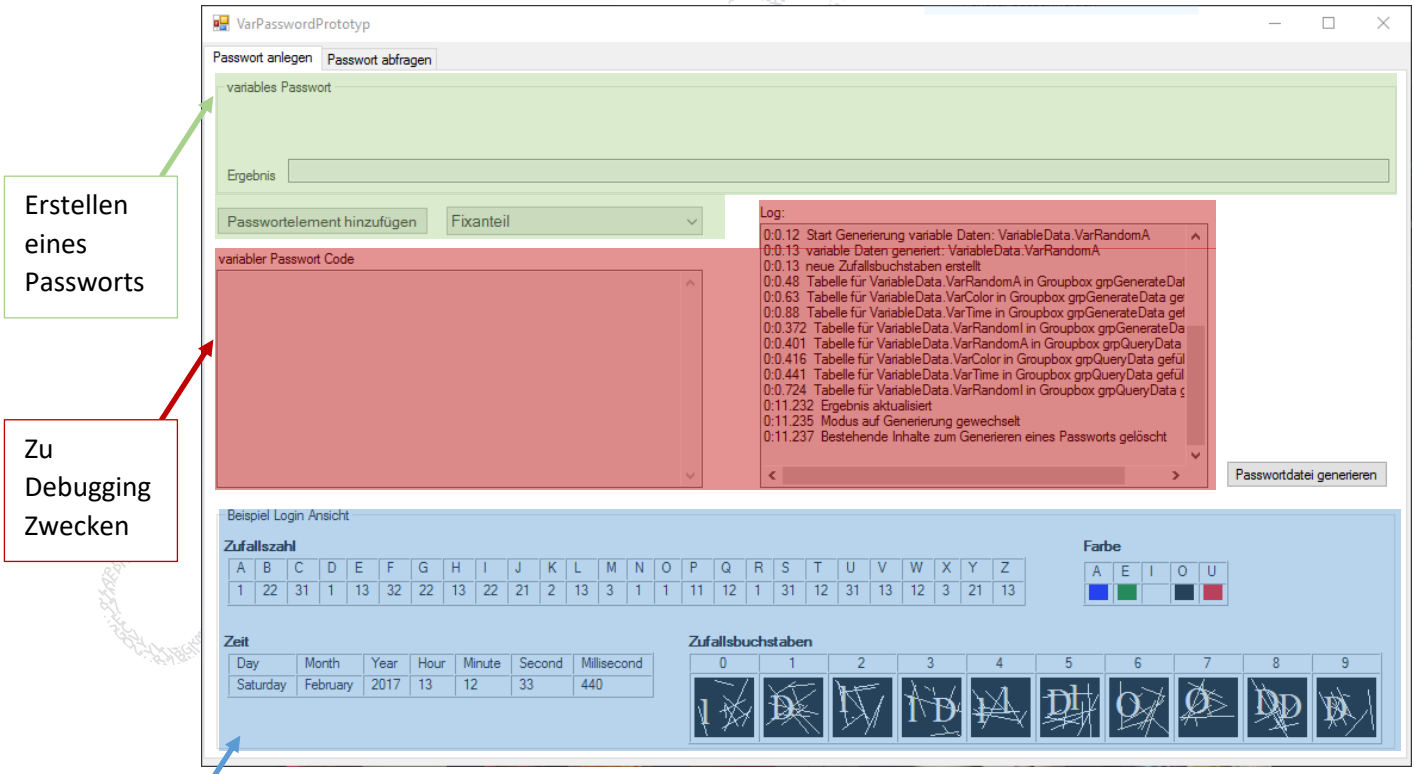
**Dynamische Passwörter gegen Brutforce, Keylogging und Pishing**

Wenn das Passwort genügend komplex gewählt wird, wird es zunehmend schwieriger aber dennoch nicht unmöglich um via Brutforce unrechtmäßig Zugang zu erlangen. Würde sich jedoch das Passwort nach jedem Login Versuch ändern oder gar dynamisch sein, sollte es doch unmöglich sein mit Brutforcetechniken weiterzukommen. Auch das Mitloggen der Tastenanschläge würde wohl im Sande verlaufen wenn sich der Zugangscode beim nächsten Login geändert hätte.

Eine wirkungsvolle Lösung gegen Pishingversuche wären oben genannte Vorschläge nicht aber auch dafür würde es meiner Meinung nach Möglichkeiten geben.

Nachfolgend möchte ich an Hand diverser Prototypen vorstellen wie oben genanntes zu erreichen wäre. Vorgestellte Möglichkeiten haben gemeinsam, dass der Server auf dem der Login erfolgen soll Zufallsdaten produziert anhand derer das Passwort ermittelt werden muss. Das Passwort wurde vorher mittels einer „dynamischen Passwortsprache“ erstellt.

