Слайд 1

Представление и приветствие

Слайд 2

Постановка задачи

Перед нами была поставлена задача реализовать парсер Firefox Cache/History данных.

* Приложение для получения данных из Firefox-a (истории посещений, закладки, сохраненные пароли и логины, файлы в кеше и т.д.).
* Экспорт полученных данных на диск
* Поиск по полученным данным.

В GUI должно быть отображение полученных данных в виде таблицы.В гриде должны отображаться колонки size, url, и другие распаршерные поля. File меню с опцией открытия файла, опция экспорта, опцией поиска текста. Также необходимо реализовать пейджирование для просмотра полученных данных. Количество элементов должно задаваться.

Должны быть следующие функциональные возможности:

* Возможности экспорта на диск из right-click меню или файл меню для конкретного файла.
* Поиск по тексту
* Возможность считать md5 файлов при экспорте (записывать его в отдельный файл и ложить рядом с экспорченным файлом).

Слайд 3

Какие коррективы в постановки задачи были сделаны

* После ресерчинга стало понятно, что для того, чтобы получить данные из Firefox нужно учитывать много факторов. Одним из которых стали профили. Т.к. в браузере может быть несколько профилей и они могут быть сохранены на диске, перед нами возникла потребность учитывать этот нюанс, чтобы получать всегда корректные данные от каждого пользователя.
* Следющая задача, которая побудила сделать коррективы, стала многопоточность приложения. Т.к. количестов записей истории, закладок очень большое и превышает несколько тысяч, то для того, чтобы комфортно их просматривать, возникла потребность запускать их парсер в новом потоке, чтобы приложение не зависало. Особенно проблема с многопоточностью остро ощущалась при парсинге кэша.
* Еще одна не маловажная деталь, которая заставила вносить коррективы – это определение путей профилей firefox. Для этого надо было узнавать пути к профилям через реестр.
* Поскольку уже почти каждое ПО является мультиязычным, то было принято решение внести коррективы и добавить в нашу программу мультиязычность.
* Поскольку записи истории не хранятся на диске в открытой форме, а храняться в sqlite-базе, то возникла необходимость подключение сторонних библиотек для работы с этими базами, что в следствии ведет к коррективам в постановки задачи
* Получение логинов и паролей имеет схожую проблему, что и получение записей истории. Однако, для того, чтобы получить пароли/логины необходимо распарсивать json-файлы, далее его декриптовать. Такой порядок действий заставил внести корректиы в постановке задачи.
* И последний момент, который вынудил внести корректировки – это парсинг кэш-файлов. Поскольку каждый кэш-файл – это единая информационная сущость, то нужно было продумать механизм, как эти файлы обработать одним проходом или все или заданное количество.

Слайд 4

Источники информации

* Поскольку данный проект довольно большой и, соответственно, мы нуждались в хорошей литературе, чтобы написать максимально эффективный код. Хорошим фундаментом послужила книга Харви М. Дейтел, Пол Дж. Дейтел – Как программировать на C++. В данной книге хорошо описано не только, то как нужно программировать на С++ с соблюдением различных правил хорошего тона, но и еще даны основы UML, который мы использовали при создании архитектуры программы.
* Следующий источник – Шлее М., Qt 5.3 Профессиональное программирование на С++ - использовалась при написании GUI, а также для понимания того, какие отличия есть в методах в методах из библиотек Qt в сравнении с такими же матодами из STL. Также эта книга помогла при создании мультиязычного приложения.
* Хоть и предыдущая книга хороша, но часто приходилось быстро находить информацию о каких-либо нюансах или же классах Qt. В данном случаи очень выручала официальная онлайн документация.
* Не менее важной является документация по библиотеки boost. Поскольку, boost у нас использован во многих местах: От парсинга файлов, до парсинга Истории и закладок.
* Скотт Чакон – Pro Git – данная книга использовалась для понимания того как устроен git, как с ним обащаться, чтобы сохранить свой проект и его работоспособность и в случаи какой-то неудачи вернуться на прошлую версию.
* Очень хорошим источником были лекции по Modern & Effective C++ и Управления ресурсами. Даныые две лекции помогли усвоить принципы работы конструкторов перемещений, а также понять, как работают умные указатели, который потом используются в нашей програамее.
* И последняя книга очень помогла при продумке архитектуры нашей библиотеки. Чтобы либа была максимально эффективной и не содержала избыточность.

Слайд 5

При написании нашего приложения мы использовали следующие технологии:

* RAII
* QtConcurrent
* ООП

А также паттерны

* 1

Слайд 6

Планирование

Слайд 7

Работая над проектом мы научились командой работы, прислушиваться к мнению друг друга, находить компромисс в каких то спорных ситуациях. Так же научились работать с системой контроля версий, продумывать архитектуру приложения, продумывать оптимальные решения задач.

Слайд 8

При написании приложения мы столкнулись с следующими проблемами:

* Получения путей к профилям Firefox из реестра
* Расшифрование паролей
* Реализация многопоточности, чтобы приложение не лагало при парсинге большого обьема данных.

Слайд 9

Что понравилось из лекций, над какими задачами хочеться работать в дальнейшем.

Из лекций мне понравилось понравилось все: а подача материала и сами материалы. Понравилось потому, что подавалась свежая и актуальная информация от людей, которые постоянно с этим работают и очень хорошо разбираются. Также очень понравились лекции и домашние задания по управлению ресурсами, modern & effective c++, code review и особенно понравилось из этого списка лекция и домашка по reverse engineering.

В дальнейшем хочется работать над какими-то подобными проектами или же над проектами, которые очень затрагивают тему управления ресурсами, modern c++, reverse engineering и Qt.

Слайд 10

Особенности приложения

Слайд 11

Наша парная работа была построена по принципу парного программирования. Т.е. мы работали надо одной задачей, переодически менялись. Допустим кто-то один из нас пишет код, кто-то обдумывает какие-то решения действия или решения проблем, потом наоборот. После каждого коммита проверяли и тестировали код. Допустим, я написал какой-то код, Дима его тестирует, находит/не находит баги. Потом исправляем их. И наоборот. Такой подход гараздо эффективен и позволяет писать более качественный код.

Слайд 12

Вопросы