3.3 Erweiterung der Backup-Lösung

3.3.1 Allgemeine Funktionsweise des Backups

Die Backup Lösung wird weiterhin mit einem Cronjob zeitlich koordiniert. Der aktuelle Cronjob enthält folgende Skripte für die Backuplösung:

- ➔ backup.sh
- ➔ cleaning.sh
- ➔ restore_backup_db.sh
- ➔ restore_backup_wf.sh

für das Verständnis bitte die „ROBERTA_Backupsystem.vsd“ ansehen.

3.3.2 Logbuch für den Verlauf der Backups

Das Logbuch wurde erstellt um den Verlauf der Backups kontrollieren zu können. Dazu werden bei jedem Backup, geprüft ob dieses Erfolgreich war. Der Status wird dann ins Logbuch eingetragen.

Mögliche Einträge:

BackupDatenbank_YYYYMMTT -> OK (wordpress enthalten,Export ,Datei nicht leer) Größe: …BYTE

Dieser Eintrag erscheint, wenn alle drei Kriterien erfüllt sind:

- im Inhalt der Datenbank der Begriff **wordpress** gefunden wurde
- & die erstellte Backup Datei der Datenbank **database_YYYYMMTT.sql.gz** existiert
- & die Größe der Backup Datei der Datenbank größer als **0Byte** ist

Falls eine der 3 Kriterien nicht erfüllt wurde, wird eine Fehlermeldung mit passender Fehlerbeschreibung ins Logbuch geschrieben:

BackupDatenbank_YYYYMMTT-> FAIL (Fehler beim Export)
BackupDatenbank_YYYYMMTT-> FAIL („wordpress“ nicht enthalten)
BackupDatenbank_YYYYMMTT-> FAIL (Datei ist leer)

BackupWebfiles_YYYYMMTT -> OK (Datei existiert, Größe) Größe:…BYTE

Dieser Eintrag erscheint, wenn beide Kriterien erfüllt sind:

- Die erstellte Backup Datei der Webfiles **files_YYYYMMTT.tar.gz** existiert
- & die Größe der Backup Datei der Webfiles größer als **0Byte** ist

Falls ein der beiden Kriterien nicht erfüllt wurde, wird eine Fehlermeldung mit passender Fehlerbeschreibung ins Logbuch geschrieben:

BackupWebfiles_YYYYMMTT-> FAIL (Fehler beim Export)
BackupWebfiles_YYYYMMTT-> FAIL (Datei ist leer)

3.3.2 Sicherung der Wordpress-Datenbank und der Webfiles (backup.sh)

Als erstes wird von der Datenbank wie bisher mit einem mysqldump Befehl das Backup erstellt und unter dem Backuppfad **/daten/backup_websystem/** mit dem Namen **database_YYYYMMTT.sql** abgelegt.

Die abgelegte Datei wird nun geprüft.

1.Stufe („wordpress“ enthalten)

Zuerst wird der Inhalt der Datei auf den Begriff wordpress durchsucht. Denn bei erfolgreichem Backup, schreibt der mysqldump Befehl den Namen der Datenbank von der das Backup erstellt wurde in das erstellte File, d.h. in unserem Fall muss der Name unserer wordpress Datenbank in dem File **database_YYYYMMTT.sql** vorhanden sein!

```
#-----------------------------------------------------------------------------------------------------------
# Erste Sicherheitsabfrage, Inhalt prüfen
# hier wird geprüft ob die vom mysqldump Befehl erzeugte Datei auch den Namen unserer zu sicherenden Datenbank
# "wordpress" enthält
#-----------------------------------------------------------------------------------------------------------


BackupOk=`grep -c 'wordpress' database_$(date +%Y%m%d).sql`              #Erstellte Datei auf wordpress
prüfen

if [ "$BackupOk" == "0" ];                                              #Falls Datei nicht wordpress enthält
then                                                                    #-> Fehlermeldung ins Logbuch + RSS
echo "BackupDatenbank_$(date +%Y%m%d) -> FAIL(wordpress nicht enthalten)" >>Logbuch.log
mysql -umartin -pmartin -e "INSERT INTO wordpress.wp_comments VALUES('',458,'Peter Zentgraf','peter.zentgraf@fh-
rosenheim.de','','141.60.160.50',NOW(),(NOW() - INTERVAL 2 hour),'Datenbank Backupfehler (wordpress nicht
enthalten)',0,1,'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0','',0,5)"
Fehler="1"                                                              # Setze Fehlervariable auf 1 = Fehler
fi
#---------------- ENDE 1.Sicherheitsstufe ------------------------------------------------------------------
```