## Erster Prototyp

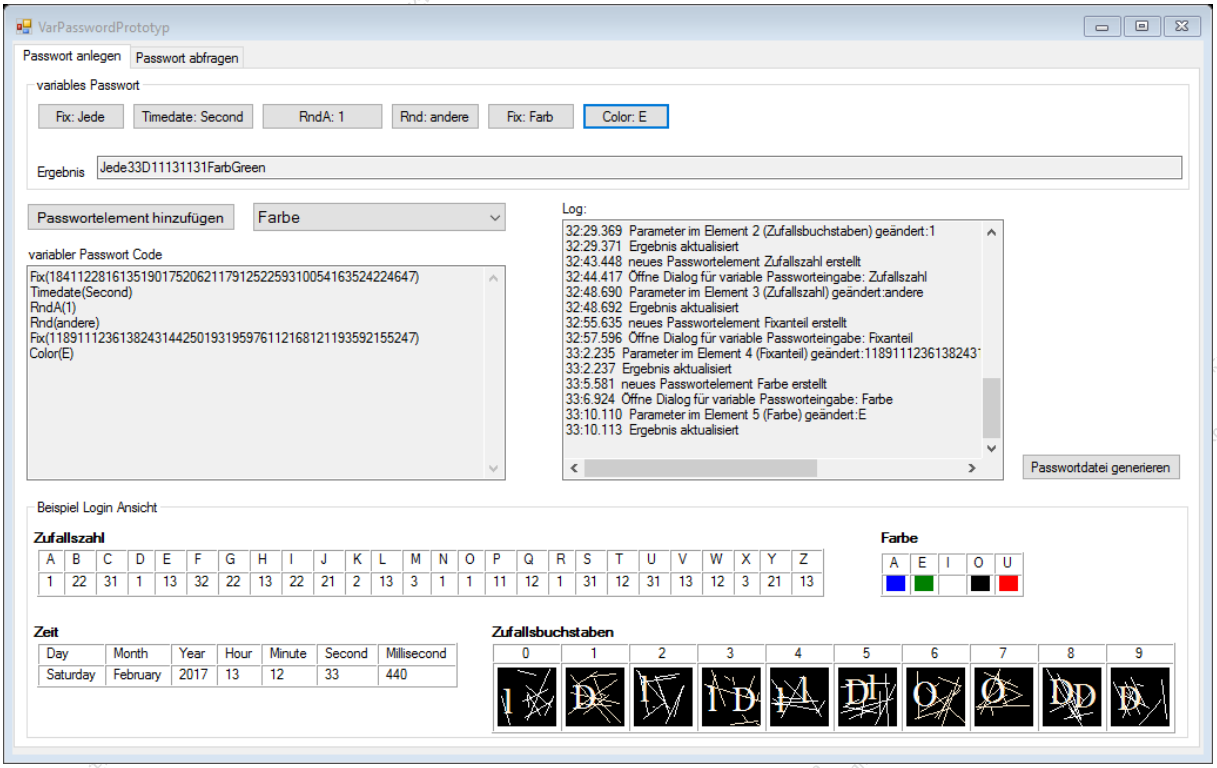


## Passwort generieren

Im oberen Teil kann nun ein Passwort angelegt werden indem in der Kombobox zwischen verschiedenen Passwortelementen ausgewählt wird. Als Beispiel nehme ich den Satz „Jede Sekunde eine andere Farbe“, wobei ich die diesen wie folgt aufteile:

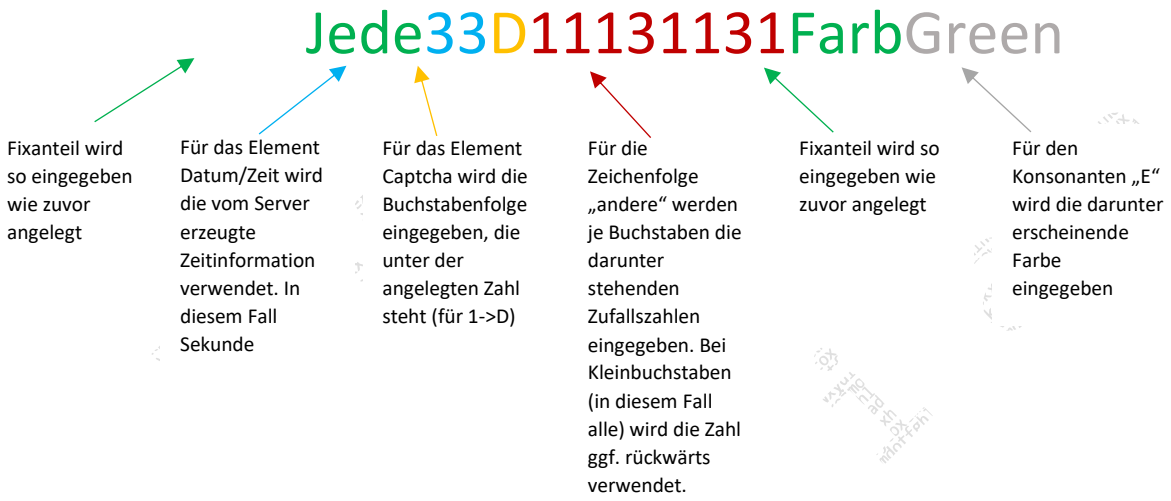
- „Jede“ → Fixanteil [Jede]
- „Sekunde“ → Zeit/Datum [Sekunde]
- „eine“ bzw. 1 → Zufallsbuchstaben (Captcha) [1]
- „andere“ → Zufallszahlen (Zuordnungstabelle) [andere]
- „Farb“ → Fixanteil [Farb]
- „E“ → Farbe [E]

Somit ergibt sich ein Ergebnis wie im folgenden Screenshot zu sehen.




Die einzelnen Elemente werden erstellt, indem man sie in der Kombobox auswählt und den Button „Passwortelement hinzufügen“ drückt. Anschließend muss das erstellte Element nochmals anklickt werden um die spezifischen Daten einzugeben oder zu wählen.

Im Feld „Ergebnis“ ist nun zu sehen welche Zeichenfolge beim Login eingegeben werden müsste wenn der Server unten angegebene Zufallsdaten erzeugt hätte. Zu beachten ist, dass das Element „Zufallszahlen (Zuordnungstabelle)“ bei Kleinbuchstaben die Zahl rückwärts interpretiert (andersherum wäre wohl leichter zu handeln gewesen). Im Detail wäre das nun folgende Zeichenfolge:




In der Textbox „variabler Passwort Code“ stehen die Daten wie sie bei Generierung einer Passwortdatei abgespeichert werden würden. Für die „dynamic password language“ habe ich hier folgende Elemente realisiert:

Fix(SHA1* hash): hiermit wird ein Fixanteil angezeigt, der wie zu sehen verschlüsselt gespeichert wird

Timedate(date parm): ein Timedate Element, das mit den Parametern Day, Month, Year, Hour, Minute, Second oder Millisecond abgespeichert werden kann

RndA(0-9): muss nachher als Buchstabenkombination, die unter Zufallsbuchstaben zu finden sind eingegeben werden

Rnd(A-Z|a-z): muss nachher als Zahlenkombination, die unter Zufallszahl zu finden sind eingegeben werden

