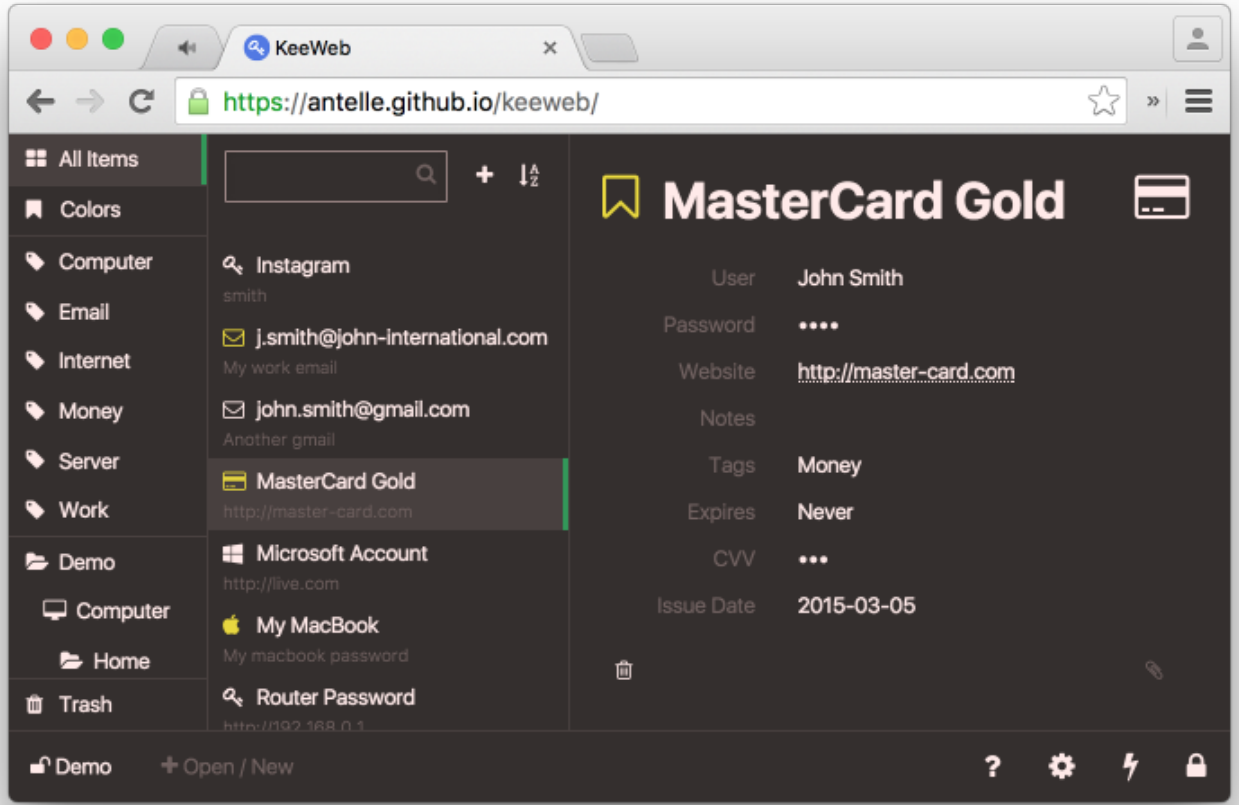
РАЗРАБОТКА ВЕБ-САЙТОВ*, ПРОГРАММИРОВАНИЕ*, ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ*, JAVASCRIPT*, HTML*

Как я делал веб-версию KeePass

Antelle 1 ноября 2015 в 09:47 62,2k

Как-то мне надо было добавить в админку просмотр списка паролей. База хранилась на сервере в формате KeePass (kdbx v2), сервер был на ноде — недолго думая, я взял первый попавшийся пакет и сделал. А потом понадобилось то же самое, но прямо у пользователя в браузере, без сервера. Ничего не нашлось. Первым желанием было форкнуть либу и заменить использование node api, но от первого просмотра кода желание пропало, решил сделать сам.



Под катом расскажу о проблемах, с которыми я столкнулся, и способах их решения

Чтение формата kdbx в браузере

Пришлось изучить формат kdbx, не описанный, к сожалению, нигде кроме исходников и этой небольшой статьи. Но зато он достаточно прост. Чтобы прочитать kdbx, нужно:

1. прочитать бинарный заголовок (что в нём?)
2. инициализировать алгоритм шифрования и генератор случайных значений (salsa-20)
3. посчитать хэш пароля и ключа, получить credentials hash
4. N раз (значение из заголовка) зашифровать хэш, получив мастер-ключ — это самая вычислительно трудоёмкая операция

5. дешифровать данные
6. проверить корректность расшифровки, сравнив блок данных с блоком из заголовка
7. разGZIPать данные
8. распарсить xml
9. сгенерировать соль для защищённых полей
10. по требованию, когда они нужны, теперь можно расXORить защищённые поля
11. прочитать xml-метаданные (что в них?)
12. прочитать группы и записи

Из алгоритмов там используются AES для симметричного шифрования и SHA256 для хеширования. Самая быстрая реализация для web, которую я нашёл — это asmCrypto, она написана на asm.js, работает быстрее остальных браузерных альтернатив и, что немаловажно, умеет собираться по кусочкам, включая только нужные модули с алгоритмами. Пропатчив её так, чтобы все циклы трансформации ключа происходили внутри asm.js, без дополнительного копирования данных, мне удалось достичь скорости в 4..7 раз меньшей, чем у native-реализации (если в KeePass поставить задержку в 1 секунду, откроется файл секунд за 6). С дефолтными настройками время открытия файла в среднем около 200мс. WebAssembly нас всех спасёт, а пока что вот так.
UPD: в комментариях подсказали про WebCrypto, теперь время открытия в браузерах с его поддержкой почти не уступает KeePass.

