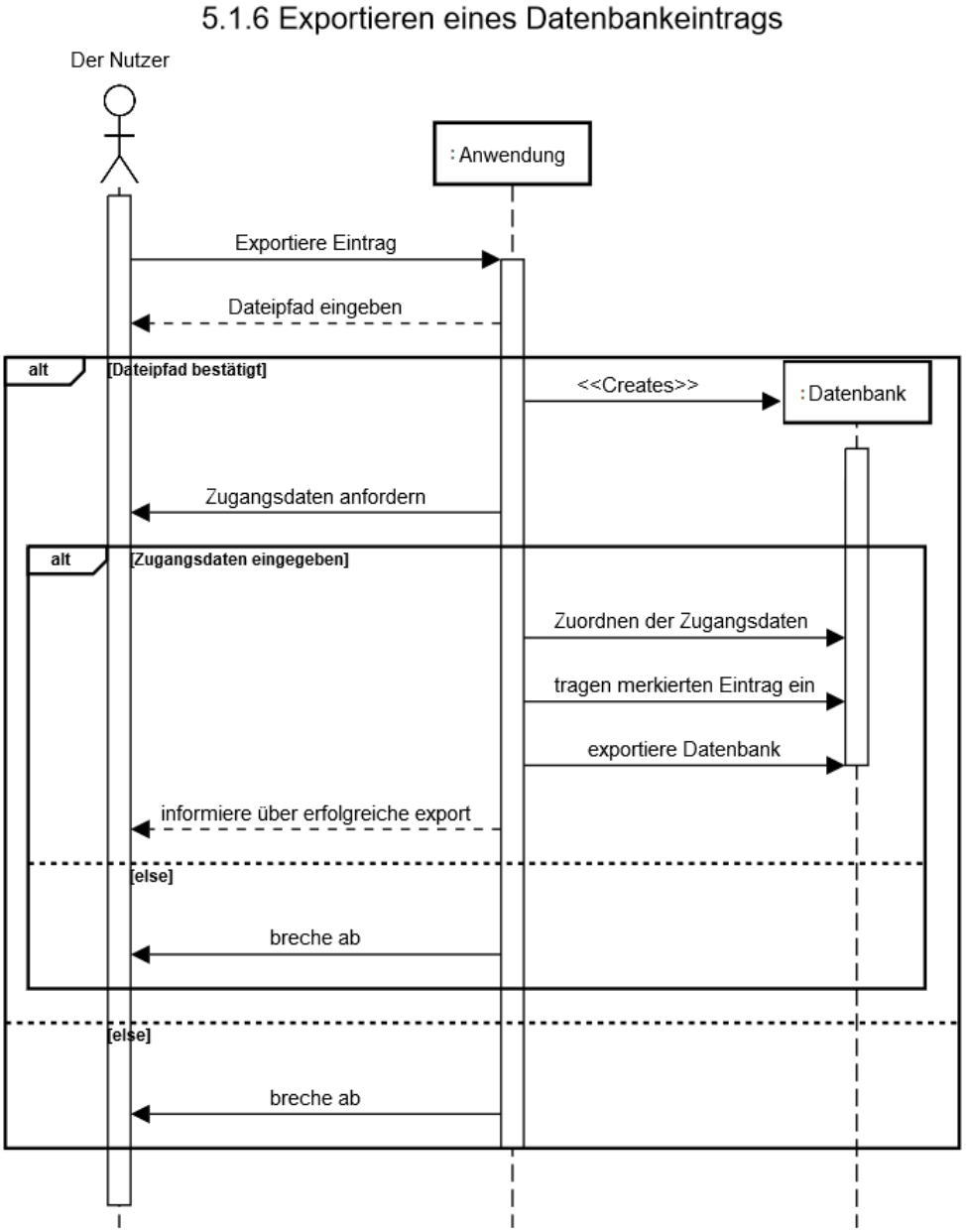
Philipps-Universität Marburg
Fachbereich Mathematik und Informatik

Prof. Gabriele Taentzer
Stefan John

Softwaretechnik
Übung 6

Name:	Matrikelnummer:	Studiengang:	Angestrebter Abschluss:
Melisa Lugji	3224131	Wirtschaftsinformatik	B.Sc.
Touni Arar	3232875	Informatik	B.Sc.
Nassim Maluli	3216710	Wirtschaftsinformatik	B.Sc.