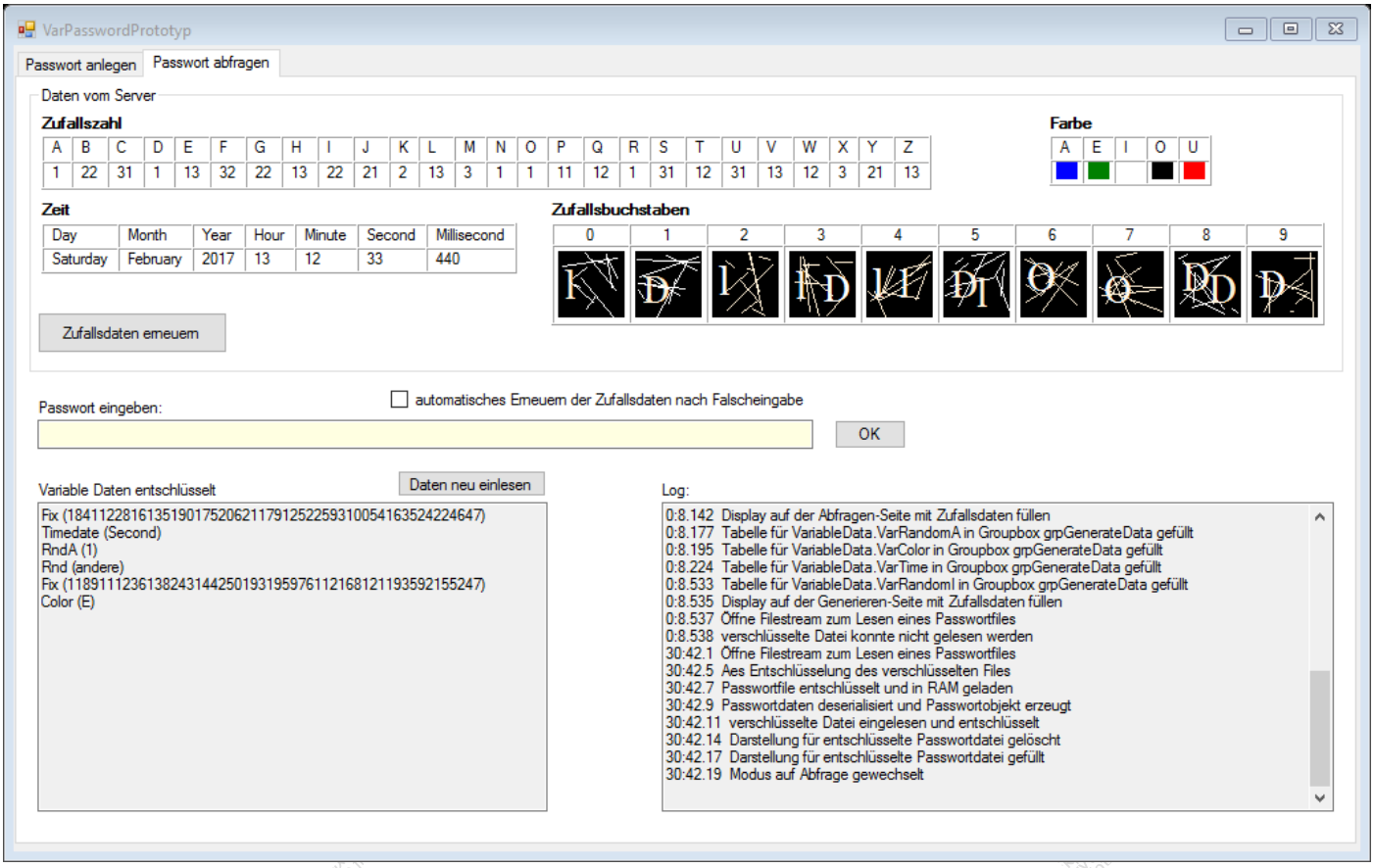
Color(A|E|I|O|U): Der angegebene Buchstabe muss bei Passworteingabe in Form einer zum Buchstaben zugewiesenen Farbe eingegeben werden (Blue, Green, White, Black oder Red)

Nun kann mit dem Button „Passwortdatei generieren“ exemplarisch eine Passwortdatei generiert werden. Im Programmverzeichnis werden zwei Dateien generiert, „rawPassword.txt“ und „encryptedPassword.bin“. Die erste der beiden dient nur zu Debugging-Zwecken. Bei dieser Version des Prototyps wird das erstellte Passwortobjekt serialisiert und abgespeichert warum die unverschlüsselte Datei nicht reinen Text enthält. Dadurch ist die Datei auch relativ groß, in diesem Fall etwa 800 Bytes. Die zweite Datei, die eigentlich später am Server abzuspeichern wäre, wird nochmals mit einem fixen Passwort AES verschlüsselt, was nicht unbedingt ideal ist, was aber im nächsten Prototyp besser gelöst ist. Noch besser wäre hier wahrscheinlich eine variable Lösung bei der der User selbst mit zur Verschlüsselung beitragen kann, dazu aber später mehr.

**natürlich sollte hier besser SHA2 bzw. SHA256 verwendet werden (siehe Bericht)*

## Passwort abfragen

Nachdem nun das Passwort generiert wurde kann zum Tab „Passwort abfragen“ gewechselt werden. Im Prinzip gibt es hier ähnliche Elemente. Der obere Teil „Daten vom Server“ würde dem Benutzer, der sich einloggen will präsentiert. Die unteren Textboxen dienen nur wieder zum Debuggen.