Для работы с gzip взял pako — он маленький, быстрый и MIT.

Сравнительно с шифрованием, значимых затрат на декомпрессии не заметил.

В результате получилась kdbxweb, которая работает в node.js и современных браузерах. Из-за необходимости работать с бинарным форматом, в списке поддерживаемых браузеров нет старых версий. Библиотека+зависимости получились в 150кб.

Приложение

Итак, либа есть. А почему бы теперь не написать веб-интерфейс для неё? Хороший менеджер паролей для веба должен:

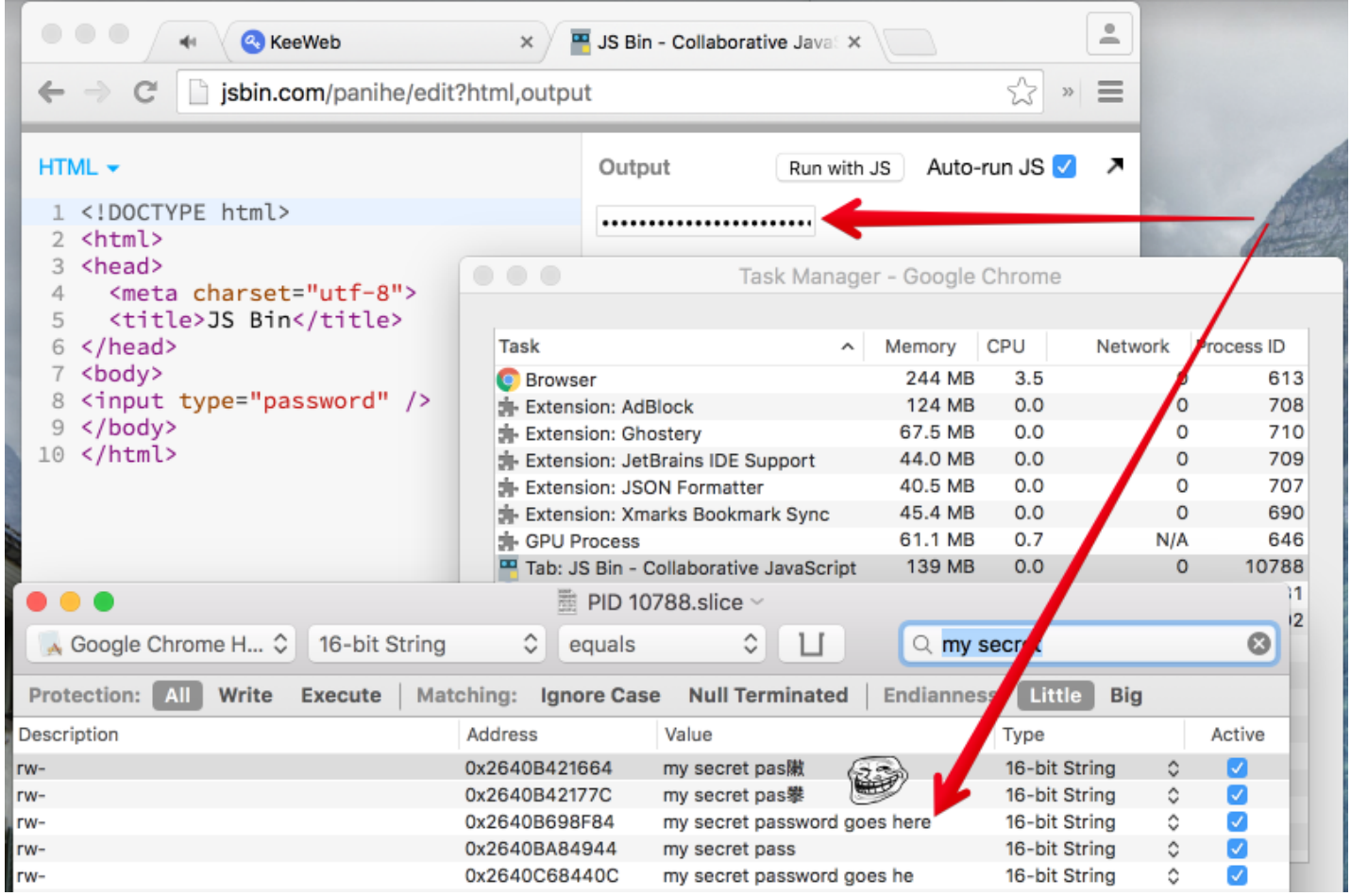
- быть одиним файлом, чтобы можно было взять и, скажем, положить себе в дропбокс, на хостинг или даже на диск
- не требовать абсолютно никакого сервера, ничего никуда не пересылать, работать в браузере
- занимать меньше 1МБ (условно; важен порядок на самом деле)
- уметь работать из файла и оффлайн
- быстро загружаться
- работать во всех браузерах
- не хранить в памяти пароли в открытом виде
- синкаться с дропбоксом
- использовать существующий формат, не изобретать велосипед
- так или иначе поддерживать все фичи формата
- работать с несколькими файлами/базами одновременно
- быть обратно-совместимым с оригинальным приложением
- быть опен-сорсным

Большинство из этого удалось достичь.