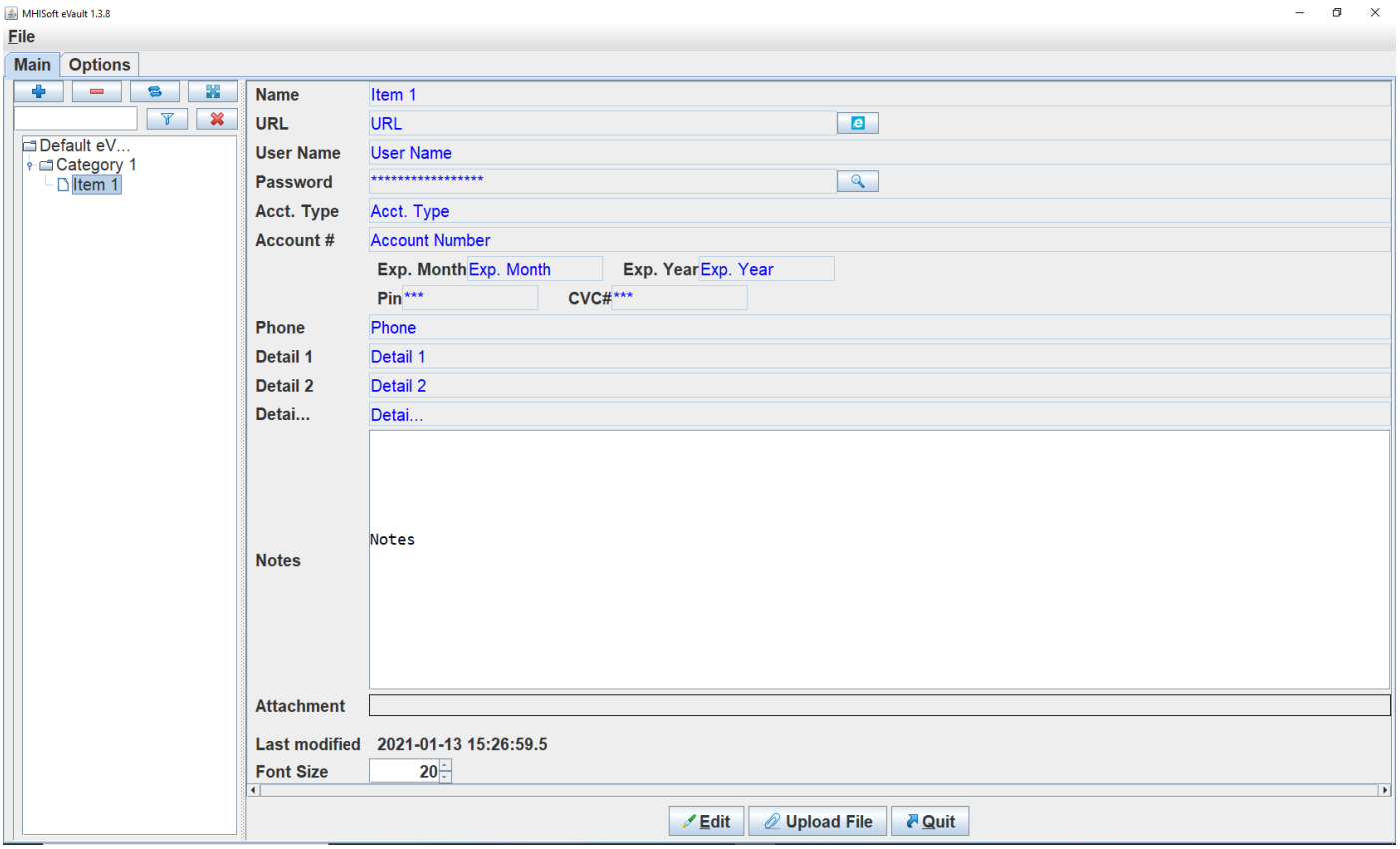
1 Interaktionsdiagramm



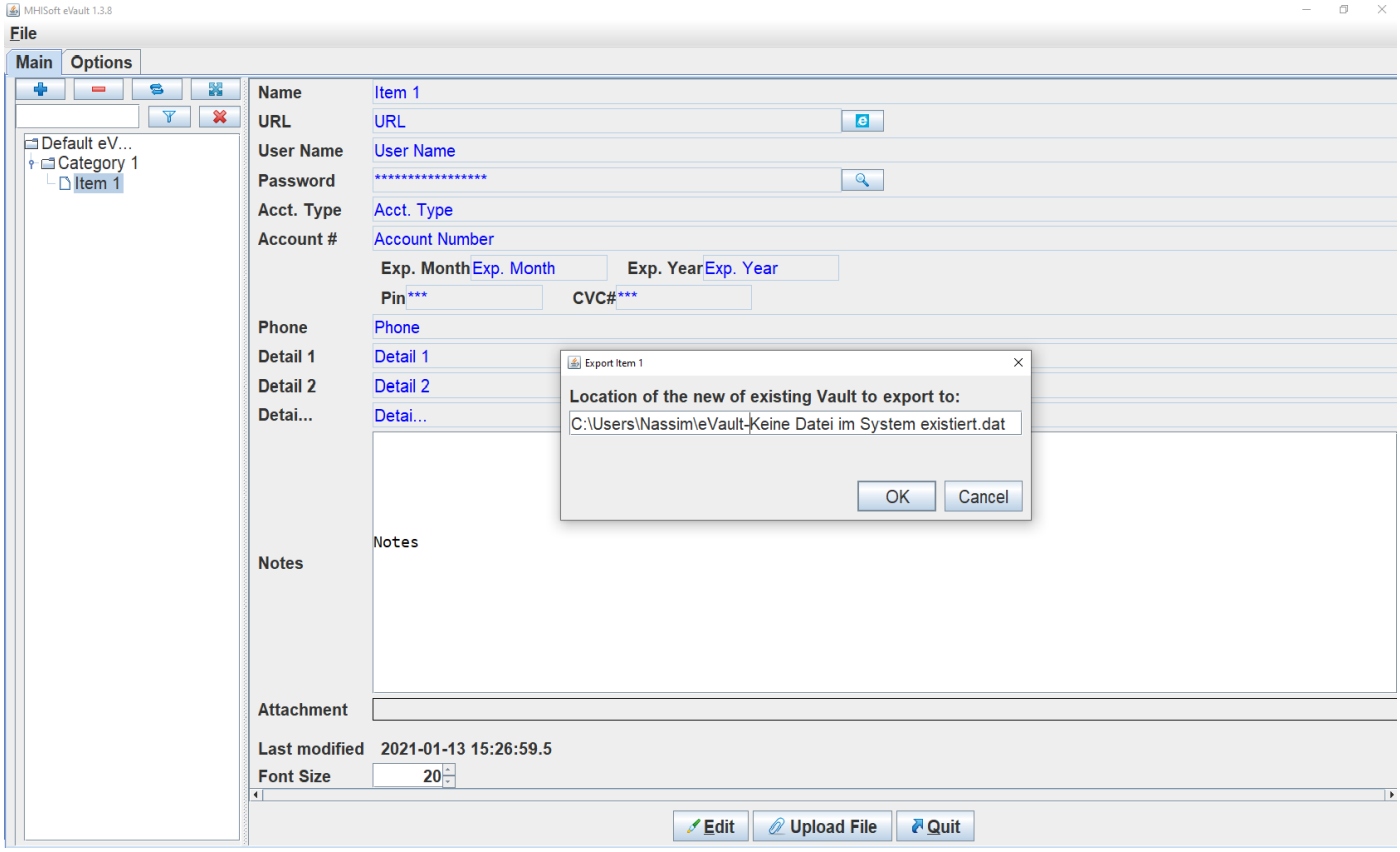
2 Analysieren, Testen, Verstehen

