HAWAI Bicycle Tracker

im Auftrag von
Herr Prof. Stephan Sarstedt

Qualitätskonzept

Louisa Spahl

Version: 0.1
Status: In Arbeit
Stand: 02.05.2015

Zusammenfassung

Dieses Dokument beschreibt die Art und Weise wie auf die Qualität des Codes sichergestellt wird.

Historie

Version	Status	Datum	Autor(en)	Erläuterung
0.1	In Arbeit	03.05.2015	Louisa	Initiale Version für das SEP2 im SoSe 2015

Inhaltsverzeichnis

1 Einleitung

1.1 Ziele

Ziel dieses Dokumentes ist die Beschreibung zur Sicherung der Qualität des Codes.

1.2 Randbedingungen

Weitere Dokumente sind die .xml Dateien zur Sicherstellung der Java Code Konvention.
Ein Dokument bzw. Datei wird dann auf die Qualität getestet, sobald ein Programmierer bzw. Entwickler das/die Issue/Ticket/Karte beendet hat und dieses/diese in die Spalte des Kanban Boards „Review“ schiebt.

1.3 Konventionen

Beispiele werden in diesem Dokument mit dem Schlüsselwort „Beispiel:“ in grüner Schrift markiert.
Codeausschnitte werden in hell blauer Schrift dargestellt.

Beispiel: // TODO (Tom) throw acceptable error

Verweise auf andere Kapitel werden in dunkel blauerer Farbe kenntlich gemacht.
Beispiel: Kapitel 2 „Java Code Konvention“

2 Java Code Konvention

Der Java Code wird mit Hilfe einer .xml Datei automatisch überprüft. Dabei wird diese Datei in die Entwicklungsumgebung IntelliJ bzw. Eclipse eingebunden. Die Konventionen halten sich zum Größten Teil an die Konventionen von Google.
Nicht enthalten in der .xml-Datei ist, dass der Code in der englischen Sprache geschrieben wird, außer es sind Ausgaben für den User auf der GUI.

Bei den Konventionen wird auf Folgendes geachtet:

2.1 TODO's und FIXME

Das Signalwort TODO darf in einem Kommentar sein, jedoch muss nach dem darauffolgenden Leerzeichen der Name des Programmierers in Klammern stehen, der dieses TODO geschrieben hat bzw. für wen das TODO gilt.
Beispiel: // TODO (Tom) implement XY

FIXME darf hingegen nicht in den Kommentaren auftreten.

2.2 Imports

Imports dürfen nicht doppelt auftreten bzw. redundant sein.

2.3 Reihenfolge der Modefier

Es gelten folgende Reihenfolgen:
Beispiel: public, protected, private, abstract, static, final, transient, volatile, synchronized, native

2.4 Konstanten, Variablen und Parameter

Konstanten werden großgeschrieben. Das erste Zeichen muss ein Buchstabe sein zwischen A und Z. Umlaute sind nicht erlaubt. Danach dürfen beliebig viele Buchstaben und Zahlen folgen. Kommt ein Unterstrich im Namen vor, so muss eine Zahl oder ein Buchstaben darauf folgen. Statische Variablen fangen mit einem Kleinbuchstaben an, darauf können Klein-, Großbuchstaben oder Zahlen folgen. Am Ende kann ein Unterstrich sein, aber danach darf nichts Weiteres folgen.
Normale Variablen, lokale (final) Variablen und Parameter müssen mit einem Kleinbuchstaben anfangen, dann können beliebig viele Klein-, Großbuchstaben oder Ziffern folgen.

2.5 Methoden

Methoden fangen mit einem Kleinbuchstaben an. Es folgen beliebig viele Klein-, Großbuchstaben oder Ziffern. Nach einem Unterstrich muss mindestens ein Buchstabe (klein oder groß) oder eine Ziffer folgen.

2.6 Codelänge

Der Code darf nicht über 150 Zeichen lang sein. Wird eine Zeile dieses überschreiten, so muss an einer vernünftigen Stelle umgebrochen werden.

2.7 Klammern

Geschweifte Klammern kommen bei if, else, try und catch in die gleiche Zeile.

Beispiel:
```
if (a != b){
    abc;
} else {
    ...
}
```

2.8 Whitespace

Nach einem Komma, Semikolon oder nach einem Typecast muss ein Leerzeichen stehen.
Es dürfen keine Leerzeichen vor und nach Unären Operationen geben.
Es dürfen keine Leerzeichen nach Klammer auf bzw. vor Klammer zu sein.
Es müssen Leerzeichen um binäre Operatoren sein.

Beispiel: unäre Operatoren: x=9; x++
binäre Operatoren: 7 >= y; i != u

3 HTML Code Konvention

Bei HTML-Elemente müssen bei multi-line die gleichen offenen und geschlossenen Elemente auf der gleichen vertikalen Linie sein. Passen HTML-Elemente auf eine Zeile, so darf auch kein Umbruch erfolgen.

Beispiel:
```
<div id="signin_box">
    <h3>Als neuer Kunde registrieren</h3>
    ....
</div>
```

4 Test Konvention

Das Backend wird mit automatisierten Tests getestet, das Frontend wird durch manuelle Tests getestet.

Die Code Konvention bei Java Tests sind ähnlich wie die Code Konvention des Nicht-Test Codes (siehe Kapitel 2 „Java Code Konvention“, allerdings mit einem Unterschied. Das Testnamenschema basiert auf „Unit of Work“. Dabei ist der Methodentestname in drei Teile aufgeteilt: Der erste Teil ist „Unit of Work“, also die Methode die man testen möchte, der zweite Teil ist das Szenario was getestet werden soll und der dritte Teil ist der erwartete Wert bzw. das erwartete Verhalten, falls das Szenario - beschrieben in Teil Zwei - eintritt.

Beispiel: Allgemein: unitOfWork_Scenario_ResultBehavior
```
IsLoginOK_UserDoesNotExist_ReturnsFalse()
AddUser_ValidUserDetails_UserCanBeLoggedIn()
IsLoginOK_LoginFails_CallsLogger()
```

5 Dokumentation Konvention

Alle Dokumentationen werden auf Rechtschreibfehler überprüft

6 Literatur

„Unit of Work“ Test Konvention:
(http://osherove.com/blog/2012/5/15/test-naming-conventions-with-unit-of-work.html)

Kanban Board:
http://hawai-ast.myjetbrains.com/youtrack/rest/agile/AST%20Kanban%20Board-0/sprint/Sprint%204