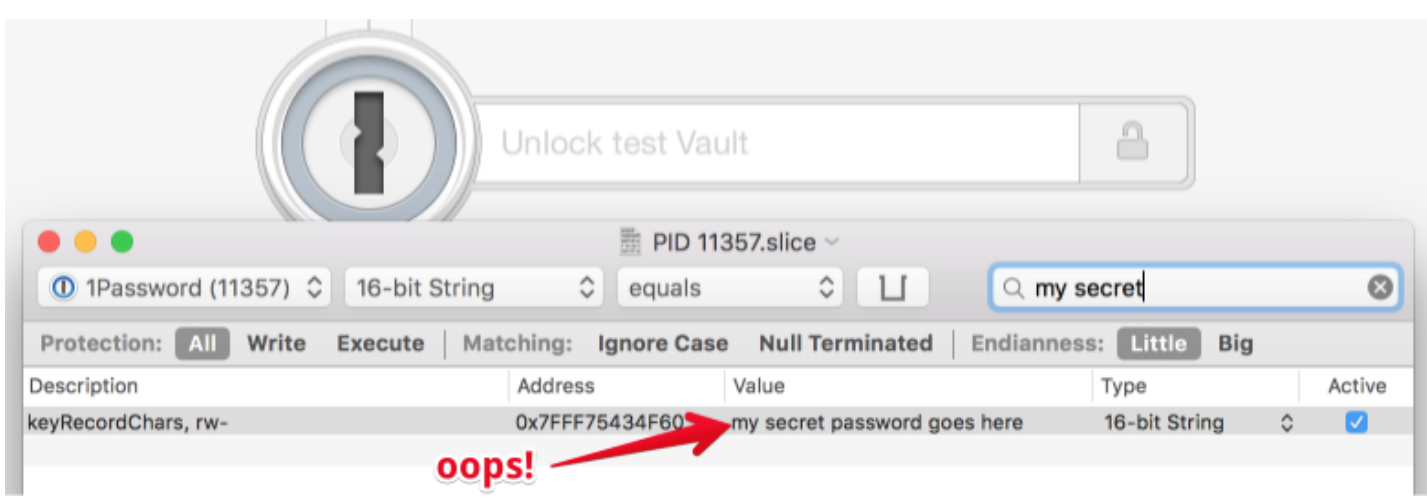
Приложение написал на backbone+zepto (сначала хотел попробовать Angular2, но как-то не пошло, сгенерированный код самого фреймворка пока что получается размером более 1MB, наверное ещё слишком бета и потом будет лучше).
Версии браузеров, поддерживающих читалку формата, не так далеки от последних, поэтому кроссбраузерных костылей практически нет.

input type=password

Когда вы вводите текст в любой инпут, все вводимые значения даже с историей ввода могут сохраняться в памяти браузера бесконечно долго. И если в десктоп-приложении можно почистить память, то в браузере у вас нет контроля за памятью, занимаемой строками: когда GC захочет, тогда и почистит, при этом память он может занулить, а может и нет:

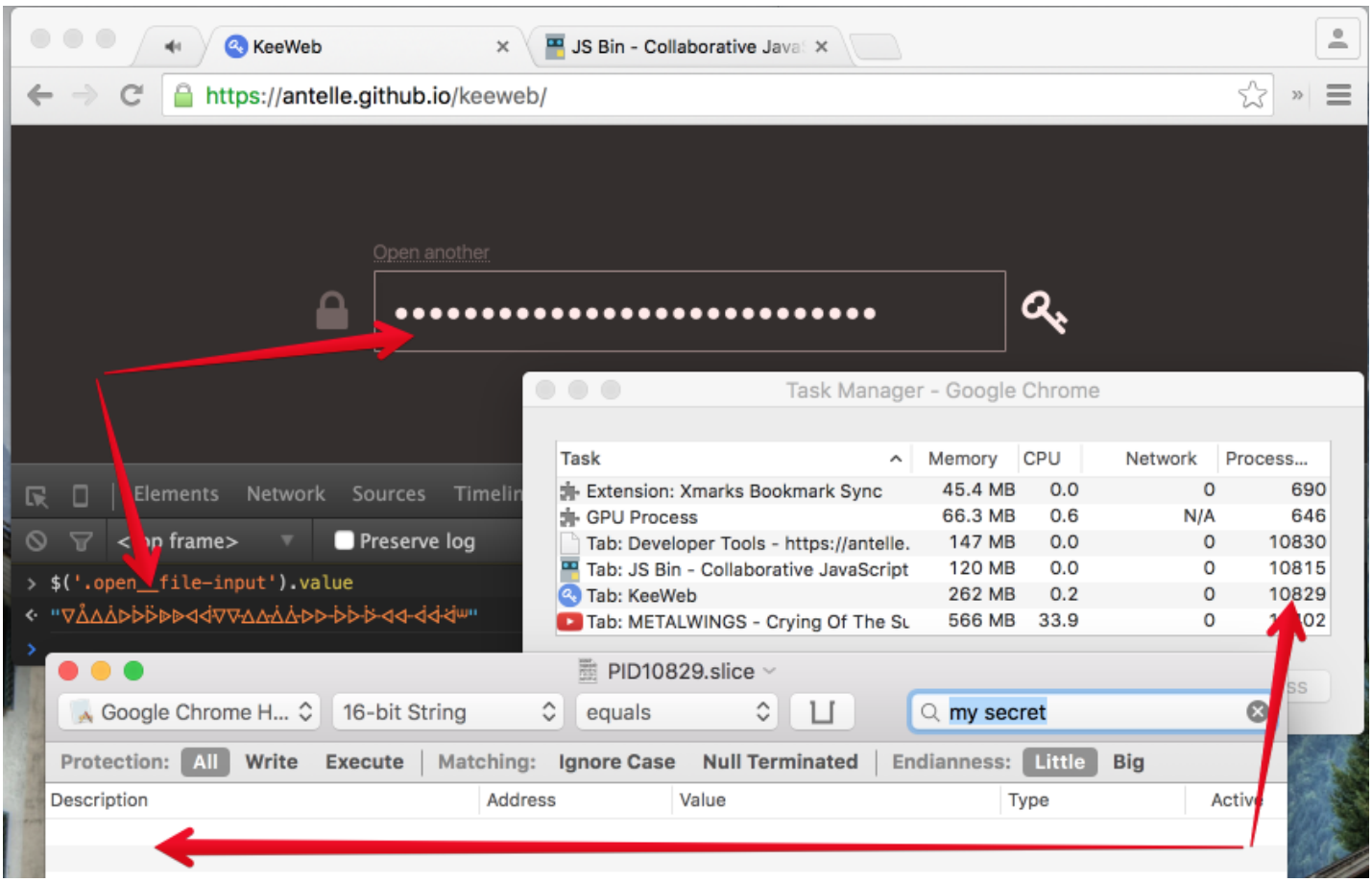


Некоторые приложения считают, что память можно не чистить даже после закрытия файла, хотя для приложений это, конечно, не так критично, как для веба:



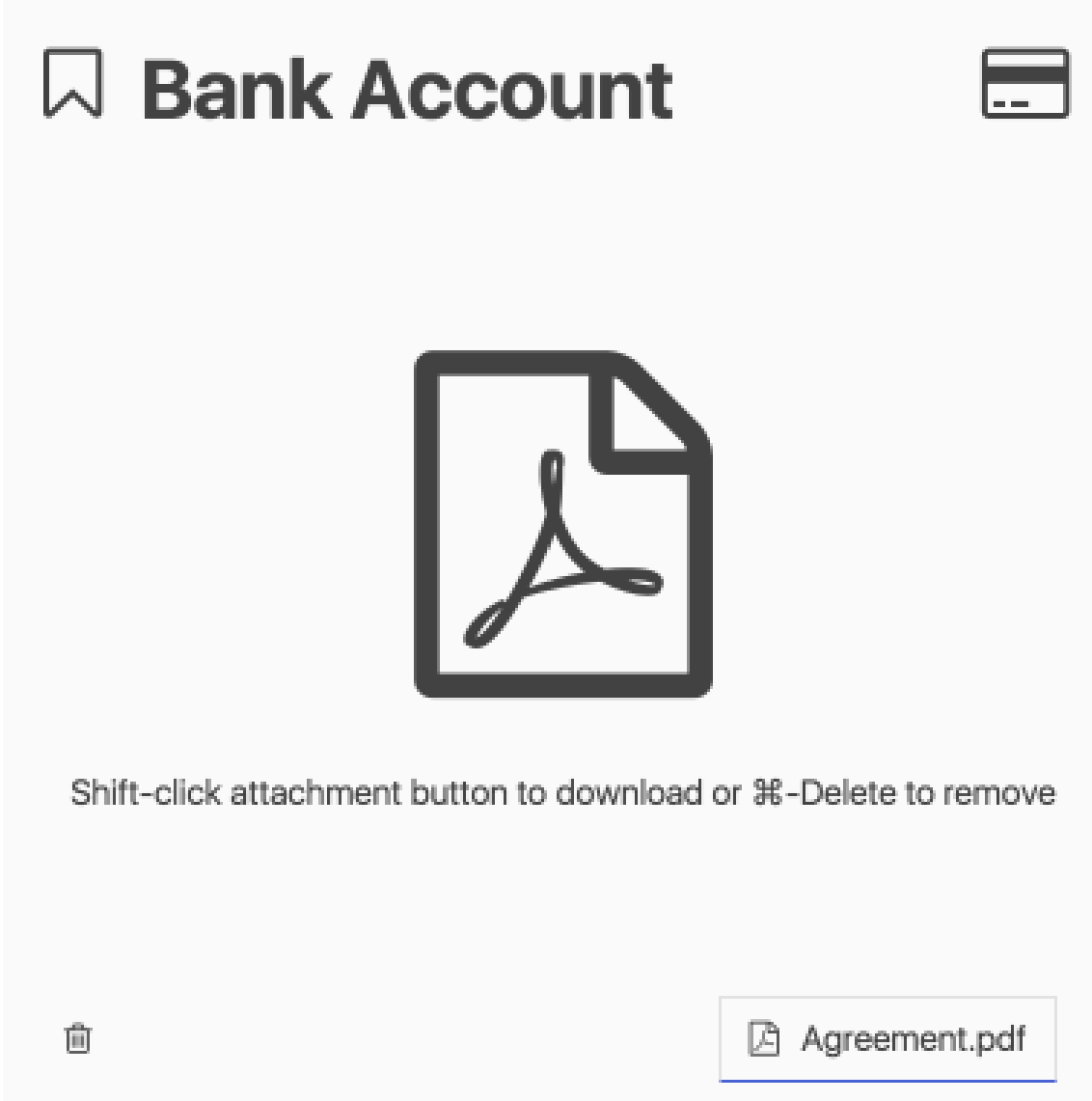
Так не пойдёт, если в приложении ещё куда ни шло, то в браузере я бы в такой инпут мастер-пароль вводить не стал, поэтому пришлось сделать велосипед: при вводе символы XOR-ятся с