a) Vorbedingung

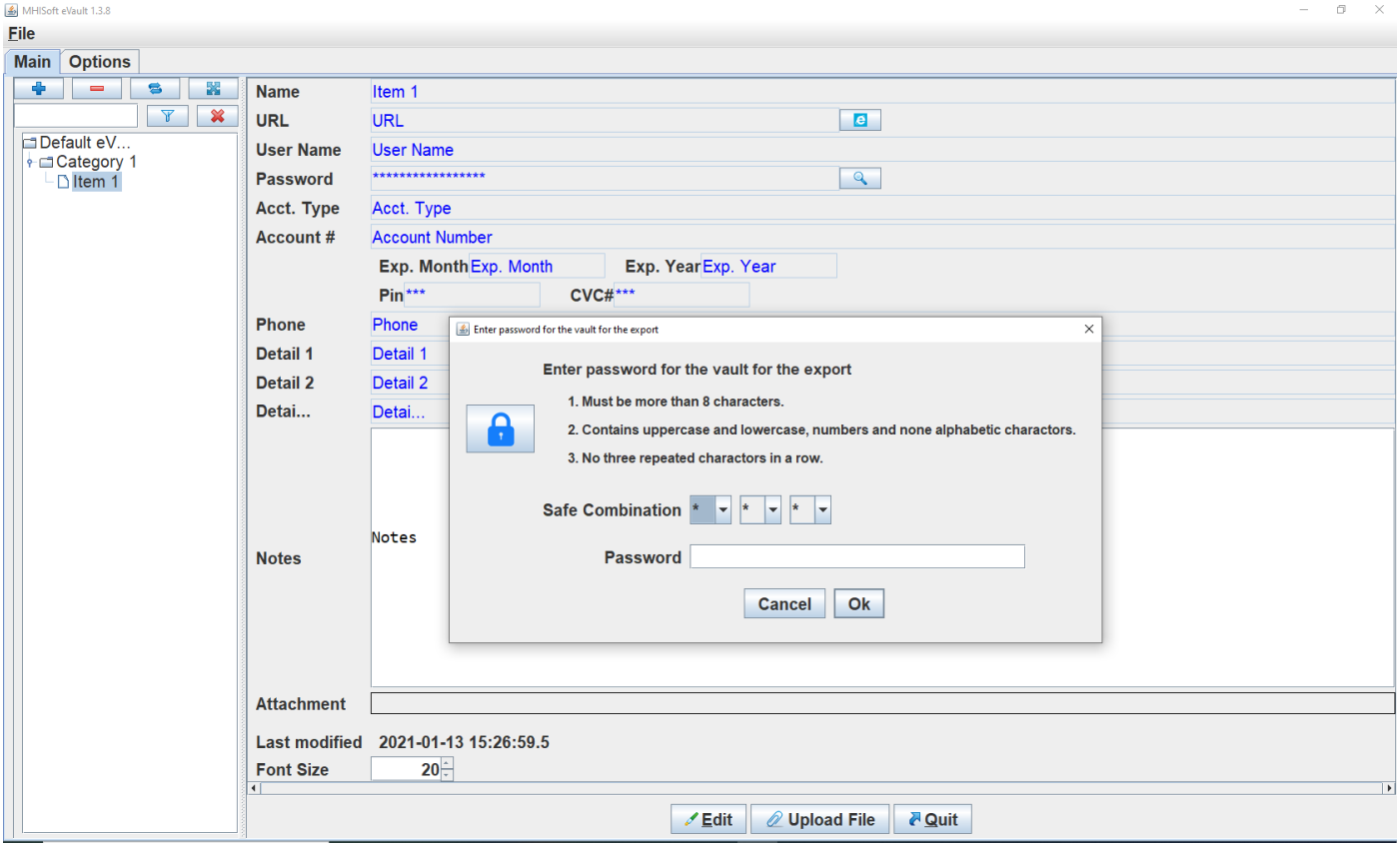
Eine Datenbank ohne aktives Änderungs-Flag ist geöffnet. Ein Eintrag ist ausgewählt.



