# Teilnehmer/innen des Teams:

|  |  |
| --- | --- |
| Klasse:  Ap17c | Team:  **Diogo Caraça**  Silvio Merz |

# Anforderungsdefinition (Meilenstein A)

|  |  |
| --- | --- |
| PC-Optimierer | |
| **Fachlicher Inhalt:**  (Allgemeine Beschreibung) | Nutzen: Mit dem Programm sollte die Performance des Computers gesteigert werden.**Automation:**  Das Programm sollte per Knopfdruck automatisch einige Scripts ausführen. Z.B:   * Speicher optimieren * Leistung optimieren   **Details:**   * Konfiguration: Auswählen was bereinigt werden muss * Integration: Startbare Applikation im Betriebssystem * Administration: Nicht nötig * Sicherheitsaspekte: Programm ist Passwort geschützt     **Erkenntnisse aus der Machbarkeitsabklärung in Windows Powershell:**  Wir haben überprüft ob man Ordner leeren kann, Tasks killen kann und ob man Applikationen vom Autostart aktivieren bzw. deaktivieren kann. |
| **MUSS**  **Kriterien:**  (Konkrete Features, die umzusetzen sind) | **Folgende Features sollen implementiert werden, um einen produktiven Ablauf sicherzustellen:**   * Die Scripts werden über ein GUI per Knopfdruck oder Formulare ausgeführt   + Papierkorb wird geleert   + Vorbestimmte Ordner leeren   + Temporäre Files werden gelöscht   + Autostart von Apps aktivieren/deaktivieren   + Tasks killen |

|  |  |
| --- | --- |
| **KANN**  **Kriterien:**  (Konkrete Features, die optional sind) | **Folgende Features können zusätzlich implementiert werden: (Varianten, Kreativität)**   * User kann zu leerende Ordner selber auswählen * Windows visuelle Effekte können optimiert werden |

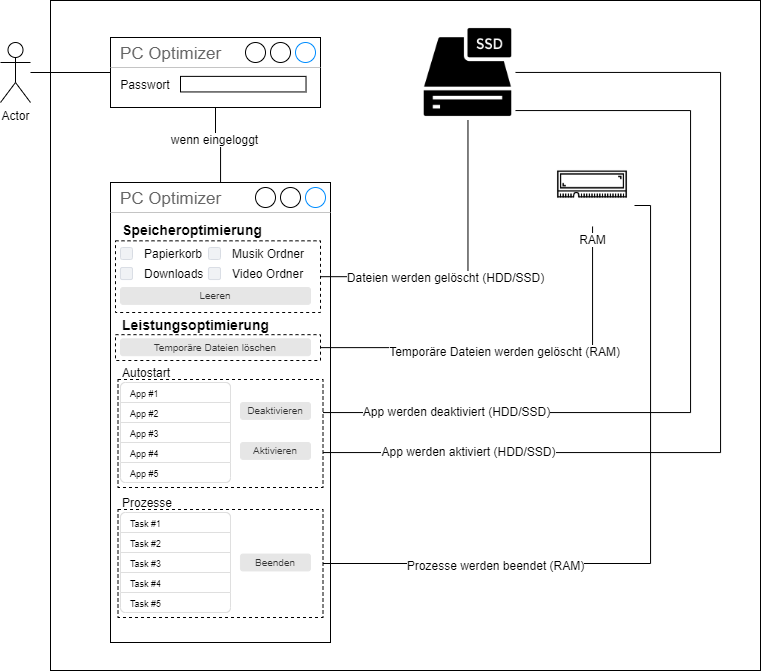
## Planung Meilensteine (LB1 / LB2)

|  |  |  |  |
| --- | --- | --- | --- |
| *MS* | *Tätigkeit / Abgabe* | *Soll-Datum* | *Ist-Datum* |
| A | ProjektstartTeam Bildung, Kollaborationsplattform, GitHub Repos \*, LehrerzugangWahl / Ausarbeitung der Anforderungsdefinition Abnahme Anforderungsdefinition durch Lehrperson | 13.03.2020 |  |
| B | Teamaufgabe 1:Abgabe: Lösungsdesign  (Funktionsmodell / GUI / PAP / Storyboard) | 20.03.2020 |  |
| B2 | Teamaufgabe 2: (Nur LB2)Abgabe: Testvorschrift und Testfälle | 27.03.2020 |  |
| C | Einzelaufgabe 2 (LB1) / 3 (LB2):Abgabe Programmcode und DokumentationFachgespräch Projektabnahme | 03.04.2020 |  |
| C2 | Einzelaufgabe 4: (Nur LB2)Abgabe: Ausgefüllter Systemtest | 03.04.2020 |  |

