумф, 2002. – 816 с.

СОБЫТИЙНО-ОРИЕНТИРОВАННАЯ МОДЕЛЬ ОБРАБОТКИ НАБОРОВ ДАННЫХ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Сафонов А. А.

Ганжа В. А. – кандидат физико-математических наук, доцент

С ростом объёмов информации, требуются новые способы их обработки. Пользователям требуется быстрый доступ к новой информации и возможность получения данных в реальном времени. Это стимулирует разработчиков создавать отзывчивые интерфейсы и модели для обработки.

Разработанная модель позволяет создавать и преобразовывать наборы данных и реагировать на их изменения. Она направлена на повышение качества кода при написании решений для обработки данных. В текущей реализации только на платформе .NET Framework.

На рисунке 1 представлена схема обработки запроса на вставку в набор чисел:

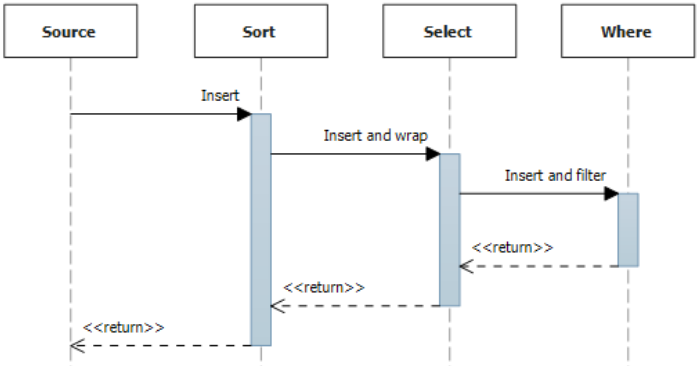

Рисунок 1 – Схема запроса на обновление дочерних наборов

Код на языке C#:

```
var resultCollection = source.SortRc(…).SelectRl(…).WhereRl(…);
source.Add(number);
```

Листинг 1 – Пример построения обновляемого посредством событий набора на языке C#

В результате исполнения кода из листинга 1, мы получим новый набор данных, который будет реагировать на изменения исходной коллекции. Операции сортировки, преобразования и фильтрации будут применяться к новым элементам набора и реагировать на изменения текущих.

В рамках проекта создана библиотека с базовым набором функций, которые повторяют функциональность операций языка SQL. Например: Select, Where, Join и т. д.

Для создания рабочего проекта использовался язык C#. В ходе исследования были изучены материалы, предоставленные Microsoft Research по построению систем, основанных на событийных моделях

Таким образом, была разработана система, реализующая спектр операций по обработке данных учитывая принципы реактивного программирования. Данный проект может применяться при построении пользовательских интерфейсов, где пользователю важно получать актуальную информацию в кратчайшие сроки.

Список использованных источников:

1. Lee Campbell, Introduction to Rx - 2012
2. Джеффри Рихтер CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# - 2015
3. [Электронный ресурс] LINQ (Language-Integrated Query) - https://msdn.microsoft.com/ru-ru/library/bb397926.aspx - 2016