Der Nutzer erhält die Möglichkeit, einen Dateipfad (ein neuer Dateipfad wurde angegeben) anzugeben

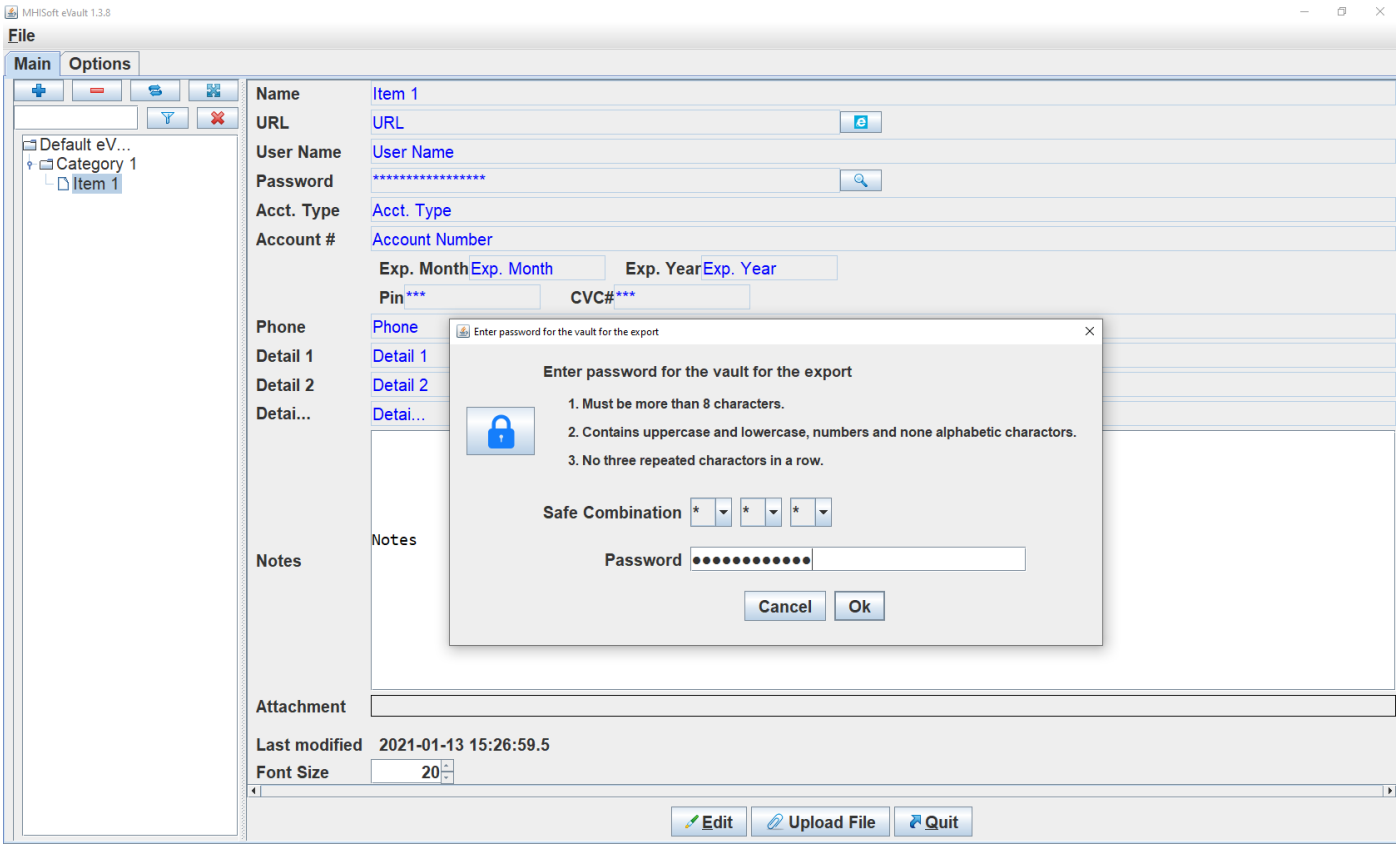


Bestätigt der Nutzer seine Wahl, wird er zur Festlegung der Zugangsdaten für eine neue Datenbank aufgefordert