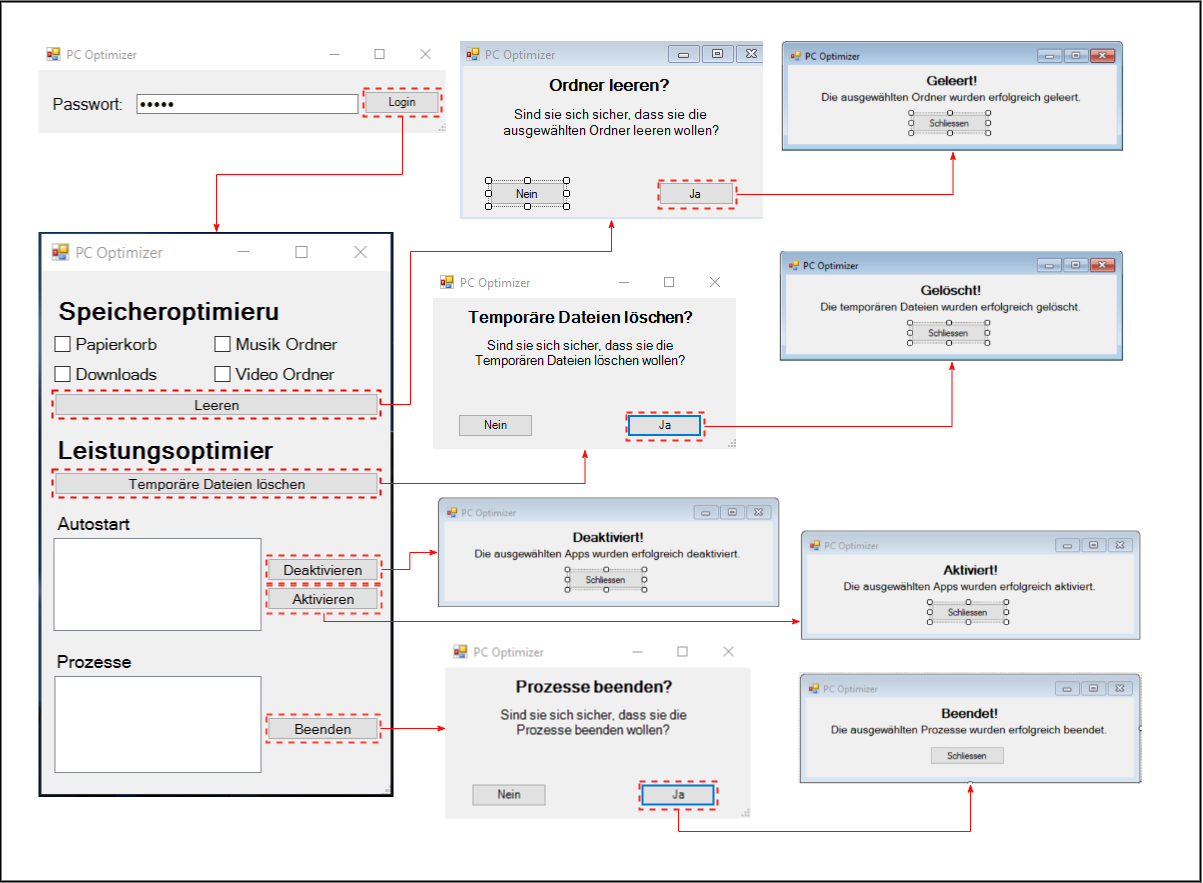
# Lösungsdesign (Meilenstein B: Teamaufgabe 1)