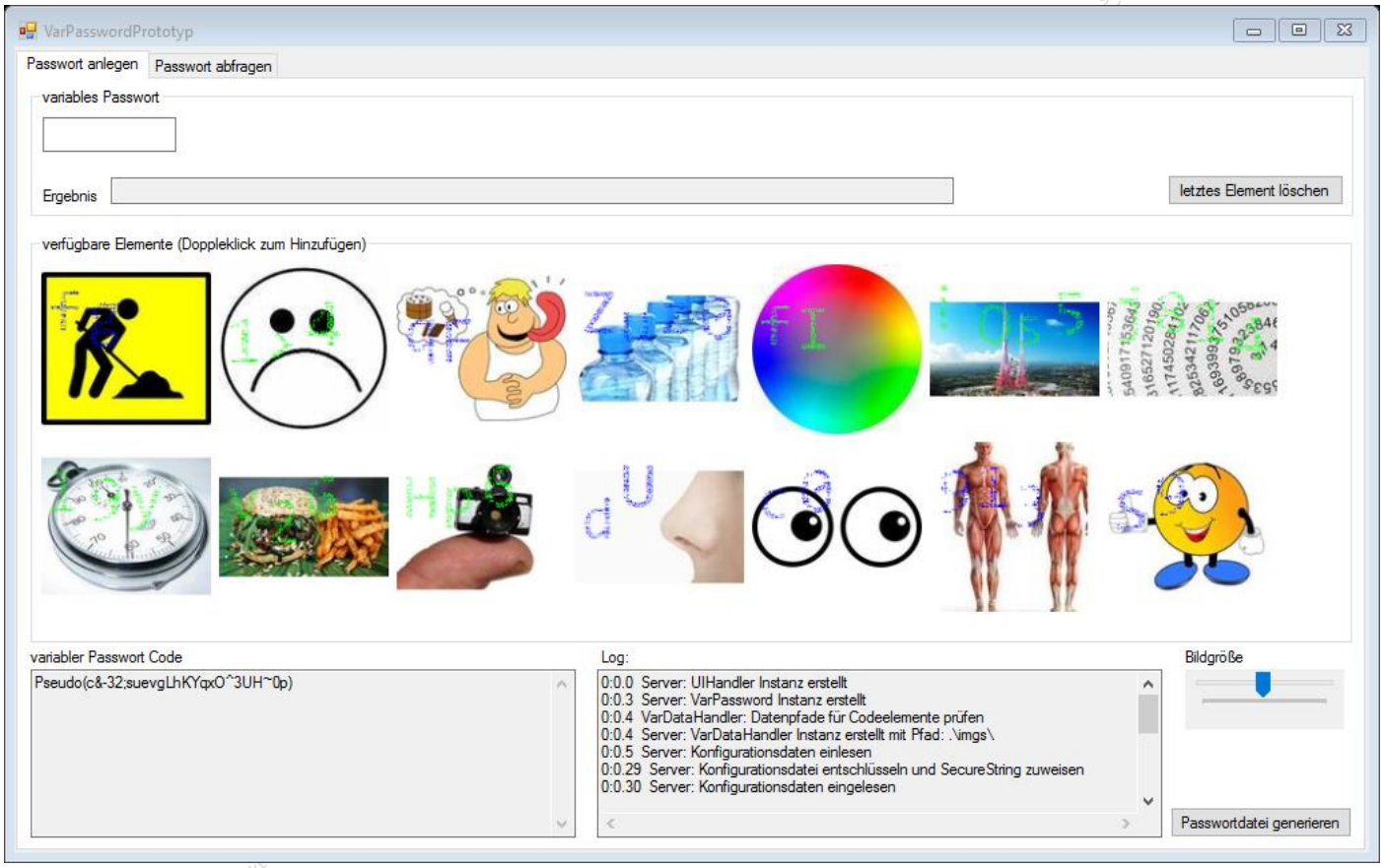

Im gelben Textfeld „Passwort eingeben:“ muss nun das vorher erstellte Passwort eingegeben werden (der Prototyp lädt die Datei „encryptedPassword.bin“ wenn vorhanden und entschlüsselt diese (wird in der linken unteren Textbox angezeigt). Da sich bis jetzt die Zufallsdaten vom fiktiven Server noch nicht geändert haben wäre im Prinzip die gleiche Zeichenfolge einzugeben, die vorher im Feld Ergebnis angezeigt wurde. Das realistischere Szenario wäre natürlich wenn die Daten erneuert wären (Button „Zufallsdaten erneuern“ drücken). Auch bei jeder Falscheingabe sollten sich die Daten ändern, wenn man ganz sicher gehen will (Checkbox „automatisches Erneuern der Zufallsdaten nach Falscheingabe“). Dies würde nun bedeuten, dass sich das Passwort ständig ändern würde. Dadurch, dass ich auch bei den Zufallszahlen keine große Varianz eingesetzt habe würde ich mich hier sogar sagen, dass mir beim Eingeben meines Passworts auf die Finger schauen kann ohne es danach nutzen zu können.

Natürlich gäbe es hier noch viel Verbesserungspotential oder Variationsmöglichkeiten. Nicht optimal ist die Sache mit den Farben, umgekehrt wäre es besser. Die Kleinbuchstaben sollten nicht rückwärts gelesen werden, eher die Großbuchstaben, da diese nicht so oft vorkommen. Auch die Captchas waren nur eine schnelle Eigenentwicklung und sind nicht sehr gut zu lesen. Man könnte noch Ersetzungen für alles Mögliche erstellen (Sonderzeichen, Formen, usw) oder für ganz paranoide z.B. Verschachtelungen wie z.B. RndA(Timedate(Second)). Diese Form des Passworts hat aber doch den Nachteil, dass es ziemlich umständlich zur Eingabe ist. Außerdem ist es auch nicht ganz einfach, sich Eselsbrücken zum Merken der Reihenfolge zu bauen.

Aufgrund dieser Nachteile und neuen Ideen kam ich zu einem zweiten Entwurf.

## Zweiter Prototyp

Das Prinzip ist ähnlich, es gibt eine „dynamic password language“, die selbst jedoch mehr oder weniger vom Betreiber des Servers festgelegt wird. Ich würde sogar so weit gehen zu sagen, dass sich der User, der den jeweiligen Dienst nutzt seine eigene „password language“ anlegen kann (später mehr dazu, auch bzgl. Pishing). Es wird eine Art Passwortcode angelegt und später werden vom Server wieder Zufallsdaten erzeugt, die je nach angelegter Passwortdatei richtig interpretiert werden müssen. Inspiriert von nachfolgendem Konzept wurde ich unter anderem vom Brettspiel Concept (https://www.youtube.com/watch?v=-92ZBTzu5tI).




Der obere Teil „variables Passwort“ dient wie beim vorherigen Programm schon dazu, das erstellte Passwort anzuzeigen bzw. zu bearbeiten. In der bereits angezeigten Textbox in diesem Teil kann wie vorher auch ein Fixanteil eingegeben werden. Zwei Fixanteile nacheinander machen keinen Sinn, weshalb dies auch nicht möglich ist, d.h. vor oder nach einem Fixanteil muss immer ein dynamisches Element stehen. Diese dynamischen Elemente werden mit Hilfe der angezeigten Bilder im Bereich „verfügbare Elemente (Doppelklick zum Hinzufügen)“ erzeugt. Es können beliebig viele dynamische Elemente hintereinander folgen. Die angezeigten Bilder entsprechen auch den später am Login Server angezeigten Zufallsdaten. Jedes Bild entspricht einer bestimmten Kategorie. Als Beispiel für obige Anzeige:


work: alles was mit Arbeit/Tätigkeiten zu tun hat


negative: alles was schlecht/traurig ist


taste: alles was mit schmecken zu tun hat


large: große Sachen

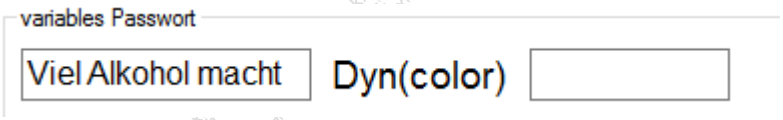

drink: alles was mit Trinken zu tun hat

color: alles mit Farben

Natürlich kann jeder die Symbole etwas anders interpretieren was allerdings zu sehr phantasievollen Ergebnissen führt (wie im Spiel Concept). Das weitere Vorgehen ist wohl am ehesten an einem Beispiel zu erklären. Man denkt sich einen Passwortsatz aus, z.B.:

Viel Alkohol macht blau

Im ersten Schritt könnte man z.B. „Viel Alkohol macht“ als Fixanteil anlegen und „blau“ mit einem dynamischen Element Farbe (blau sollte mit Farbe in Verbindung gebracht werden können) ersetzen. Dazu wird in der bereits existierenden Textbox oben der Fixanteil eingegeben und anschließend auf das Symbol, das die Farbe repräsentiert doppelgeklickt. Das Ergebnis sollte ungefähr so aussehen:




Zu sehen ist hier, dass nach einem dynamischen Element immer vorsorglich schon mal wieder eine Textbox für ein Fixelement erstellt wird, welches jedoch immer weiter rückt falls weitere dynamische Elemente folgen. Was hat sich noch getan?

In der untersten linken Textbox „variabler Passwort Code“ kann man sehen was in die Passwortdatei geschrieben würde. Im Wesentlichen gibt es hier nur noch drei verschiedene Elemente:

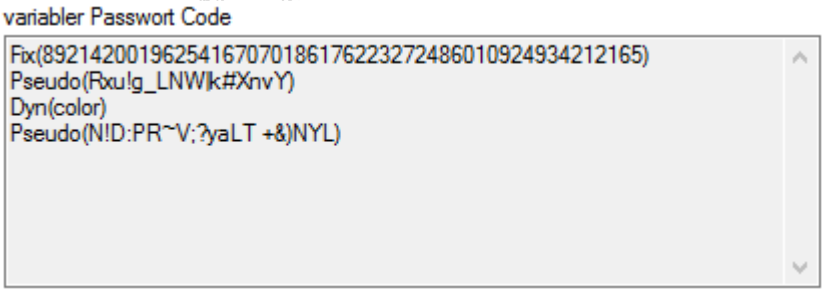
Fix(SHA1*): Das Schlüsselwort „Fix“ steht für einen Fixanteil und wird mit dem Hash als Parameter abgespeichert

Dyn(varElement): Das Schlüsselwort „Dyn“ steht für jegliches dynamisches Element welches als Parameter das spezielle Element kennzeichnet.

Pseudo(random): Nach jeder Änderung des variablen Passworts wird zufällig mind. ein Pseudoelement min zufälligen Zeichen generiert, so dass sich zwei gleich erzeugte Passwortdateien nie in ihrem Bytecode gleichen.

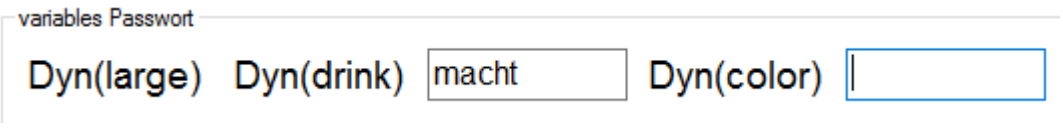
**natürlich sollte hier besser SHA2 bzw. SHA256 verwendet werden (siehe Bericht)*

Für das oben generierte Passwort sieht das bei mir in etwa folgendermaßen aus:




Im oberen Teil in der Textbox „Ergebnis“ wäre nun die Zeichenfolge zu sehen, die mit den angezeigten Bildern einzugeben wäre. Wie vielleicht schon aufgefallen ist, ist jedes Bild mit einer Zeichenfolge überlagert. Diese Zeichenfolge muss bei Eingabe des Passworts anstelle des dynamischen Elements eingegeben werden. Dazu später mehr.

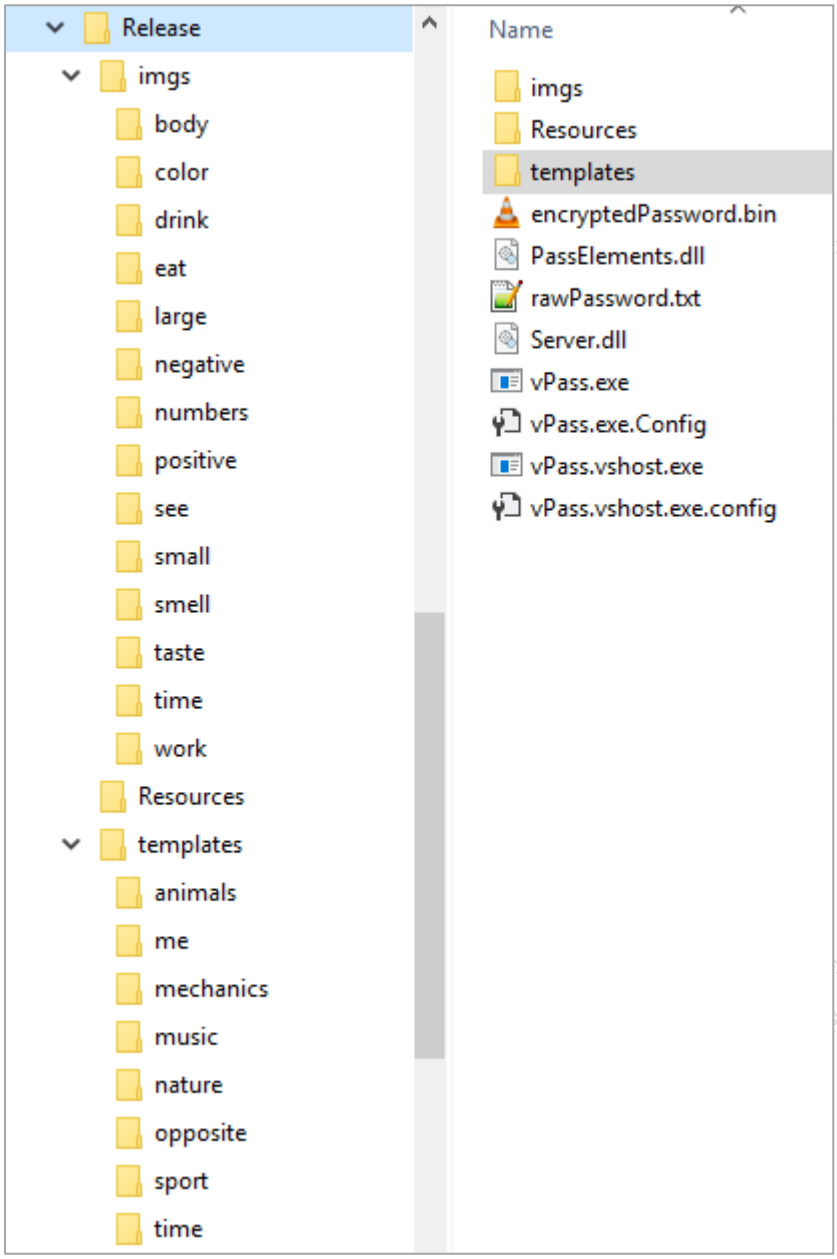
Wer sich nun schon etwas an das Konzept gewöhnt hat wird aber bestimmt mehr dynamische Elemente in sein Passwort einbauen. So könnte das Wort „Alkohol“ durch das dynamische Element oder Symbol des Trinkens ersetzt werden. Das Wort „Viel“ könnte auch durch das Symbol für Großes substituiert werden. So dass sich folgendes ergäbe:



Ganz Mutige würden wohl auch noch das work-Element für das Wort „macht“ einsetzen, im Sinne von „etwas machen“ oder „tun“. Da aber dann nur noch dynamische Elemente vorhanden sind wäre das meiner Meinung nach nicht gut. Es sollten immer beide Typen vorhanden sein und wäre auch in einer endgültigen Version zwingend zu erfüllen.

## Eigene dynamische Elemente erstellen

Was jedoch wenn so gar nichts passen will oder ein Betreiber seine eigenen Bilder und Kategorien möchte?




Kein Problem. Im Programmverzeichnis gibt es einen Ordner „imgs“, der die ganzen Bilder enthält. Nicht nur die Bilder sondern auch die verfügbaren Kategorien werden hier festgelegt. Die Anzahl der Kategorien ist nicht begrenzt, jedoch wird es mit zunehmender Anzahl immer schwieriger sprechende, der Kategorie zuordenbare Bilder zu finden (evtl. können das andere Leute sehr viel besser als ich). Wenn nun jemand z.B. Tiere als zusätzliche Kategorie einführen/bereitstellen möchte dann wird im imgs-Ordner einfach ein Unterordner mit dem Namen der Kategorie (tiere oder animals) angelegt. Ich habe bereits für ein paar zusätzliche Kategorien im Ordner templates Bilder zum Testen gesammelt. Diese müssen zur Verwendung wie erwähnt in den imgs-Ordner kopiert werden.

Beim Starten der Anwendung wird der Ordner imgs gescannt und die vorhandenen Kategorien als Parameter für das Dyn-Passwortelement zur Verfügung gestellt.

Vorstellbar wäre hier z.B., dass sich ein User seine eigenen Kategorien mit eigenen Bildern erstellen könnte, die bei Eingabe eines Usernamens erscheinen. Dies hätte einen bestimmten Wiedererkennungswert zur Folge der nur mühsam nachgeahmt werden könnte (Stichwort Pishing).

Für eigene Bilder ist nur zu beachten, dass die Größe 130x130 Pixel sein muss. Um die Bilder auf diese Größe zu bringen und auf eine weiße Leinwand zu bringen habe ich das Script AdaptSize.xbs für den Bildbetrachter XnView abgespeichert. Unter Werkzeuge-Stapelverarbeitung muss hierfür der Ordner mit den Originalbildern hinzugefügt und im Bereich „Skript“ selbiges aktiviert und geladen werden. Starten.-Fertig.

## Dynamisches Passwort speichern

Nach man sich nun ein originelles Passwort ausgedacht und angelegt hat wird dieses über den Button „Passwortdatei generieren“ erzeugt und gespeichert. Es werden wie bei der Vorgängerversion zwei Dateien, „rawPassword.txt“ und „encryptedPassword.bin“ im Programmverzeichnis abgelegt. Jedoch ist es bei dieser Version

so, dass keine Serialisierung eines Objekts stattfindet. Der Passwortcode wird in Klartext (zumindest für Debugging Zwecke) gespeichert, wie in der unverschlüsselten Datei „rawPassword.txt“ zu sehen ist. Dies spart auch Speicherplatz wie an den etwa 150 Bytes zu sehen ist. In dieser Datei steht derselbe Inhalt wie er zuvor in der Textbox „variabler Passwort Code“ zu sehen war. Die verschlüsselte Passwortdatei wird wieder AES verschlüsselt, jedoch nicht mit einem Passwort, das wie in der Vorgängerversion im Code abgelegt war. Hier wird mit einem Schlüssel codiert, der in der XML-Datei „vPass.exe.Config“ abgelegt war. Hier verwende ich Microsofts DPAPI (Data Protection API), d.h. dass vor dem ersten Starten der Anwendung in der genannten XML Datei ein Passwort zu hinterlegen ist.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <configSections>
        <sectionGroup name="applicationSettings" type=
        "System.Configuration.ApplicationSettingsGroup, System, Version=4.0.0.0,
        Culture=neutral, PublicKeyToken=b77a5c561934e089" >
            <section name="protectedSection" type="System.Configuration.ClientSettingsSection,
            System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
        </sectionGroup>
    </configSections>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2"/>
    </startup>
    <applicationSettings>
        <protectedSection>
          <setting name="Password" serializeAs="String">
          <value>Hier ein 32 Stelliges Passwort !</value>
          </setting>
        </protectedSection>
    </applicationSettings>
</configuration>
```

Beim ersten Start der Anwendung prüft diese, ob das Passwort noch in Klartext vorhanden ist oder schon verschlüsselt abgelegt wurde. Ist das Passwort noch in Klartext vorhanden, wird dieses verschlüsselt und folgendermaßen abgelegt:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <configSections>
        <sectionGroup name="applicationSettings" type="System.Configuration.ApplicationSettingsGroup,
        System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >
            <section name="protectedSection" type="System.Configuration.ClientSettingsSection,
            System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" allowLocation=
            "true" allowDefinition="Everywhere" allowExeDefinition="MachineToApplication"
            overrideModeDefault="Allow" restartOnExternalChanges="true" requirePermission="true" />
        </sectionGroup>
    </configSections>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2"/>
    </startup>
    <applicationSettings>
        <protectedSection configProtectionProvider="DataProtectionConfigurationProvider">
            <EncryptedData>
                <CipherData>
                    <CipherValue>
                    AQAAANCMnd8BFdERjHoAwE/Cl+sBAAAAvaaQV++kAk2opluEevFKrQQAAAACAAAAAAAQZgAAAAEAACAAAAA
                    3JxjPKru4J6HkmdLmN4G5HwIfhX9ahBfmldIdnN67fwAAAAAOgAAAAIAACAAAADfgTbVXJ8S4yNvyvWuiA
                    KpOYqRcDjkMtnG45urZBL3iVABAAAyS8wbtYS6ArFpmKlZUEpHSLadQdEL1OwoLXUg6mYAEWVtzC9gXpnR8
                    BIiPBEOAPU4U3J2uAeixJWrb41fN8FTnk++m0qz6pR4WlBcouekUKOTQSvJeKV7SSEHnL1li+v3d2w+J8JZ
                    M+kuxifNH30N+D5DeOg37pdrDk+TFdaLRSBT52IHZ4U3ViNazPmYfjGh+T20djtPq9PunJOh7EgjPJMr8Lj
                    wdkvIRiixCCHA3Q6BDOV7XsksdJJ6B3eqdVDUnCzmNt8/vzMgrqhBuF/Y0ePO9Z9VtPghka8Z8p9ZAxTs+G
                    KeWNs1go84X2NSAEppnq/l2NxHdz7qLtsiLH4kO4gP+YfQ86aqYqKkYwyJhb/wt0GBrEya6TdG9C/6HCQm6
                    50vMAPIrO/YXx34aviKsvqVKbAK71lgsLH9UVrM0rN7AwYsxV5sPyiZAXh8OKVAAAAA8R5BCRsxkNPrAJf2
                    RIAgT2h6d/B6CNdV5SlyQNzliy5+xpafNi5XxiZBP6JJbchd9pwuXdUkZT2kS7pGeWv4xg==
                    </CipherValue>
                </CipherData>
            </EncryptedData>
        </protectedSection>
    </applicationSettings>
</configuration>
```