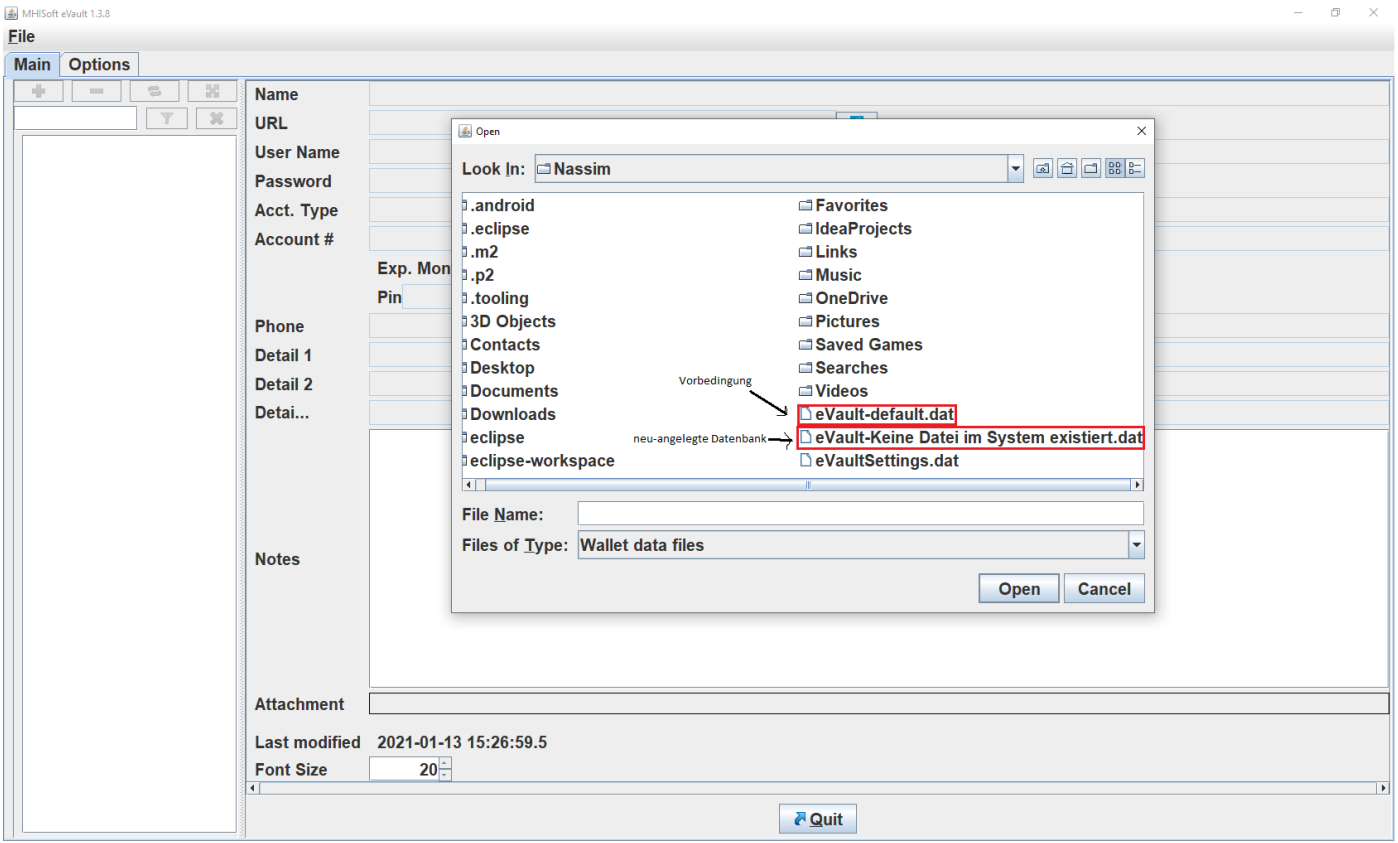


Test-Ablauf:

Angabe der Zugangsdaten:



Anschließend wird unter dem gewählten Pfad eine neue Datenbank mit den eingegebenen Zugangsdaten angelegt, die den markierten Eintrag enthält.