2.Stufe (Export der Daten erfolgreich)

Nun wird geprüft ob auch nach dem zip Vorgang die Daten noch vorhanden sind. Dazu wird in unserm Backuppfad **/daten/backup_websystem/** geprüft ob die gezippte Datei **database_YYYYMMTT.sql.gz** vorhanden ist!

```
#-----------------------------------------------------------------------------------------------------------
# Zweite Sicherheitsabfrage, Existenz
# hier wird geprüft ob die vom mysqldump Befehl wirklich unter dem Richtigen Pfad /daten/backup_websystem/
#-----------------------------------------------------------------------------------------------------------

if [ -f database_$(date +%Y%m%d).sql ];                                 #Ist Backupdatei vorhanden?
then
echo ""                                                                 # Ja, mache nichts
else                                                                    # Nein, Fehlermeldung ins Logbuch + Rss
echo "BackupDatenbank_$(date +%Y%m%d) -> FAIL(Fehler beim Export)" >>Logbuch.log
mysql -umartin -pmartin -e "INSERT INTO wordpress.wp_comments VALUES('',458,'Peter Zentgraf','peter.zentgraf@fh-
rosenheim.de','','141.60.160.50',NOW(),(NOW() - INTERVAL 2 hour),'Datenbank Backupfehler (Fehler beim
Export)',0,1,'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0','',0,5)"
Fehler="1"                                                              # Setze Fehlervariable auf 1 = Fehler
fi
#---------------- ENDE 2.Sicherheitsstufe ------------------------------------------------------------------
```

3.Stufe(Datei enthält Daten)

Um sicher zu gehen, dass wirklich alles funktioniert hat, wird zum Schluss die Datei auf Ihre Größe geprüft, um zu prüfen ob die Backupdatei noch einen Inhalt enthält und somit seine Größe, größer als **0Byte** sein muss.

```
#-------------------------------------------------------------------------------------------------------------
# Dritte Sicherheitsabfrage, Größe
# hier wird geprüft ob die vom mysqldump Befehl wirklich Daten enthält, d.h. wenn keine Daten in Datei-> Größe 0
#-------------------------------------------------------------------------------------------------------------

sizeBackup=`stat -c "%s" database_$(date +%Y%m%d).sql `                # Auslesen der Größe in Byte

if [ $sizeBackup == "0" ];                                             # Ist Größe 0 Byte
then                                                                   # Fehlermeldung in Logbuch + RSS
echo "BackupDatenbank_$(date +%Y%m%d) -> FAIL(Datei ist Leer)" >>Logbuch.log
mysql -umartin -pmartin -e "INSERT INTO wordpress.wp_comments VALUES('',458,'Peter Zentgraf','peter.zentgraf@fh-
rosenheim.de','','141.60.160.50',NOW(),(NOW() - INTERVAL 2 hour),'Datenbank Backupfehler (Datei ist
Leer)',0,1,'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0','',0,5)"
Fehler="1"                                                             # Setze Fehlervariable auf 1 = Fehler
fi
#---------------- ENDE 3.Sicherheitsstufe ----------------------------------------------------------------------
```

Falls in einen der 3.Sicherheitsstufen ein Fehler auftaucht, wird eine passende Fehlermeldung (siehe 3.3.2 Logbuch) ins Logbuch schrieben, zusätzlich wird auch ein RSS Trigger ausgelöst.

4. Erfolgreiches Backup

Bei einem erfolgreichen Backup, wird das File zuerst gezippt und dann ein Eintrag ins Logbuch geschrieben.

```
# Datei komprimieren
gzip /daten/backup_websystem/database_`date '+%Y%m%d'`.sql
# Größe komprimierten Datei auslesen
sizeBackup=`stat -c "%s" database_$(date +%Y%m%d).sql.gz `

# Wenn kein Fehler aufgetreten ist wird ins Logbuch ein Eintrag mit OK gemacht + die Größe der gzippten Datei
if [ $Fehler == "0" ];
then
echo "BackupDatenbank_$(date +%Y%m%d) -> OK(wordpress,Export,Datei enthält Daten)       Größe: $sizeBackup Byte
Gespeichert unter: /daten/backup_websystem/" >>Logbuch.log
mysql -umartin -pmartin -e "INSERT INTO wordpress.wp_comments VALUES('',458,'Peter Zentgraf','peter.zentgraf@fh-
rosenheim.de','','141.60.160.50',NOW(),(NOW() - INTERVAL 2 hour),'Datenbank Backup OK (wordpess,Export,Datei enthält
Daten)',0,1,'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0','',0,5)"
fi
#---------------------- ENDE DATENBANK BACKUP ------------------------------------------------------------------
```

Bei den webfiles werden nur die Sicherheitsstufen 2 und 3 durchgeführt, es ändert sich dort nur der Name **files_YYYYMMTT.tar.gz** Pfad bleibt identisch.

