PS Softwareentwicklung und Projektmanagement

Code-Review

Reviewteam:

Mitglied 1: (Peter Christian Rottmayr, 01633227)
Mitglied 2: (Manuel Kuß, 11722667)

Softwareteam: 5

Proseminargruppe: 3

Datum: 17.05.2019

Hinweis: Dieses Dokument versteht sich als Guideline und dient der Strukturierung des Review-Berichts. Bitte halten Sie in diesem Bericht zusammengefassten Beobachtungen, Kritiken, Verbesserungsvorschläge in einer verständlichen und konstruktiven Art und Weise fest. Das Entwicklungsteam muss auf Ihren Review schriftlich Stellung nehmen und gegebenenfalls Änderungen am Source Code vornehmen. Wenn notwendig, untermauern Sie Ihr Review mit Source Code-Auszügen, Screenshots etc.

1. ## Zusammenfassung

Der Code ist großteils schön strukturiert und übersichtlich. Es gibt des öfteren noch auskommentierte Codezeilen bzw. ganze Blöcke. Die Dokumentation sollte konsequenter durchgezogen werden. Grobe Fehler sind uns keine aufgefallen, die Teilung der QuizRoom Datei in mehrere kleine Blöcke, ein einheitliches Logging und die ausführlichere Dokumentation wären als leichte Fehler aber noch verbesserbar. Im Gesamten scheint der Code aber sehr zufriedenstellend und gut erarbeitet.

1. ## Architektur

- Wie wird die Architektur aus dem Systemkonzept im Code umgesetzt?

  Der Code gibt ziemlich genau das Systemkonzept wieder, sowohl in den angebotenen Funktionen als auch in der GUI Oberfläche.

  Folgende Use Cases, sind noch nicht vollständig implementiert: Change Password, Use Joker, Receive Questions. (Anmerkung)

- In welchen Komponenten wird das MVC-Pattern umgesetzt?

  Das Datenbank Model ist sehr gering gehalten und wird mit einer zusätzlichen sehr großen Logik-Schicht ergänzt. Die Funktion der Controller wird eher durch die Beans definiert. (Mittlerer Fehler)

- Ist die Folge von Systemaufrufen von der Benutzungsoberfläche bis zur Datenbank und wieder zurück verständlich und qualitätsvoll umgesetzt?

  Ja.

- Ist das aktuelle Inkrement (vgl. Zeitplan im Systemkonzept)vollständig umgesetzt?

  Ja. Das Team liegt gut in der Zeit

1. ## Code

- Ist der Code verständlich und hält Konventionen ein?

  Großteil ja. Teilweise fehlen Dokumentationen und es gibt sehr viele Kommentare. (leichte Kritik)

- Gibt es redundanten Code?

  Fast keinen. z.B. in: AWTImageService.java (Schönheitsfehler)

- Ist die Größe von Klassen und Methoden angemessen?

  Großteils schon.

  Aber es gibt Ausnahmen:
  z.B: QuizRoom, RandomOnlineTestBean, PasswordBean, ...

- Werden Design-Patterns verwendet?

  Es werden bereits implementierte Patterns wie der Iterator oder die Fabrik verwendet. Selbst implementierte Design-Patterns konnten wir keine finden. (Anmerkung)

- Existieren unnötige globale Variablen oder Gottklassen[1]?

  Nein, außer ein bisschen QuizRoom.java

- Implementiert der Code Funktionalität, die von einer Library ersetzt werden kann?

  Nein.

- Befinden sich unnötige Debug-Funktionen im Code (z.B. System.out.println)?

  Nein.

- Wird ein Logger verwendet[2]?

  Ja, es wird ein Logger mit der LoggerFactory aus dem paket slf4j erstellt, jedoch nicht einheitlich verwendet. (leichte Kritik)

- Sind Klassen-, Variablen- und Methodennamen verständlich?

  Meistens gut verständliche Namen, teilweise werden nur buchstaben als Abkürzungen verwendet. Es sollten eher längere (sprechendere) Namen verwendet werden. (Schönheitsfehler)

  Bsp.:

Player	- p	(AddPlayerBean, JoinGameController, ...)
IPlayerAction	- qr	(JoinGameController)
Question	- q	(QuizRoom)
List<ActiveQuestion>	- qs	(QuizRoom)
List<String>	- QSstrings	(QuizRoom)

1. ## Sicherheit

- Werden Inputs überprüft und korrekt weiterverarbeitet? (zB String Konkatenation in SQL Queries)

  Eine Inputvalidierung wird entweder implizit durch den definierten Übergabetype erledigt oder auch wie beim JoinGameController.joinPin durch die Überprüfung der Existenz der PIN in der Datenbank.

- Werden externe Libraries korrekt eingebunden und Exceptions aus diesen korrekt behandelt?

  Ja, aber Exceptions werden teilweise nicht behandelt, sondern nur gecatched. (leichte Kritik)

- Werden Outputs korrekt vorbereitet und ausgegeben?

  Ja.

- Werden ungültige Daten korrekt behandelt?

  Jaein, teilweise werden sie richtig geloggt und behandelt, teilweise wird der Fehler nur gecatched und ausgegeben. (leichte Kritik)

  Bsp:

  ChangeAvatarBean: handleFileUpload() → logged richtig
  CreateManagerBean: redirectRegistration() → printed nur den Stacktrace

1. Dokumentation

- Ist der Code adäquat dokumentiert?

  Das Dokumentieren wird nicht sauber durchgezogen. Oft wird sehr genau und viel dokumentiert, aber in machen Klassen gar nicht. (leichte Kritik)

- Sind Corner Cases dokumentiert[3]?

  Auf Corner Cases wird in der Dokumentation kaum eingegangen. (leichte Kritik)

- Sind verwendete Libraries dokumentiert?

  Nein.

- Sind Datenstrukturen und deren Verwendung adäquat dokumentiert?

  Nein.

- Existiert auskommentierter Code?

  Ja, im Java Code in geringem Ausmaß, in den .xhtml Dateien der GUI mehr. (Schönheitsfehler)

1. Tests

- Werden Unit-Tests automatisch ausgeführt? Werden diese korrekt ausgeführt? Scheitern einzelne Tests oder werden einzelne Tests ignoriert (@Ignore-Annotation)?

  Ja werden automatisch und korrekt ausgeführt. Es laufen alle.

- Sind die Tests verständlich?

  Ja. Verständlich und schön gegliedert.

- Laufen die Tests?

  Ja

- Ist der Code testbar programmiert?[4]

  Ja eigentlich schon, aber durch fehlende Dokumentation und Corner Cases müssen für die Testerstellung die Funktionen mit größerem Aufwand neu verstanden werden. Dadurch verlängert sich möglicherweise das erstellen weiterer Tests. (leichte Kritik)