# Anforderungsanalyse

*OOT Semesterprojekt Wer wird Millionär – Gruppe „Millionär“*

## Funktionale Anforderungen

### Funktion „Fragenkatalog in einer Datenbank“

1. Alle Fragen sollen in einer zentralen Datenbank gespeichert werden.
   1.1. Die Fragen sollen nach ihrer Schwierigkeit sortiert werden können.
   1.2. Falls keine Verbindung zu dieser Datenbank aufgebaut werden kann, soll eine lokale Datenbank (die dem Spiel beiliegt) als Ersatz verwendet werden.
      1.2.1. Diese Datenbank wird nicht aktuell gehalten, sie soll dem Nutzer nur das Spielen ermöglichen.
2. Die Applikation muss dem Nutzer eine Oberfläche zur Verfügung stellen, mithilfe dessen er alle Fragen, die im Spiel vorkommen, einsehen und bearbeiten kann.
   2.1. Diese Funktion soll nur autorisierten Nutzern (Administratoren) zur Verfügung stehen.
      2.1.1. Eine simple Passwort-Authentifizierung reicht hierfür.
   2.2. Administratoren sollen nur die Fragen der zentralen Datenbank bearbeiten können (nicht der lokalen!).
   2.3. In dieser Oberfläche muss es möglich sein alle Spielrelevanten Parameter von allen Fragen zu bearbeiten:
      2.3.1. Fragetitel, alle vier Antwortmöglichkeiten, die richtige Antwort und die Schwierigkeit der Frage.

### Funktion „Wer wird Millionär spielen“

1. Die Applikation muss es dem Nutzer erlauben, das Spiel „Wer wird Millionär“ zu spielen.
   1.1. Die Fragen werden aus einer Datenbank entnommen (siehe Funktion „Fragenkatalog in einer Datenbank“)
   1.2. Die gestellten Fragen sollen progressiv in ihrer Schwierigkeit steigen.
   1.3. Es sollen insgesamt 15 Fragen gestellt werden.
   1.4. Wenn eine Frage gestellt wird, soll der Nutzer eine von vier Antwortmöglichkeiten auswählen können:
      1.4.1. Wenn er die richtige Antwortmöglichkeit wählt, steigt sein potentieller Gewinn in Abhängigkeit von der beantworteten Frage (siehe Checkpoints und Gewinne).
         1.4.1.1. Er hat nun die Möglichkeit seinen aktuellen Gewinn zu behalten, und sich vom Spiel zurückzuziehen.
         1.4.1.2. Wenn er von dieser Funktion nicht Gebrauch macht, wird die nächste Frage gestellt.
      1.4.2. Wenn er die falsche Antwortmöglichkeit wählt, gewinnt er den Betrag seines zuletzt erreichten Checkpoints (siehe Checkpoints und Gewinne) und es werden keine weiteren Fragen gestellt.
   1.5. Immer wenn eine Frage gestellt wird, soll der Benutzer die Möglichkeit haben, einen von drei Jokern zu verwenden (siehe Funktion „Joker“).

1.6. Wenn das Spiel endet (weil alle Fragen beantwortet wurden oder der Spieler verliert), soll der Nutzer dazu in der Lage sein, seinen Gewinn als Highscore[1] zu speichern (siehe Funktion „Highscores“).
    1.6.1. Der Spieler hat hier die Möglichkeit, seinen Namen festzulegen.
    1.6.2. Der Gewinn soll nicht tatsächlich ausgezahlt werden. ☺
1.7. Anschließend sollen die besten Highscores in einer Oberfläche angezeigt werden.
    1.7.1. Diese Highscores werden (nur) aus der zentralen Datenbank geladen.

## Funktion „Highscores“

1. Die Applikation muss es dem Nutzer erlauben, seine erreichten Gewinne („Highscores“) zu speichern und mit anderen Nutzern zu vergleichen.
    1.1. Diese Highscores sollen ebenfalls in der zentralen Datenbank verwaltet werden.
    1.2. Es soll möglich sein, die besten zehn Highscores einzusehen.
    1.3. Falls keine Internetverbindung zu Verfügung steht, soll diese Funktion deaktiviert werden.

## Funktion „Joker“

1. Die Applikation definiert drei Joker, die der Spieler einsetzen kann, um eine Frage zu erleichtern
    1.1. Jeder Joker darf nur einmal verwendet werden und ist dann bis zum nächsten Spiel deaktiviert.
    1.2. Der 50:50 Joker deaktiviert zwei falsche Antwortmöglichkeiten.
    1.3. Der Telefonjoker empfiehlt dem Spieler eine Antwortmöglichkeit.
        1.3.1. Diese Auswahl ist so gewichtet, dass dem Spieler zu 75% die richtige Antwortmöglichkeit empfohlen wird.
    1.4. Der Publikumsjoker simuliert eine Publikumsbefragung:
        1.4.1. Jeder Zuschauer wählt die Antwort, die er/sie für richtig hält.
        1.4.2. Zu jeder Antwortmöglichkeit wird dann ein Prozentsatz angezeigt, der die Anzahl der Zuschauer wiederspiegelt, die diese Antwort gewählt haben.
        1.4.3. Die Auswahl der Antwort durch die Zuschauer soll so gewichtet sein, dass mehr als die Hälfte der Zuschauer die richtige Antwort wählt.
            1.4.3.1. Die restlichen Antworten werden zufällig gewählt.

