Product - Backlog für M1 Programming (Übung 2)
Social Media Bildbearbeitungstool SOMPE ;)

Mathias Buß (s521064) - Patrice Schwarz (s521409)

Gliederung:

1 Backlog

Punkte: 1 2 3 5 8 13 21 34 ... ↺ == geplante Zeiteinheit überschritten
Punkte werden pro Story geschätzt.
Urmeter: 3 Punkte

- **Story: 13** **Fertig**
  als Nutzer kann ich mein Bild bei Facebook hochladen, um es dort zu archivieren
- Tasks:
  In progress
  Done:
  Recherche:
  -Wie erfolgt der Upload, d.h. welche Daten werden benötigt?
  -welches Format hat das Bild, gibt es Beschränkungen?
  -Einfügen eines Buttons, mit dem per Klick ein Bild auf Facebook veröffentlicht werden kann
  -dabei werden zunächst die vorhandenen Alben ermittelt
  -der Nutzer kann sich anschließend ein Album auswählen, in welches das

Bild eingefügt werden soll
-klickt der Nutzer dann den Upload-Button, wird das Bild hochgeladen
-der Upload-Prozess wird in einer Progress-Bar dargestellt
-nach Ende des Uploads erhält der Nutzer eine Meldung über den Erfolg oder, im Falle eines Fehlers, eine entsprechende Meldung

- **Story: 21 Punkte** **Fertig**
  als Nutzer kann ich verschiedene Filter auf mein Bild anwenden, um es selbst anzupassen
- Tasks:
  In progress
  Done:
    Recherche:
    -Wie kann auf Bilddaten zugegriffen werden?
    -Wie können Bilddaten einfach und performant auf Pixelebene verändert/manipuliert werden?
  -Erstellung von min. 5 verschiedenen Filter Klassen zum bearbeiten von Bildern
  -Aufbau einer MVC ähnlichen Struktur (für das gesamte Programm) zu besseren Trennung von Programmlogik, Ein/ausgabe
  -Gebrauch des Singelton Pattern für einmalige Objekte (z.b. Haupt-GUI-Form, eigene Image Object Klassen)
  -Filterberechnungen durch optionales Multithreading (einstellbar über zentrale config)
  -Koordinierung der Filter Threads durch unabhängigen extra Thread -> rechtzeitige Bildfreigabe ↺
  -Regelung der Threadanzahl und anderer Konstanten über zentrale Enum Klassen
  -Logik zur Zwischenspeicherung der manipulierten Bilder in eigener Klasse
  ⇒ Zwischenschritte zum zurücksetzen einzelner Änderungen durch Filter
  -Speicherung der eigentlichen Zwischenbilder und relevanter Daten in eigener Image Klasse

- **Story: 5**
  als Nutzer kann ich verschiedene statistische Auswertungen einsehen
- Tasks:
- In progress

- Done:

- **Story: 2** **Fertig**
  als Nutzer kann ich mein verändertes Bild speichern
- Tasks:
  In progress
  Done:
  -der Nutzer kann sein Bild durch einen Klick auf "Bild Speichern" im Menü auf der Festplatte speichern

- **Story: 8**
  als Nutzer kann ich ein Bild über meine Webcam erzeugen
- Tasks:
- In progress
- Done:

- **Story: 34**
  als Nutzer kann ich aus verschiedenen Voreinstellungen zur Bildbearbeitung wählen
- Tasks:
- In progress
- Done:

2 Topic Report

2.1 A2: .NET Framework - use of Existing APIs

Wir haben in unserer Übung mehrere Apis genutzt, darunter die von C# vorgegebene System.Windows.Form-API sowie unter Zuhilfenahme des C#-Facebook-SDKs (http://facebooksdk.codeplex.com/documentation) die Facebook-API.
Zu Beginn unserer Arbeit haben wir uns zunächst überlegt, was wir gestalten wollten und haben uns während der Recherche dazu entschieden, zur Gestaltung unserer Oberfläche auf die System-Windows-API zurückzugreifen.
Da wir bereits etwas Erfahrung im Umgang mit der Facebook-API hatten, wollten wir auch diese in unser Projekt einbauen. Hinsichtlich der Facebook-API ging es zunächst darum, die vorhandenen PHP-Funktionen für C# nutzbar zu machen. Im Zuge der Recherche dazu fanden wir das oben genannte SDK in C#, welches bereits viele der von Facebook in PHP angebotenen API-Schnittstellen für C# verwendbar macht.
Anschließend ging es darum, die im Backlog erstellten Anforderungen in unsere Übung einzubauen. Der erste Teil ging um das Einloggen bei Facebook und der dazu notwendigen Autorisierung des Nutzers. Für diesen Teil war die API sehr hilfreich und der Login-Vorgang konnte wie geplant relativ problemlos ins Programm integriert werden. Dazu war insbesondere auf die Berechtigungen zu achten, die der Nutzer dem Programm hinsichtlich des Zugriffs auf seine Nutzerdaten gibt. In unserem Programm erbitten wir nur Zugriff auf die Standard-Daten sowie Zugriff auf Fotos und vorhandene Alben.
Im nächsten Abschnitt ging es darum, die bei Facebook vorhandenen Foto-Alben des Nutzers zu ermitteln und in eine ComboBox innerhalb unseres Programmes als Auswahlmöglichkeit zu übertragen. Auch hier bot die Facebook-API, deren genaue Beschreibung man sich unter https://developers.facebook.com/docs/reference/api/ anschauen kann, eine gute Unterstützung. Unter Nutzung des SDKs erstellt man

dazu ein sogenanntes FacebookMediaObject. Diesem werden als Wert die Art des Inhalts der Datei sowie die URL der Datei übergeben. Anschließen wird der Upload gestartet. Der Nutzer muss dann noch auf Facebook den Upload des Bildes bestätigen, anschließend ist das Bild im zuvor gewählten Ordner verfügbar.
Der Upload an sich wird grafisch in einer unterhalb des Buttons befindlichen Progress-Bar dargestellt. Sobald der Upload erfolgreich beendet wurde, erfolgt eine Meldung an den Nutzer, bei einem Fehler erscheint die entsprechende Fehlermeldung.

2.2 B7: Nebenläufigkeit

Multithreadingfähig sind momentan (Stand 21.12.2011) alle Filterklassen des Programms, die dazu dienen Bildmanipulationen vorzunehmen. Über eine zentrale Konfig Klasse kann die Anzahl der Threads eingestellt werden, die sich um die Bearbeitung eines Bildes kümmern sollen. Erstellt werden die Threads in einer eigenen Klasse ImageManipulator, die dazu dient Aufrufe vom Controller entgegen zu nehmen und an die richtigen Filterklassen weiter zu delegieren. Dazu wird in der Klasse jedes zu bearbeitende Bild für die einzelnen Threads aufgeteilt. Bei 2 Threads würde das Bild entsprechend der Hälfte der Höhe geteilt. Das Byte Array mit den Bilddaten wird komplett übergeben. Jede Filterklasse, aufgerufen durch einen Thread, weiß anhand der übergebenen Parameter start- und stopindex, welcher Teil des Arrays zu berechnen ist. Durch einen Pointer auf die Bilddaten hat jede Filterklasse Zugriff auf das zu bearbeitende Bild.
Da jeder Thread nur seinen eigenen Speicherbereich des gemeinsamen Arrays verändert, ist es nicht nötig das Byte Array zu synchronisieren. Allerdings muss während der Berechnungen das eigentliche Bild der Visual Studio PictureBox gelocked werden. Ein gleichzeitiger Zugriff löst andernfalls eine Exception aus. Bevor ein verändertes Bild auf dem Screen neu ausgegeben wird, muss sichergestellt sein, dass alle Threads ihre Berechnungen erledigt haben, da es ansonsten vorkommen kann,

dass einzelne Bildteile nicht aktualisiert werden. Diese Aufgabe übernimmt die Klasse ThreadHandler, welche alle Threads in einer ArrayList verwaltet. Die Klasse bietet hauptsächlich Funktionen zum hinzufügen von Threads und abfragen ihres IsAlive Status. Wird über den main Thread im controller eine Filterberechnung angestoßen, dann kann mit Hilfe der ThreadHandler Klasse eine “korrekte” Aktualisierung des Bildschirm Inhalts (also des Images) sichergestellt werden. Dazu bietet sich die Funktion refresh() an, die einen eigenen seperaten Thread anstößt, der nichts anderes macht, als den isAlive Status alle Threads der ArrayListe abzufragen und das Bild zu aktualisieren, sobald alle Threads beendet wurden. Zum Aktualisieren des Bildes ist es hier nötig ein MethodInvoker Object zu erzeugen, da alle anderen Threads (außer dem Main Thread) keine Berechtigung haben Änderungen vorzunehmen, die in Beziehung zur GUI stehen.