рандомной последовательностью, подобранной так, чтобы на выходе получились символы некоего неиспользуемого блока unicode. Их и показывает инпут, думая, что это значение (всё равно там звёздочки), само значение при этом посимвольно сохраняется в буфер. Сложить значение с ключом, имея полный доступ к памяти, конечно, можно, но это уже другой уровень сложности:



Сохранение файлов

Сохранить файл, сгенерированный в браузере, не так-то просто, но нужно:



Есть FileSaver.js, но в нём засада в Safari, вот тред с интересными комментариями, где автор просит купить ему макбук. Почти все браузеры сейчас поддерживают

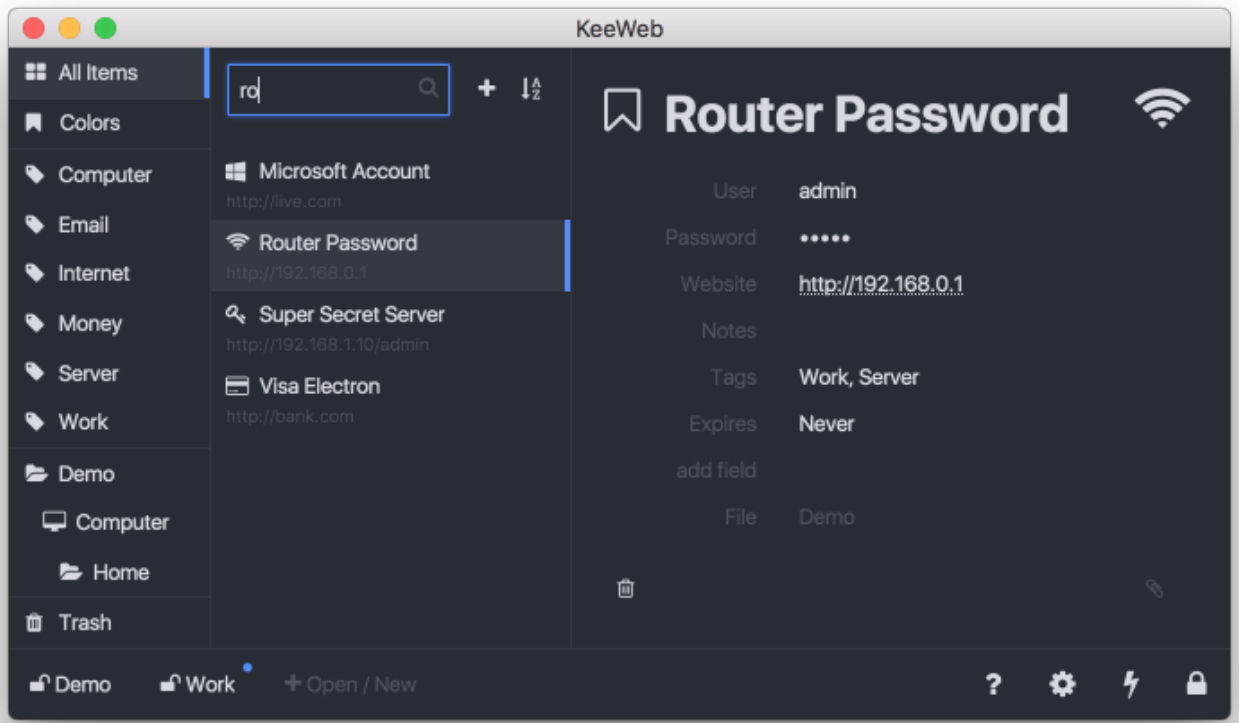
```
<a href="" download="file-name.ext">
```

Но только не Сафари. Сафари упорно не хочет ни понимать атрибут download (чувсвтва по этому поводу можно излить на webkit.org), ни скачивать Blob, жалуясь на то, что это якобы небезопасно. Но если законвертировать файл в base64, скачать его таки можно. Отправил автору патч, теперь в сафари FileSaver работает, но имя файла всё равно осталось Unknown. Если кто-то

знает способ сделать лучше — расскажите, пожалуйста.

Десктоп

Работать, каждый раз скачивая файл, неудобно, собрал приложения на electron. Работает он шустро, API отлично описанное и интуитивно понятное, проблем с ним не заметил вообще:

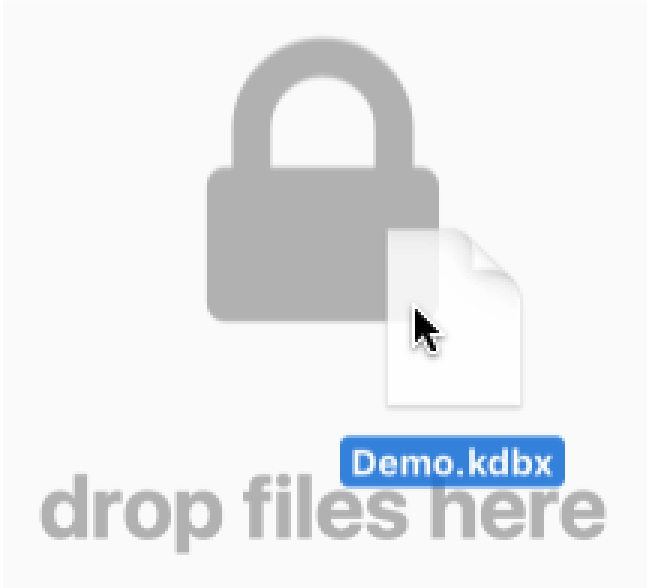


Авто-апдейтер пока только под мак, но в будущем скорее всего его допишут и под другие ос. Как и для node-webkit, для электрона уже появились electron-builder и electron-packager, которыми я и воспользовался для сборки приложения и инсталлятора.
Конфиг для сборки инсталлятора, который делает dmg для mac os

x и exe installer для windows (сборку deb для linux пока они не поддерживают):