Anhand der Analyse wurde folgendes Lösungsdesign entworfen:

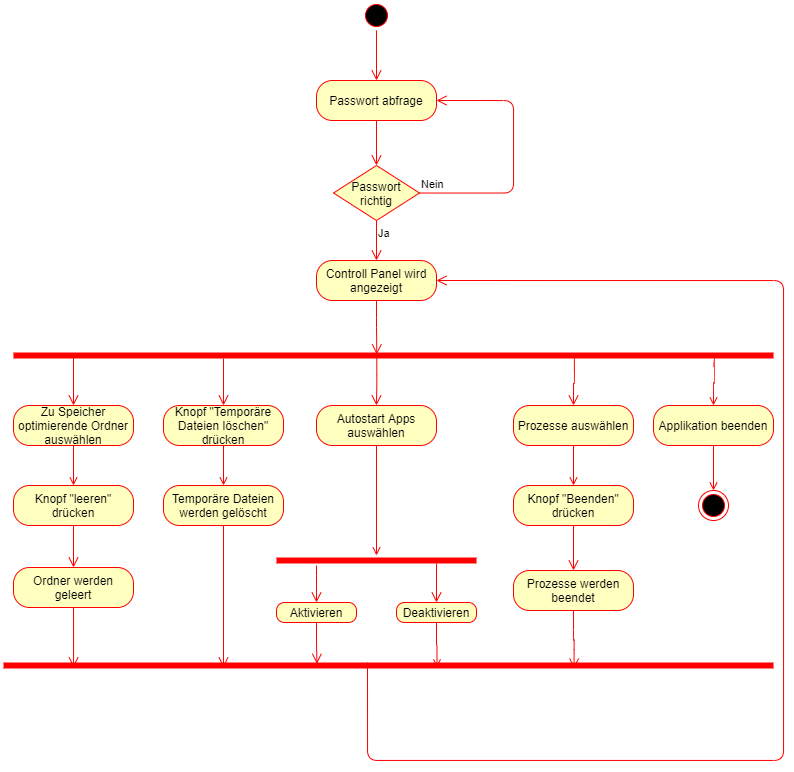
## Schematische Darstellung der Funktionalität, sog. Funktionsmodell



## Graphische Benutzer Schnittstelle (GUI) zur Konfiguration des Ablaufs



## Ablauf der Automation



# Testvorschrift (LB2 Meilenstein B2: Teamaufgabe 2)

Testbeschrieb und vorbereitetes Testprotokoll siehe Dokument:   
M122\_LB2\_Testvorschrift\_MS-B2\_Caraça\_Merz.docx

# Testprotokoll (LB2 Meilenstein C2: individuelle Aufgabe 4)

Ausgefülltes Testprotokoll siehe Dokument:

Von Silvio Merz: M122\_LB2\_Testvorschrift\_MS-C2\_Merz.docx

# Systemdokumentation (Meilenstein C: individuelle Aufgabe 3)

Die erstellten Projekt-WPS-Scripts sind hier abgelegt und für Entwickler dokumentiert:

Öffentliche GitHub-URLs im Ablageordner auf dem BSCW ablegen! (pro Team)

Namenskonvention URL: <https://github.com/SilvioMerz/M122_Ap17c_Diogo_Silvio/>

## Umfang / Abgrenzung / Änderungen gegenüber Design

Aufgrund unten beschriebener Umstände sind Anpassungen des ursprünglichen Lösungsdesigns gemacht worden:

* Pop-Ups / Messageboxes „Nichts gewählt“ hinzugefügt
  + Damit der User sieht, wenn er keine Elemente ausgewählt hat
* „Aktualisieren“-Button bei den Prozessen hinzugefügt
  + Damit die Prozesse in der ListView auf dem neusten Stand sein können
* Maximier- und Minimier-Button entfernt
  + Damit das Design des Fensters immer gleichbleibt
* Pop-Ups / Messageboxes Designänderung
  + Messageboxes enthalten Icons und keine Titel, weil dies dem Design einer Messagebox entspricht