[1] Eine „Highscore“ ist der tatsächlich durch den Spieler am Ende eines Spiels erreichte Gewinn.

## Nicht Funktionale Anforderungen

1. Vorgegeben:
   - 1.1. Implementierung soll in Java erfolgen.
     - 1.1.1. Oberflächen sollen mit dem Swing-Toolkit erstellt werden.
   - 1.2. Source Code soll mit JavaDoc dokumentiert werden.
   - 1.3. Das System soll in mehreren Klassen organisiert sein.
   - 1.4. Für jede Methode müssen ausreichende Unit-Tests definiert und anwendbar sein.
     - 1.4.1. Als Testframework soll J-Unit eingesetzt werden.
     - 1.4.2. Testprozeduren sind ebenfalls zu entwicklen.
   - 1.5. Fehler müssen behandelt werden:
     - 1.5.1. Z.B. Typ- und Plausibilitätsprüfungen bei Nutzereingaben.
   - 1.6. Vor „fatalen“ Benutzerinteraktionen (z.B. Ende) soll es eine Rückfrage beim Benutzer geben.
2. Weitere NFRs:
   - 2.1. Verbindung mit der Datenbank
     - 2.1.1. Die Datenbankverbindungen sollen in einem Connection Pool gespeichert werden.
       - 2.1.1.1. Hierfür wird die Apache DBCP Implementierung genutzt.
     - 2.1.2. Die lokale Datenbank soll eine SQLite Datenbank sein.
       - 2.1.2.1. Diese Datenbank wird beim Kompilieren im Artefakt gespeichert.
       - 2.1.2.2. Diese Datenbank ist read-only.
     - 2.1.3. Die zentrale Datenbank ist eine MariaDB Datenbank.
   - 2.2. Das Projekt wird als Maven[2] multi-module Projekt organisiert.
     - 2.2.1. Maven übernimmt den Build-Vorgang und Dependency Managment.
     - 2.2.2. Mit dem Surefire Report Plugin werden mittels Maven ausführliche Testberichte kompiliert.
   - 2.3. Das Projekt wird mit Github versioniert und ist unter http://oot.marcelherd.com einsehbar.
     - 2.3.1. Bei jedem Push wird mittels Travis CI[3] automatisch ein neuer Build Prozess gestartet.
       - 2.3.1.1. Dabei werden auch alle Unit Tests automatisch ausgeführt.
       - 2.3.1.2. Sollte der Build fehlschlagen, wird automatisch eine E-Mail Benachrichtigung an den Verantwortlichen (Autor des Commits) geschickt.
   - 2.4. Alle Diagramme und Dokumente sollen wahlweise als .jpeg oder .pdf exportiert werden.
   - 2.5. Zu erstellende Artefakte sind:
     - 2.5.1. Anforderungsanalyse
     - 2.5.2. Use-Case-Diagramm(e)
     - 2.5.3. Aktivitätsdiagramm(e)
     - 2.5.4. Sequenzdiagramm(e)
     - 2.5.5. Klassendiagramm(e)
     - 2.5.6. Testreports und schriftliche Testprozeduren
     - 2.5.7. Installations-, ggf. Konfigurations- und Startanleitung
     - 2.5.8. Lauffähiges Programm

[2] Projekt Management Tool
[3] Continuous Integration

# Erläuterungen

## Checkpoints und Gewinne

Das Spiel „Wer wird Millionär“ besteht aus 15 gewählten Fragen, wobei jede Frage gleichzeitig eine Stufe repräsentiert. Mit jeder Stufe wird ein Gewinn assoziiert:

| | |
|---|---|
| 1. Frage | €50 |
| 2. Frage | €100 |
| 3. Frage | €200 |
| 4. Frage | €300 |
| **5. Frage** | **€500** |
| 6. Frage | €1,000 |
| 7. Frage | €2,000 |
| 8. Frage | €4,000 |
| 9. Frage | €8,000 |
| **10. Frage** | **€16,000** |
| 11. Frage | €32,000 |
| 12. Frage | €64,000 |
| 13. Frage | €125,000 |
| 14. Frage | €500,000 |
| 15. Frage | €1,000,000 |

Die fünfte und zehnte Frage stellen jeweils „Checkpoints“ dar – wer diese Fragen richtig beantwortet, hat den damit assoziierten Gewinn sicher (auch wenn man verliert).

Beim (erfolgreichen) Ende des Spiels, wird dann *genau der* Betrag ausgeschüttet, der mit den zuletzt richtig beantworteten Fragen assoziiert wird. Der Gewinn beträgt **nicht** die Summe aller vorherigen Beträge!