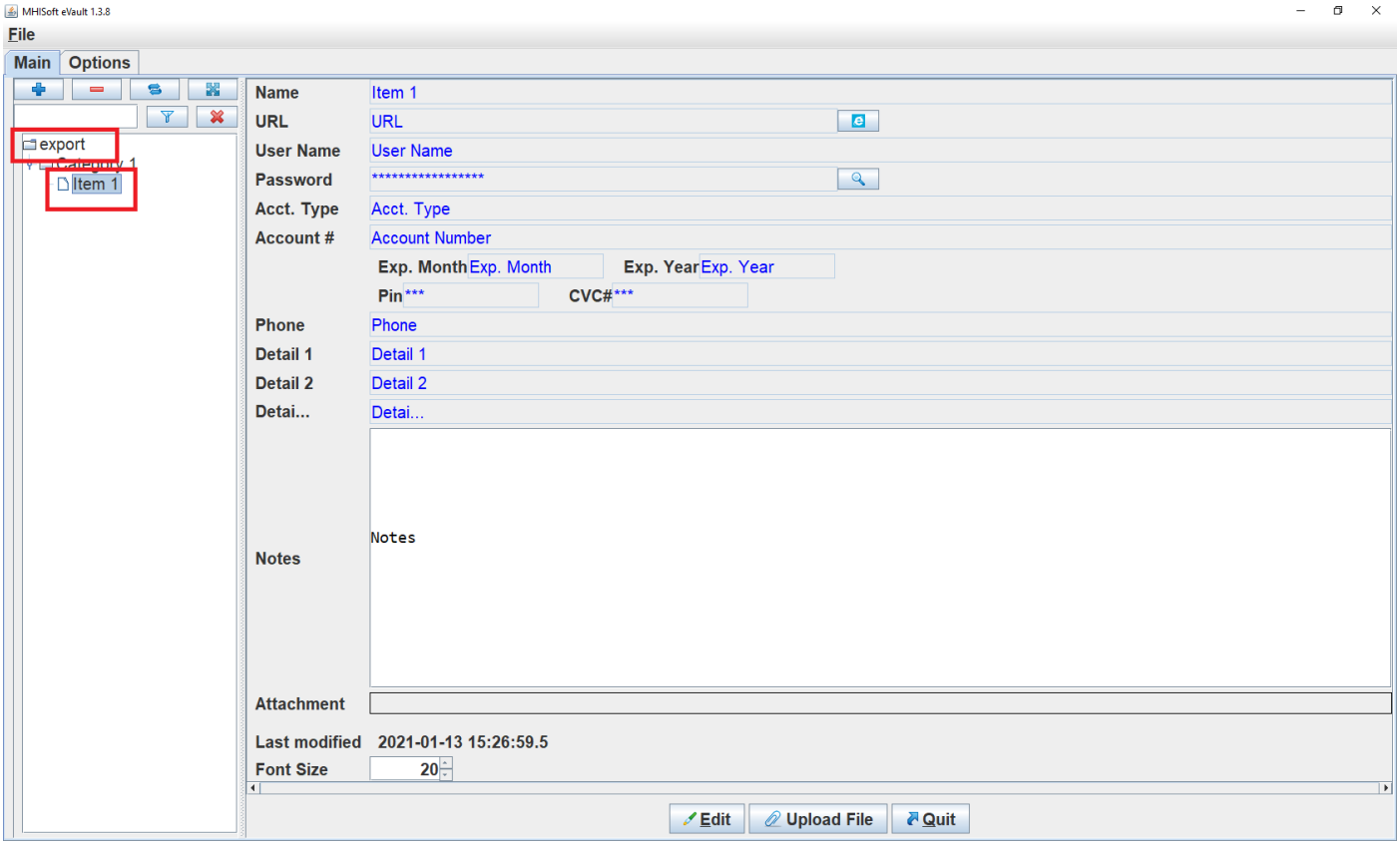


Erwartete Ergebnis:

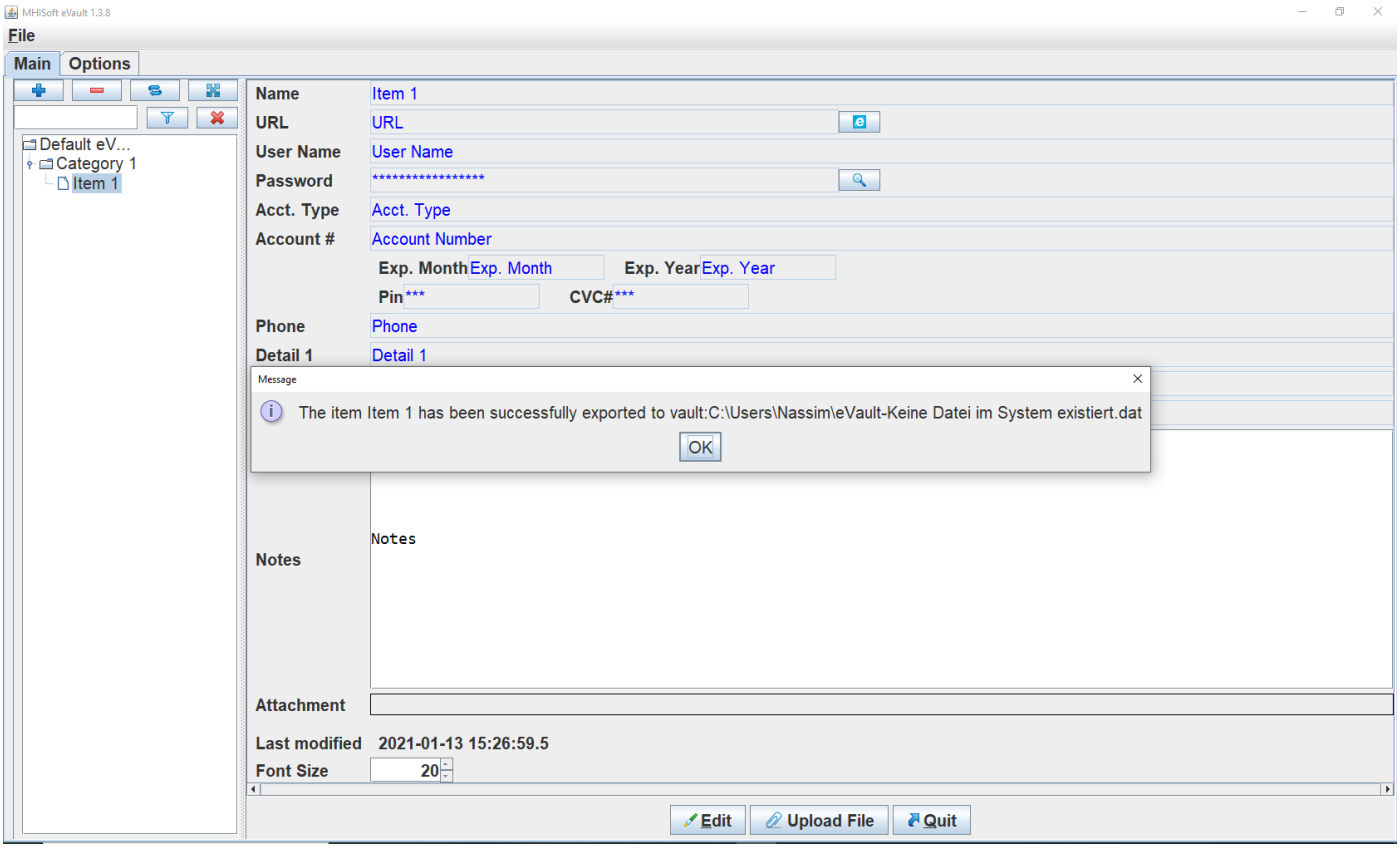
Eine neue Datenbank mit den angegebenen Zugangsdaten ist unter dem angegebenen Pfad angelegt und enthält den exportierten Eintrag. Der Nutzer wird über den erfolgreichen Export informiert.