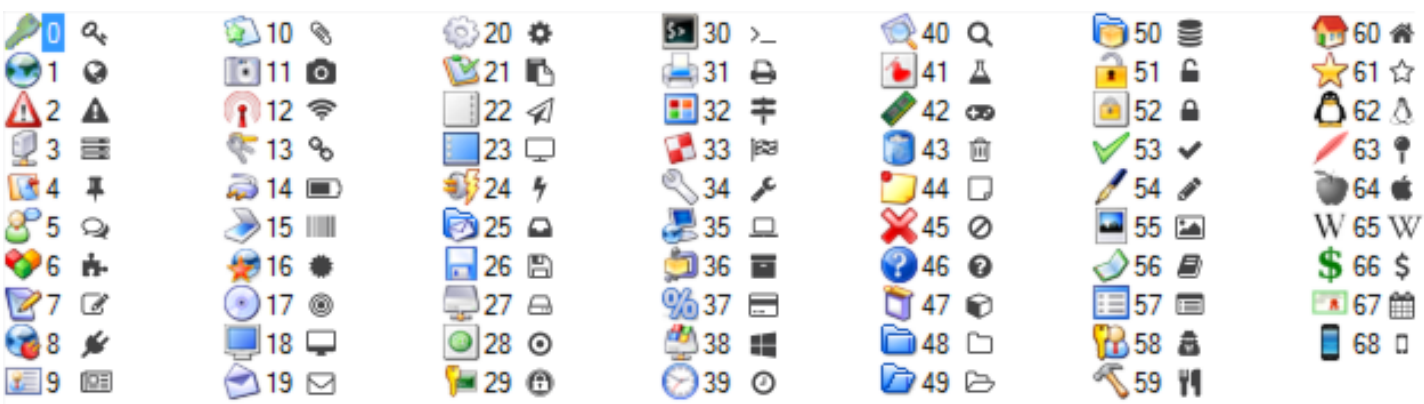
```
{
  "osx" : {
    "title": "KeeWeb",
    "background": "../graphics/dmg-bg.png",
    "icon": "../graphics/app.icns",
    "icon-size": 80,
    "contents": [
      { "x": 438, "y": 344, "type": "link", "path": "/Applications"
},
      { "x": 192, "y": 344, "type": "file" }
    ]
  },
  "win" : {
    "title": "KeeWeb",
    "icon": "graphics/app.ico"
  }
}
```

Electron добавляет приятную возможность получить путь открываемых файлов. Когда в приложение тащат файл или открывают его из инпута, в File добавляется свойство path, в котором содержится полный путь к файлу. Сохранить файл можно потом как обычно в ноде, fs.writeFile.



Иконки

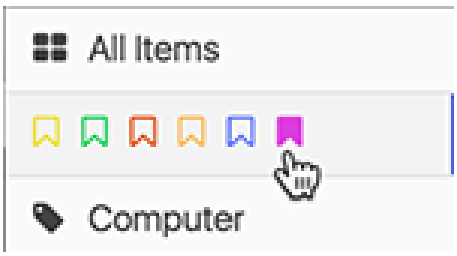
В KeePass есть много иконок со странными названиями в енуме. Но во-первых, они немного устарели для 2015, а во-вторых, растровую графику использовать не хотелось, чтобы сохранить приложение маленьким. Пришлось составить соответствие вручную, к счастью, в font awesome нашлось абсолютно всё:



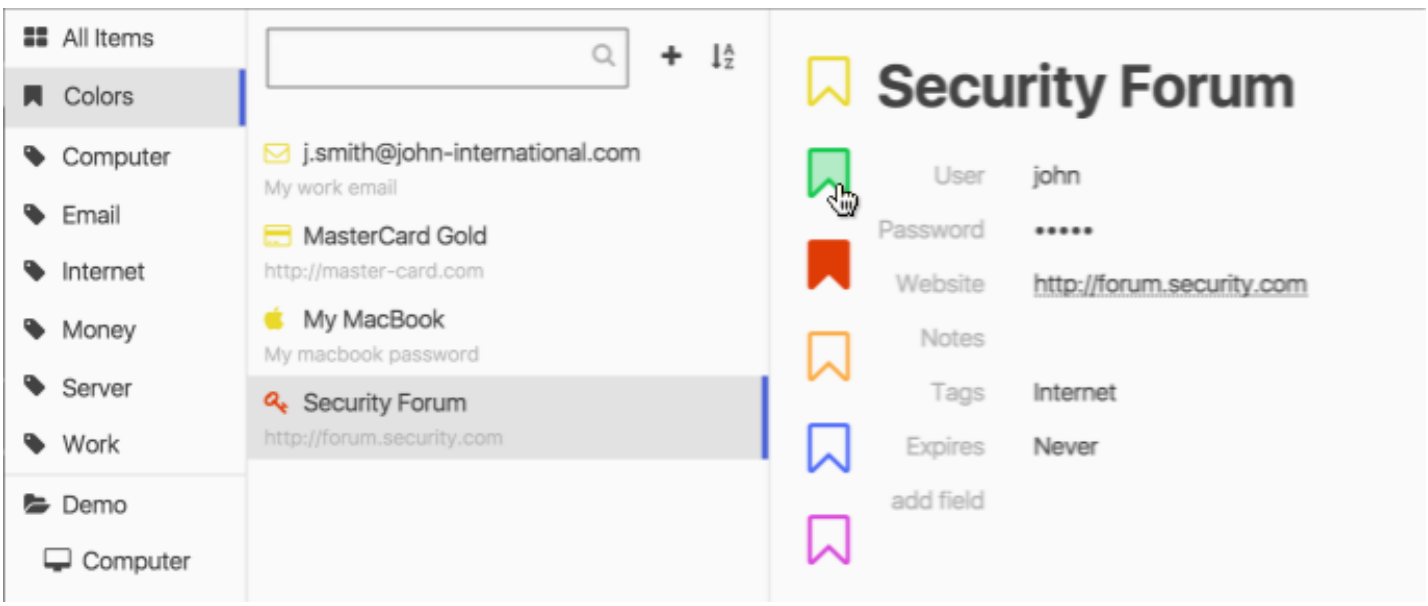
Цвета

Формат позволяет задать цвет записи и цвет фона, что KeePass и использует, однако я считаю возможность выбирать произвольный цвет текста и фона самому из колор-пикера не лучшим решением.

Цвет записи нужен для отнесения записи к какой-любо логической группе (например, зелёные — деньги) и быстрого их поиска — чтобы отмеченные бросались в глаза и были быстро доступны.



Поэтому в приложении цвет используется как пометка звёздочкой, но вместо жёлтой звезды можно выбрать звезду любого цвета:



В случае, когда в импортитуемом файле есть цвета, алгоритм вычисляет ближайший по схожести тона, Hue. Обратно в KeePass цвета экспортируются как светлый фон выбранного оттенка.

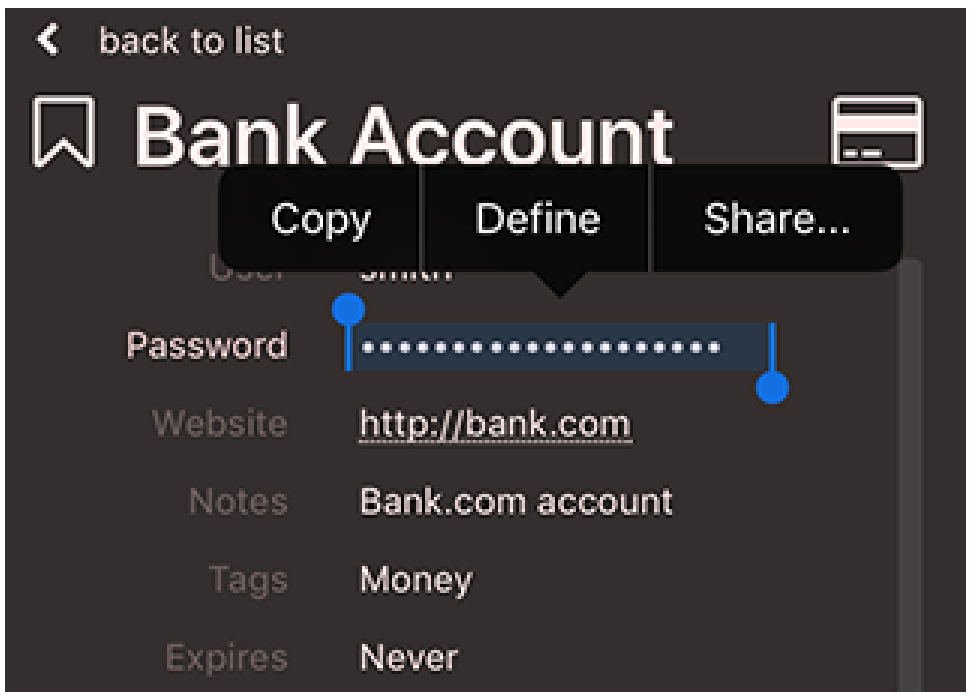
Dropbox

Приятно удивился, что для подключения к дропбоксу из SPA через dropbox-js теперь не надо встраивать «зашифрованный» secret key

в приложение, как это было раньше. Он не так и не научился отслеживать закрытие вкладки в popup-авторизации, но его легко научить (но надо патчить либу).

Копирование

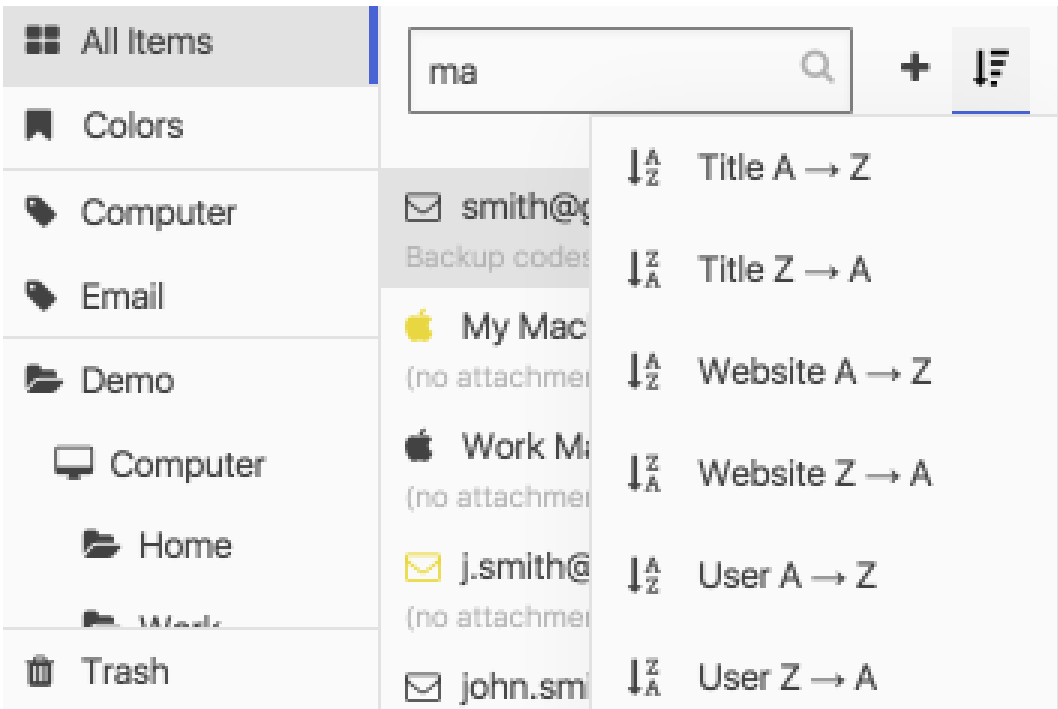
API копирования заработало в браузерах не так давно: в chrome оно появилось летом, в firefox — осенью этого года, в IE было раньше, но с вопросом пользователю. В сафари его до сих пор нет, а копировать текст как-то хочется. Особенно в мобильном сафари. Но если копировать там и нельзя, то можно показать пользователю баббл Copy, сделав прозрачный input. В экран ещё раз тыкнуть придётся, но всё-таки уже не так болезненно, как выделение и копирование:



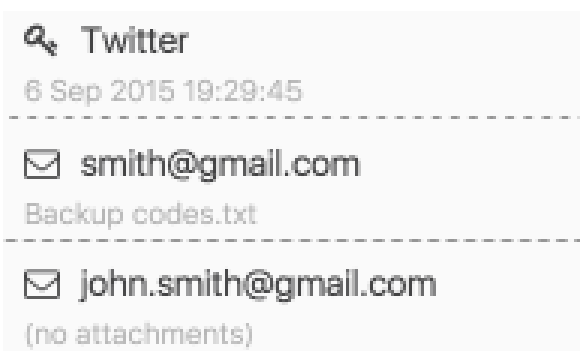
Поиск и корзина

Если человек открыл несколько файлов, скорее всего, он хочет,

чтобы поиск был по ним всем. Поэтому поиск, фильтры, сортировка и корзина сделаны общими. Фильтры расположены в порядке востребованности пользователем. Сначала All Items (он даже доступен по шорткату Ctrl/Cmd-A, т.к. чаще всего хочется показать всё и просто найти в поиске необходимое), потом цвета-фавориты, за ними теги, структура и уже совсем внизу мусорка:



Поиск умеет работать по всему: поля, теги, вложения. В списке записей, при сортировке по разным полям, на второй строке показывается поле, по которому сортировали:



Формат поддерживает файлы без корзины, однако в UI корзина

всё время есть. Это сделано затем, чтобы показать её отдельно от списка групп, одну на все файлы. В списке обычных групп корзине делать нечего, она должна быть особенной.

Название

Когда я рассказал о приложении на sourceforge в форуме KeePass, Dominik Reichl (разработчик KeePass) написал мне на почту, что получилось хорошо, предложил добавить приложение на страничку неофициальных клиентов, и попросил поменять название, что я и сделал.

О скучном

В приложении пока что встречаются баги, которые я постепенно вылавливаю. Используйте на свой страх и риск, делайте бэкапы. В крайнем случае есть экспорт в XML, где можно посмотреть, что пошло не так. Многое пока не сделано, но вообще идея приложения мне понравилась, я собираюсь его развивать и поддерживать.

Все компоненты приложения, как и оно само, доступны под MIT.

Ссылки

keeweb — веб-приложение
releases — десктоп-релиз
github — код
keeweb.info — сайт с фичами на одной страничке в виде картинок

kdbxweb — читалка kdbx для браузеров и node.js

kee_web — twitter проекта

Проголосовать:

+132

Поделиться:

Сохранить:

Комментарии (160)

Похожие публикации

[в закладки] Инструменты для тестирования JavaScript-проектов

8

ПЕРЕВОД

ru_vds • 19 февраля в 15:27

JavaScript ES6 — синтаксис оператора spread (…)

4

ПЕРЕВОД

IsaNesquik • 9 февраля в 14:27

JavaScript ES6: оператор расширения

18

ПЕРЕВОД

ru_vds • 9 февраля в 12:02

Популярное за сутки

Наташа — библиотека для извлечения структурированной информации из текстов на русском языке

14

alexkuku • вчера в 16:12

Unit-тестирование скриншотами: преодолеваем звуковой барьер. Расшифровка доклада

4

lahmatiy • вчера в 13:05

Люди не хотят чего-то действительно нового — они хотят привычное, но сделанное иначе

25

ПЕРЕВОД

Smileek • вчера в 10:32

Руководство по SEO JavaScript-сайтов. Часть 2. Проблемы, эксперименты и рекомендации

2

ПЕРЕВОД

ru_vds • вчера в 12:04

Как адаптировать игру на Unity под iPhone X к апрелю

0

P1CACHU • вчера в 16:13

Лучшее на Geektimes

Стивен Хокинг, автор «Краткой истории времени», умер на 77 году жизни — 33

HostingManager • вчера в 13:49

Обзор рынка моноколес 2018 — 70

lozga • вчера в 06:58

«Битва за Telegram»: 35 пользователей подали в суд на ФСБ — 40

alizar • вчера в 15:14

Стивен Хокинг и его работа — что дал ученый человечеству? — 8

marks • вчера в 14:46

Sunlike — светодиодный свет нового поколения — 17

AlexeyNadezhin • вчера в 20:32