## Funktionalität der Implementation.

Zusätzlich zu der Inline-Dokumentation sind hier folgende Funktionen / Eventhandler detailliert beschrieben:

Event «Leeren»-Button click:

1. ##################################################
2. #Wenn der Button "Leeren" gedrückt wird:
3. ##################################################
4. $emptyDirectives=
5. {
6. #Es wird überprüft, ob alle Checkboxes nicht gewählt wurden
7. if (!$music.Checked -and !$video.Checked -and !$downloads.Checked -and !$paperBin.Checked) {
8. #Benachrichtugungsbox von Zeile 23
9. & $messageBoxEmpty
10. } else {
11. #Wenn mindestens eine Box gewählt wurde, passiert folgendes:
12. #Messagebox wird gerufen, Bestätigung der Aktion wird gefordert
13. $msgBoxInput = [System.Windows.MessageBox]::Show('Ordner leeren?','PC Optimizer','YesNo','Question')
15. switch ($msgBoxInput) {
17. #Wenn der Benutzer 'Yes' drückt:
18. 'Yes' {
19. $checkedfolders = @()
20. #Nimmt den Pfad des Musik Ordners
21. if ($music.Checked) {
22. $checkedfolders += "$env:USERPROFILE\Music"
23. }
24. #Nimmt den Pfad des Video Ordners
25. if ($video.Checked) {
26. $checkedfolders += "$env:USERPROFILE\Videos"
27. }
28. #Nimmt den Pfad des Download Ordners
29. if ($downloads.Checked) {
30. $checkedfolders += "$env:USERPROFILE\Downloads"
31. }
32. #Nimmt den Pfad des Papierkorbs
33. if ($paperBin.Checked) {
34. $sid = ([System.Security.Principal.WindowsIdentity]::GetCurrent()).User.Value
35. $checkedfolders += "C:\`$Recycle.Bin\$sid"
36. }
38. #Eine Iteration holt jedes einzelne Objekt der ausgewählten Ordner
39. Get-ChildItem -Path $checkedfolders -Recurse -Force |
40. ForEach-Object {
41. try {
42. #Löscht das Objekt
43. $\_ | Remove-Item -Recurse -Force -ErrorAction Stop
44. } catch {
45. #Falls es benutzt wird, wird eine Warnung geschrieben
46. Write-Warning "Could not remove Item $($\_.TargetObject)"
47. }
48. }
50. #Messagebox wird gerufen, Ausführung der Aktion wird bestätigt
51. $msgBoxInput = [System.Windows.MessageBox]::Show('Die ausgewählten Ordner wurden erfolgreich geleert','PC Optimizer')
52. }
53. #Wenn der Benutzer 'No' drückt, passiert nichts & die Messagebox geht zu
54. }
55. }
56. }
57. ##################################################

Event «Temporäre Dateien löschen»-Button click:

1. ##################################################
2. #Wenn der Button 'Temporäre Dateien löschen' gedrückt wird:
3. ##################################################
4. $deleteTempFiles=
5. {
6. #Messagebox wird gerufen, Bestätigung der Aktion wird gefordert
7. $msgBoxInput = [System.Windows.MessageBox]::Show('Temporäre Dateien löschen?','PC Optimizer','YesNo','Question')
9. switch ($msgBoxInput) {
11. #Wenn der Benutzer 'Yes' drückt:
12. 'Yes' {
13. #Array mit den Pfaden der Ordner, die temporäre Dateien enthalten
14. $tempfolders = @(“C:\Windows\Temp\\*”, “C:\Windows\Prefetch\\*”, “C:\Users\\*\Appdata\Local\Temp\\*”)
16. #Eine Iteration holt jedes einzelne Objekt der Temp-Ordner
17. Get-ChildItem -Path $tempfolders -Recurse -Force |
18. ForEach-Object {
19. try {
20. #Löscht das Objekt
21. $\_ | Remove-Item -Recurse -Force -ErrorAction Stop
22. } catch {
23. #Falls es benutzt wird, wird eine Warnung geschrieben
24. Write-Warning "Could not remove Item $($\_.TargetObject)"
25. }
26. }
27. #Messagebox wird gerufen, Ausführung der Aktion wird bestätigt
28. $msgBoxInput = [System.Windows.MessageBox]::Show('Die temporären Dateien wurden erfolgreich gelöscht','PC Optimizer')
29. }
30. #Wenn der Benutzer 'No' drückt, passiert nichts & die Messagebox geht zu
31. }
32. }
33. ##################################################