Erhaltene Ergebnis:

- Neue Datenbank enthält exportierte Eintrag



- Der Nutzer wird über den erfolgreichen Export informiert:



b)

In der Klasse *WalletServiceTest* ist bereits ein JUnit-Test für einen wesentlichen Teil der Export-Funktion, die Methode *exportItem*, implementiert. Analysieren Sie den Testfall und extrahieren Sie aus diesem eine Testfall-Spezifikation.

Achten Sie darauf, dass auch ein Testfall *Bugs* enthalten kann. Erläutern Sie kurz einen Fehler im Testfall, der bei der Initialisierung und Verwendung der zu exportierenden Datenbank erkennbar ist. [4 Punkte]

- **Vorbedingung**

- Ein initialisiertes Wallet „**test_vault_001**“ mit folgender Test-Struktur [Wurzel: **My Default Wallet 1**, Kategorie 1: **Bank Info**, Item 1: PNC Bank, Item 2: GE Bank, Kategorie 2: **Car**, Item 3: Audi, Item 4: Honda] dessen Item **cNode** exportiert wird.

- Ein neu initialisiertes Wallet "**test_vault_001_exported**" mit neuen angelegten Zugangsdaten „passVO2" dient als Export-Ziel des Items aus **„test_vault_001**".
- Ein Item (cNode) wird in das Ziel-Wallet zusammen mit den Zugangsdaten exportiert.

- **Test-Ablauf**

Die Anzahl der tatsächlichen Items wird mit dem erwartete Anzahl der exportierten Items (also cNode und ihre Kinder : dNode, eNode + den Wurzel) verglichen.

Das exportierte cNode Item wird zusammen mit ihren Kindern mit dem Items an dem entsprechenden Stellen in dem Ziel-Wallet, die in der Datei sind.

Es werden auch die Väter der exportierten Kindern verglichen.

- **Erwartete Ergebnis**

Die erwartete Anzahl der exportierten Items beträgt **4** und es wird erwartet, dass die exportierte Item/s in dem Ziel-Wallet „**test_vault_001_exported**" (was z.B. mit dem Befehl **fc.getWalletItems().get(2)** zugegriffen werden kann) in jedem Test den Ursprünglichen Items entsprechen.

Die Väter der exportierten Items (**cNode**) sollen die Väter der Items, die in dem Ziel-Wallet „**test_vault_001_exported**" entsprechen.

Fehler:

- Ausführung der Clear() und SetupTestData() Methode zur Bereinigung der ItemsFlatList und einfügen von neuen Elementen, die

nicht in der Export-Test Verwendet wird (setup() und SetupTestData() machen alles durcheinander) .

- Das Exportieren von cNode, obwohl sie eine Kategorie und kein Eintrag ist.

- Der Aufruf von:

  assertEquals(fc.getWalletItems().get(1), cNode);

  assertEquals(fc.getWalletItems().get(2), dNode);

  assertEquals(fc.getWalletItems().get(3), eNode);

  assertEquals(dNode.getParent(), cNode);

  assertEquals(eNode.getParent(), cNode);

  würde mehr Sinn machen, wenn das Default Wallet "**test_vault_001**", das das zu exportierten Item enthält, mit dem Ziel-Wallet „**test_vault_001_exported**" an der relevanten Stelle verglichen wird und nicht ein Objekt cNode exportieren und mit dem Ziel-Wallet zu vergleichen.

  Es ist auch sehr wichtig darauf hinzuweisen, dass die Väter von dNode und eNode (was eigentlich cNode ist) mit dem Items (als Objekt von Typ WalletItem) verglichen wird (es dient keinem Test-zweck mit cNode zu vergleichen).