3.3.3 Cleaning der Backupdateien (cleaning.sh)

In diesem Skript werden unter dem Pfad **/daten/backup_websystem**/ alle Backupfiles gesucht deren Änderungsdatum älter ist als 14 Tage. Die gefunden Dateien werden dann gelöscht. Dieses Skript ist nötig um Speicherkapazität einzusparen, da sich dieser ansonsten laufend aufaddiert.

3.3.4 Widereinbinden der Backupdateien

Für das einbinden der Backupdateien wurden die Skripte **restore_backup_db.sh** (für die Datenbank) und **restore_backup_wf.sh** (für die Webfiles) erstellt.

Vorgehensweise beim Widerherstellen:

➔ Öffnen Sie den Pfad in der alle Backups gespeichert sind **/daten/backup_websystem/**

- ➔ Dort finden sie die Skripte **restore_backup_db.sh** und **restore_backup_wf.sh** und alle Backupdateien der letzten 14 Tage
- ➔ Ausführen der Skripte: **./restore_backup_db.sh** bzw. **./restore_backup_wf.sh**
- ➔ Man wird nun aufgefordert einen Dateinamen einzugeben.
- ➔ Geben sie nun den Dateinamen ein z.B. **database_YYYYMMTT.sgl.gz** (restore_backup_db.sh), **files_YYYYMMTT.tar.gz** (restore_backup_wf.sh)
- ➔ Der Rest wird dann automatisch erledigt.

3.3.5 Simulation von Fehlern

Um die Fehler zu simulieren wird im Backupskript (backup.sh) die Zeile mit dem mysqldump Befehl auskommentiert, denn es schwierig direkt in den mysqldump Befehl einen Fehler zu erzeugen.

1. Fall: Es wurde von der Falschen oder keiner Datenbank ein Backup erstellt

Die Backupdatei der Datenbank **database_YYYYMMTT.sgl.gz** wird zuerst entpackt. Nun ist diese Datei unter dem Namen **database_YYYYMMTT.sgl** gespeichert. Wird ändern nun den Inhalt dieser Datei indem wir alle **wordpress** Begriffe aus dem File löschen. Diese muss zuerst ausgepackt werden um nun aus dem Inhalt des Files alle **wordpress** zu löschen, dann unter **database_YYYYMMTT.sgl** speichern.

Nachdem das backup.sh Skript ausgeführt wurde öffnen wir nun das Logbuch und sehen die Fehlermeldung: **BackupDatenbank_YYYYMMTT-> FAIL (Inhalt nicht korrekt)**

2. Fall: Das Backupfile enthält keine Daten -> Größe ist 0Byte

Auch hier wird die Backupdatei zuerst **database_YYYYMMTT.sgl.gz** entpackt. Auch hier wir sie dann unter dem Namen **database_YYYYMMTT.sgl** gespeichert. Nun wird der Ganze Inhalt der Datei gelöscht.

Nachdem das backup.sh Skript ausgeführt wurde öffnen wir nun das Logbuch und sehen die Fehlermeldung: **BackupDatenbank_YYYYMMTT-> FAIL (Datei Größe falsch)**

3. Fall: Die Backupdatei ist nicht vorhanden, wurde nicht erstellt

In diesem Fehlerfall löschen wird das Backupfile komplett, nun ist es nicht mehr im Verzeichnis /daten/backup_webfiles/ vorhanden.

Nachdem das backup.sh Skript ausgeführt wurde öffnen wir nun das Logbuch und sehen die Fehlermeldung: **BackupDatenbank_YYYYMMTT-> FAIL (Datei existiert nicht)**

Der Ablauf der Fehlersimulation für die Webfiles sind, bis auf Fall1 identisch mit denen der Datenbank. Fall1 existiert bei den Webfiles nicht!