Event «Deaktivieren»-Button click:

1. ##################################################
2. #Wenn der Button "Deaktivieren" gedrückt wird:
3. ##################################################
4. $deactivateApp=
5. {
6. #Es wird überprüft, ob eine App nicht gewählt wurde
7. if ($listAutostart.SelectedIndices.Count -eq 0) {
8. #Benachrichtugungsbox von Zeile 23
9. & $messageBoxEmpty
10. } else {
12. # Name der gewählten App wird geholt
13. $IdColumnIndex = $listAutostart.Columns[0].Index
14. $appname = ($listAutostart.Items[$listAutostart.SelectedIndices[0]].SubItems[$IdColumnIndex]).Text
16. # Appstart wird deaktiviert
17. Set-ItemProperty -Path HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\Run -Name $appname -Value ([byte[]](0x33,0x32,0xFF))
19. #Messagebox wird gerufen, Ausführung der Aktion wird bestätigt
20. $msgBoxInput = [System.Windows.MessageBox]::Show('Die ausgewählte App wurde erfolgreich deaktiviert','PC Optimizer')
21. }
22. }
23. ##################################################

 Event «Aktivieren»-Button click:

1. ##################################################
2. #Wenn der Button "Aktivieren" gedrückt wird:
3. ##################################################
4. $activateApp=
5. {
6. #Es wird überprüft, ob eine App nicht gewählt wurde
7. if ($listAutostart.SelectedIndices.Count -eq 0) {
8. #Benachrichtugungsbox von Zeile 23
9. & $messageBoxEmpty
10. } else {
12. # Name der gewählten App wird geholt
13. $IdColumnIndex = $listAutostart.Columns[0].Index
14. $appname = ($listAutostart.Items[$listAutostart.SelectedIndices[0]].SubItems[$IdColumnIndex]).Text
16. # Appstart wird aktiviert
17. Set-ItemProperty -Path HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\Run -Name $appname -Value ([byte[]](0x02,00,00,00,00,00,00,00,00,00,00,00))
19. #Messagebox wird gerufen, Ausführung der Aktion wird bestätigt
20. $msgBoxInput = [System.Windows.MessageBox]::Show('Die ausgewählte App wurde erfolgreich aktiviert','PC Optimizer')
21. }
22. }
23. ##################################################

 Event «Aktualisieren»-Button click:

1. ##################################################
2. #Wenn der Button "Aktualisieren" gedrückt wird:
3. #Leider konnte ich nicht in kurzer Zeit eine Real-Time Aktualisierung implementieren
4. ##################################################
5. $updateProcesses=
6. {
7. #Listview leeren
8. $listProcesses.Items.Clear()
9. $listProcesses.Columns.Clear()
11. #Prozesse holen
12. $Processes = Get-Process | Select Id,ProcessName | Sort-Object CPU
13. $ProcessProperties = $Processes[0].psObject.Properties
15. #ListView Columheader erstellen
16. $ProcessProperties | ForEach-Object {
17. $listProcesses.Columns.Add("$($\_.Name)") | Out-Null
18. }
19. #Prozesse anzeigen
20. ForEach ($Process in $Processes){
22. $ProcessListViewItem = New-Object System.Windows.Forms.ListViewItem($Process.Id)
24. $Process.psObject.Properties | Where {$\_.Name -ne "Id"} | ForEach-Object {
25. $ColumnName = $\_.Name
26. $ProcessListViewItem.SubItems.Add("$($Process.$ColumnName)") | Out-Null
27. }
28. $listProcesses.Items.Add($ProcessListViewItem) | Out-Null
30. }
32. #Listview Spaltenbreiten angeben
33. $listProcesses.Columns[0].Width = 40
34. $listProcesses.Columns[1].Width = 174
35. }
36. ##################################################