Ein sinnvoller Test könnte eventuell folgendermaßen aussehen:

assertEquals(fc.getWalletItems().get(1),model.getItemsFlatList().get(1));

assertEquals(fc.getWalletItems().get(2), model.getItemsFlatList().get(2));

assertEquals(fc.getWalletItems().get(3), model.getItemsFlatList().get(3));

assertEquals(fc.getWalletItems().get(2).getParent(),model.getItemsFlatList().get(1));

assertEquals(fc.getWalletItems().get(3).getParent(),model.getItemsFlatList().get(1));

c) Dieser Randfall ist von der Anforderungsspezifikation nicht gesondert erfasst. Überlegen Sie selbst, wie die Anwendung in diesem Fall reagieren sollte und passen Sie die Anwendungsfallbeschreibung entsprechend an.

Vorbedingung: bleibt unverändert.

Ablauf: ....Wahl, Der Dateipfad wird überprüft, ob eine Datenbank unter diesem Pfad existiert. Wenn das der Fall ist, werden die Zugangsdaten für die vorhandene Datenbank angefordert, anderenfalls wird er....

Abbruchszenarien: ... abgebrochen oder wird eine ungültige Datei, die keine Datenbank unter dem Dateipfad repräsentiert erkannt, wird der Export abgebrochen.

Nachbedingung: ...Eintrag. Gleichermaßen wird ein Eintrag in eine bereits existierende Datenbank exportiert. Der Nutzer...

```
@Test(expected = WalletServiceException.class)
   public void testExportEntry() {
      String eVaultFileExp =
"C:\\Users\\Nassim\\Desktop\\SWT20_Arar_Lugji_Maluli\\code\\evault-
project\\wallet\\src\\test\\resources\\test.dat";
      File f=null;
      try {

         // liste in model leeren
         model.getItemsFlatList().clear();

         // liste in model mit den Test Daten befüllen
         model.setupTestData();

         model.setVaultFileName("test_vault_001.dat");
         ServiceRegistry.instance.getWalletForm().setModel(model);
         PassCombinationVO passVO = new
PassCombinationEncryptionAdaptor("testPa!ss213%", "112233") ;
         String hash = HashingUtils.createHash(passVO.getPass());
         String combinationHash = HashingUtils.createHash(passVO.getCombination());
         model.setHash(hash, combinationHash);
         model.initEncryptor(passVO);

         PassCombinationVO passVO2 = new
PassCombinationEncryptionAdaptor("testPa!ss213%_new","030405") ;
         WalletModel expModel = new WalletModel();
         expModel.initEncryptor(passVO2);

         f = new File(eVaultFileExp);
         f.delete();

         // export model.getItemsFlatList().get(1) = "Bank Info" in model
         walletService.exportItem(model.getItemsFlatList().get(1), passVO2,
eVaultFileExp );

         String hash2 = HashingUtils.createHash(passVO2.getPass());
         String combinationHash2 = HashingUtils.createHash(passVO2.getCombination());
         expModel.setHash(hash2, combinationHash2);

         StoreVO fc  = walletService.loadVault(eVaultFileExp, expModel.getEncryptor()
);

         model.getItemsFlatList().clear();
         model.setItemsFlatList(fc.getWalletItems());
         model.buildTreeFromFlatList();

         // Anzahl der Elemente untersuchen
         assertEquals(4, fc.getWalletItems().size());
         // Schritte in Aufgabe 2b
         assertEquals(fc.getWalletItems().get(1), model.getItemsFlatList().get(1));
         assertEquals(fc.getWalletItems().get(2), model.getItemsFlatList().get(2));
         assertEquals(fc.getWalletItems().get(3), model.getItemsFlatList().get(3));
         assertEquals(fc.getWalletItems().get(2).getParent(),
model.getItemsFlatList().get(1));
         assertEquals(fc.getWalletItems().get(3).getParent(),
model.getItemsFlatList().get(1));

      } catch (HashingUtils.CannotPerformOperationException | WalletServiceException e)
{
         e.printStackTrace();
      }
      finally {
         if (f!=null)
            f.delete();
      }
 }
```

Die Datei Aufgabe2c.txt enthält den gleichen Code falls diese nicht übersichtlich ist.

3 Testabdeckung

```
@Test
public void testAddItemToItem(){
   WalletItem walletItem = new WalletItem();
   walletItem.setType(ItemType.item);
   WalletItem walletItem1 = new WalletItem();
   assertThrows(RuntimeException.class, () -> walletItem.addChild(walletItem1));
}

@Test
public void testAddItemToEmptyCategory(){
   WalletItem walletItem = new WalletItem();
   walletItem.setType(ItemType.category);
   WalletItem walletItem1 = new WalletItem();
   walletItem.addChild(walletItem1);
   assertTrue(walletItem.getChildren().contains(walletItem1));
}

@Test
public void testAddItemToCategoryWithChildList (){
   WalletItem walletItem = new WalletItem();
   walletItem.setType(ItemType.category);
   ArrayList<WalletItem> children = new ArrayList<>();
   walletItem.setChildren(children);
   WalletItem walletItem1 = new WalletItem();
   walletItem.addChild(walletItem1);
   assertTrue(walletItem.getChildren().contains(walletItem1));
}
```

Die Datei Aufgabe3.txt enthält den gleichen Code falls diese nicht übersichtlich ist.