Von da an ist die Datei bzw. der Schlüssel für die Decodierung der dynamischen Passwortdatei nur noch auf dem PC zu entschlüsseln auf dem sie/er erzeugt wurde. DPAPI verwendet ein Geheimnis, das mit dem verwendeten System gekoppelt ist.

Um dies zu testen ist im templates-Ordner eine XML Datei „vPass.exe.config“ mit noch nicht verschlüsselten Geheimnis abgelegt. Diese Datei kann in das Programmverzeichnis kopiert und an entsprechender Stelle ein geheimer Code (bei „Hier ein 32 Stelliges Passwort !“) eingegeben werden.

## Abfrage des erstellten dynamischen Passworts

Beim Wechsel in den Tab „Passwort abfragen“ ist es wieder ähnlich der Vorgängerversion. Die Bilder mit den zugehörigen überlagerten Zeichenfolgen sind die Zufallsdaten, die vom fiktiven Server erzeugt werden. Dabei werden auch die Positionen der Bilder immer zufällig angeordnet. Da auch die Zeichenfolgen in ihren Grenzen zufällig angeordnet werden kann es vorkommen, dass diese oft schwer zu lesen sind. Auch der Unterschied zwischen verschiedenen Buchstaben oder Groß- und Kleinschreibung ist oftmals schwer zu erkennen (I wie Igel oder l wie lang, O oder o, V oder v, W oder w….). Dazu habe ich mir folgendes einfallen lassen und eigens eine Schriftart erstellt:

Bleibt der Mauszeiger über dem Bild stehen, wird ein Ausschnitt des Bildes vergrößert und mit größeren Buchstaben gefüllt. Diese können wieder etwas anders aussehen wie die originalen aber es sind dieselben Zeichen. Bei Schwierigkeiten der Unterscheidung hilft die Schriftart selbst. Großbuchstaben bestehen wiederum aus Großbuchstaben, Kleinbuchstaben aus Kleinbuchstaben und Zahlen/Zeichen aus Zahlen/Zeichen. So ist es ziemlich einfach zu erkennen ob es sich um Groß- oder Kleinschreibung, ob es sich um eine Zahl/Zeichen oder um einen Buchstaben handelt. Die Schriftart „LetterFont.ttf“ ist im Ordner Resources abgelegt. Wird diese aus dem Ordner entfernt wird eine Standardschriftart verwendet die etwas besser zu lesen ist. Warum ich diese Schriftart selbst erstellt habe? Weil ich es testen wollte, ob man das ständige Problem, dass man die Buchstaben in Captchas nicht identifizieren kann so lösen könnte. Dies geht aber tatsächlich nur wenn man diese dann auch als Vergrößerung oder gleich sehr groß darstellt. Einen anderen Punkt, den ich damit erreichen wollte ist, dass evtl. Systeme, die DeepLearning einsetzen Schwierigkeiten haben die Buchstaben zu lesen. Dies scheitert aber in diesem Beispiel wohl schon an der geringen Auflösung der Bilder. Somit kann ein „Atomzeichen“ aus denen die Buchstaben bestehen gar nicht erkannt werden. Jedoch habe ich aber leider auch sehr wenig Wissen und/oder Erfahrung in diesem Bereich.

Ich habe nun also ein Passwort für meinen Merksatz „viel Alkohol macht blau“ mit folgenden Elementen angelegt:

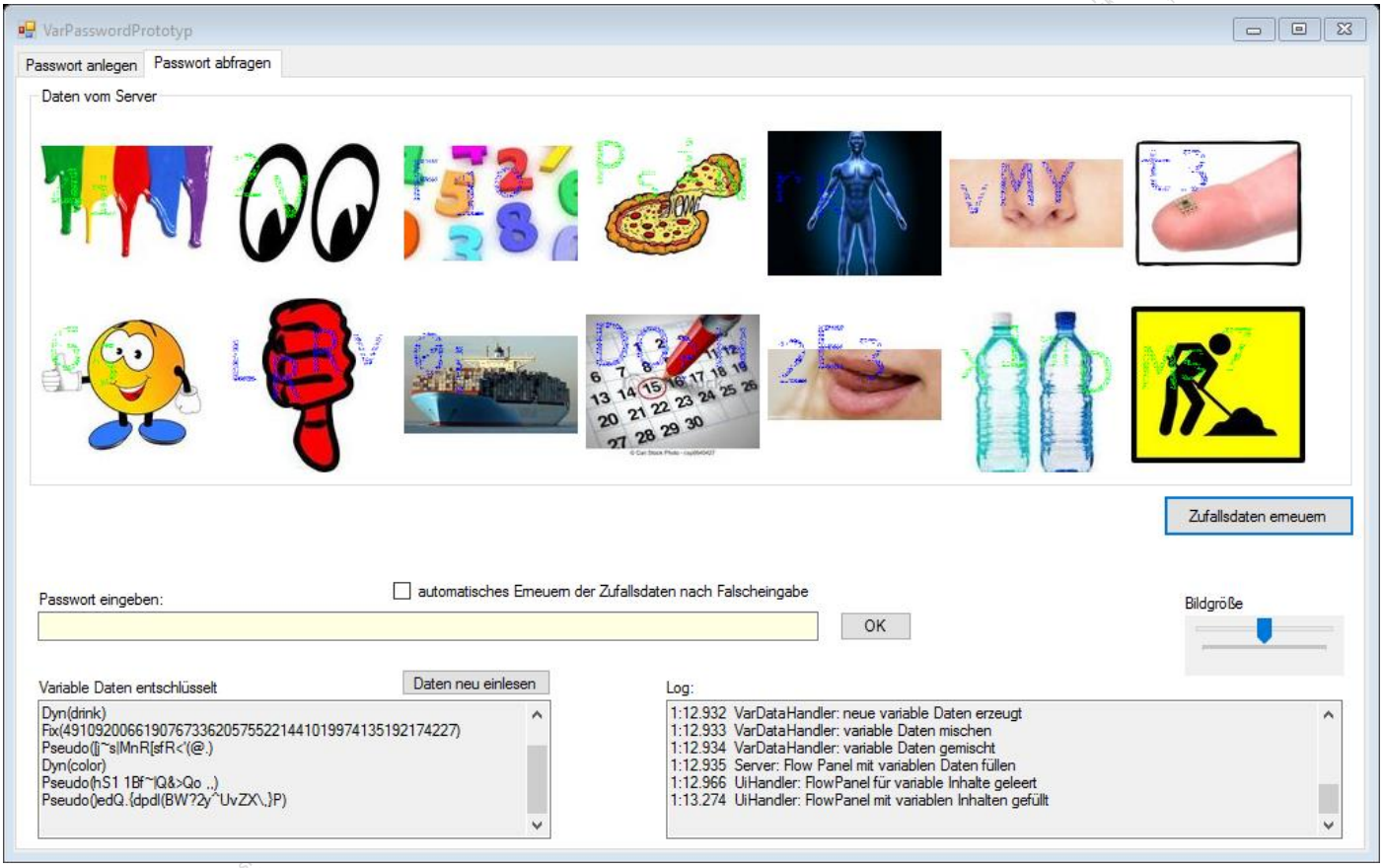
„viel“ -> Fixanteil

„Alkohol“ -> Substitution durch drink-Kategorie

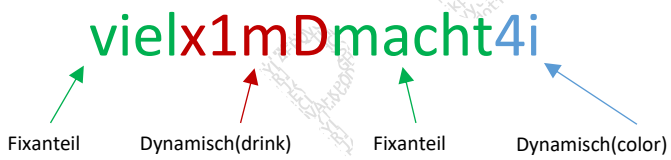
„macht“ -> Fixanteil

„blau“ -> Substitution durch color-Kategorie

Im Reiter „Passwort abfragen“ sehe ich folgende Bilder:




Mein Passwort muss ich also wie folgt eingeben:




Soviel zu meinen Ideen bezüglich der Software. Leider habe ich keine Zeit um eine Lösung für das Web zu programmieren, da ich dort noch weniger Erfahrung habe. Ich hoffe ich konnte mit meinen Prototypen das Prinzip aufzeigen und auch etwas Begeisterung wecken. Für mich bleibt noch ein kleiner Nachteil bei dem Ganzen, nämlich die synchrone AES Verschlüsselung der Passwortdatei auf dem Server. Hierfür würde mir noch eine Lösung einfallen, hatte jedoch keine Zeit mehr diese umzusetzen.

Zusätzliche Ideen

1. Um die Verschlüsselung der Passwortdatei auf dem Server sicherer zu machen könnte man z.B. ein Sonderzeichen definieren, das im Fixanteil einen speziellen Bereich kennzeichnet. Angenommen das definierte Sonderzeichen wäre ein „°“-Zeichen (Grad). Um das Beispiel von vorhin wieder aufzugreifen könnte man dann z.B. den zweiten Fixanteil „macht“ so schreiben „m°@ch°t“. Alles was zwischen diesen speziellen Sonderzeichen steht wird beim Speichern des Passworts nun dazu verwendet um einen Teil des bereits vom Betreiber definierten Schlüssels (der in der XML-Datei steht) zu ersetzen.

2. Bilder wie in diesem Programm oder in den nachgeschalteten Bilderrätseln des neuen reCaptcha sollten meiner Meinung nach anders umgesetzt werden. Sollte sich DeepLearning weiter entwickeln (wovon ich stark ausgehe) ist es doch nur wieder eine Frage der Zeit um Aufgaben von einem Computer zu lösen wie „Wähle alle Bilder, die Geschäfte von vorne zeigen aus“ oder „Wähle alle Bilder mit Blumen aus“. Ich denke, es wäre besser nicht nach Substantiven zu fragen sondern eher nach Verben oder Adjektiven und evtl. auch andersherum, so das man nur ein Bild braucht. Beispiel: Es wird ein Bild abgebildet und verschiedene Worte angeboten werden, von denen die richtigen gewählt werden müssen. Das Bild zeigt ein lachendes Kind mit wehenden Haaren, das auf einer Wiese steht; angebotene Wortauswahl: lustig, traurig, böse, fliegend, schwimmend - im nächsten Fall z.B. gleiches Bild mit den Worten: jung, alt, windig, windstill, drohend. Bei angebotenen Worten wäre darauf zu achten, dass sich nicht ein Wort zu sehr hervorhebt, da sonst automatisch leichter ausfindig zu machen.

Zum Schluss möchte ich noch darauf hinweisen, dass ich keine der angebotenen Prototypen bis ins kleinste Detail getestet habe. Ebenfalls habe ich nicht sehr viel Wert auf Fehlerbehandlung gelegt, d.h. es kann durchaus zu Situationen kommen in dem die Anwendung einfach abstürzt. Bekannt ist mir jedoch nur der Fall, bei dem der bereits verschlüsselte Schlüssel in der XML Datei geändert wird.