 Event «Beenden»-Button click:

1. ##################################################
2. #Wenn der Button "Beenden" gedrückt wird:
3. ##################################################
4. $quitProcess=
5. {
6. Write-Host $listProcesses.SelectedIndices[0]
7. Write-Host $listProcesses.SelectedIndices[0]
8. #Es wird überprüft, ob Prozesse nicht gewählt wurden
9. if ($listProcesses.SelectedIndices.Count -eq 0) {
10. #Benachrichtugungsbox von Zeile 23
11. & $messageBoxEmpty
12. } else {
14. # Array von gewählten Prozessen holen
15. $SelectedProcesses = @($listProcesses.SelectedIndices)
17. #Ids der Prozesse holen um genau diesen gewählten Prozess zu beenden
18. $IdColumnIndex = ($listProcesses.Columns | Where {$\_.Text -eq "Id"}).Index
20. #Jeden einzelnen gewählten Prozess beenden
21. $SelectedProcesses | ForEach-Object {
23. # ...contains the Id of the process that is currently being "foreach'd",
24. $ProcessId = ($listProcesses.Items[$\_].SubItems[$IdColumnIndex]).Text
26. # ...and stop it.
27. Stop-Process -Id $ProcessId -Confirm:$false -Force
29. }
31. # Damit die Änderung ersichtlich ist in der Listview, erzeugt man hier ein Delay
32. Start-Sleep -s 0.55
34. # Listview aktualisieren nach dem Beenden
35. &$updateProcesses
37. #Messagebox wird gerufen, Ausführung der Aktion wird bestätigt
38. $msgBoxInput = [System.Windows.MessageBox]::Show('Die ausgewählten Prozesse wurden erfolgreich beendet','PC Optimizer')
39. }
40. }
41. ##################################################

Event «Login»-Button click:

1. $loginBtn.add\_Click({
2. if ($passwortInput.Text -ceq "PC-Optimizer\_Admin") {
4. # Login Form schliessen / verstecken
5. $loginF.Hide()
6. $loginF.Close() | Out-Null
8. #PC Optimizer wird gezeigt
9. GenerateForm
10. } else {
11. [System.Windows.MessageBox]::Show('Passwort ist falsch!','PC Optimizer','OK','Warning')
12. }
13. })

# Betriebsdokumentation (Meilenstein C: individuelle Aufgabe 3)

Für Administrator und Benutzer wird folgende Anleitung ausgeliefert ...

## Installationsanleitung für Administratoren

Das Programm ist folgendermassen zu installieren und konfiguriert ...

* PCOptimizer.ps1 herunterladen und in einem gewünschten Ort speichern
* WPS als Administrator starten
* „Set-Executionpolicy unrestricted“ eintippen und ausführen
* Ausführung mit Ja bestätigen
* Script ausführen (Den Pfad eingeben und ausführen)

## Bedienungsanleitung für Benutzer

Das Programm ist folgendermassen zu bedienen ...

* Passwort „PC-Optimizer\_Admin“ eingeben
  + **Speicheroptimierung**
    - Ordner auswählen, die zu leeren sind
    - Auf „Leeren“-Button drücken
    - Ausführung mit „Ja bestätigen“
  + **Temporäre Dateien löschen**
    - Auf „Temporäre Dateien löschen“-Button drücken
    - Ausführung mit „Ja bestätigen“
  + **Autostart**
    - Deaktivieren
      * App auswählen, die zu deaktivieren ist
      * Auf „Deaktivieren“-Button drücken
    - Aktivieren
      * App auswählen, die zu aktivieren ist
      * Auf „Aktivieren“-Button drücken
  + **Prozesse**
    - Auf „Aktualisieren“-Button drücken, um die Liste mit aktuellen Prozessen zu füllen
    - Prozesse auswählen, die zu beenden sind
    - Auf „Beenden“-Button drücken
    - Ausführung mit „Ja bestätigen“