EINARBEITUNG

10 MIN

- Startet das Launchscript
  - eclipse: TanteEmmas.launch
  - intellij: tante_emmas.xml -> .idea/runConfigurations
  - shell:

```
java
-cp [classpath]
io.vertx.core.Launcher run net.amygdalum.tanteemmas.server.Server
```

- Navigiert zu http://localhost:8080
- Benutzt die Anwendung
- Stöbert im Source-Code

EIN PAAR HINWEISE ...

- Immer den Server (Vert.x) stoppen bevor man ihn neu startet
- Wenn ihr das Launch-Script verändert behaltet eine Kopie des alten Standes
- Wenn ihr Tests generiert, kopiert die alten Tests am besten in ein eigenes Package damit sie nicht verloren gehen.
- Lest die Javadocs zu den Annotationen
- Generiert genügend Tests, mindestens 100, andernfalls werden manche Effekte zufällig nicht sichtbar

TESTRECORDER INSTALLIEREN

10 MIN

- Legt testrecorder-0.3.x-jar-with-dependencies.jar in euren Workspace
- Fügt testrecorder-0.3.x-jar-with-dependencies.jar dem Klassenpfad hinzu
- Kopiert `AgentConfig.java` ins Package `net.amygdalum.tanteemmas.testrecorder` im Verzeichnis src/main/java
- Macht euch mit der Konfiguration in `AgentConfig.java` vertraut

- Modifiziert das Launchscript durch einfügen von

```
-javaagent:testrecorder-0.3.x-jar-with-dependencies.jar=net.amygdalum.tanteemmas.testrecorder.AgentConfig
```

  - eclipse: run configurations -> TanteEmmas -> Arguments -> Vm Arguments
  - intellij: run -> edit configurations -> tante-emmas -> Vm Options
  - shell:

```
java
-cp [classpath]
-javaagent:[...]
io.vertx.core.Launcher run net.amygdalum.tanteemmas.server.Server
```

- Startet das Launchscript
  - Der Server startet nun mit `loading AgentConfig`
  - Wenn nicht, stattdessen mit `loading default config`? Dann wurde die Klasse `AgentConfig` nicht gefunden (prüft die Packages auf Rechtschreibfehler)
  - Wenn nicht? Dann wurde der agent nicht geladen (der Pfad zur `testrecorderXXX.jar` ist wohl falsch)
- Stoppt den Server
- Annotiert `PriceCalculator.computePrice` mit `@Recorded`

- Startet das Launchscript
  - es startet mit `loading AgentConfig`
  - es folgt `recording snapshots of ...`
  - andernfalls Konfiguration prüfen
- Navigiert zu http://localhost:8080
- Benutzt die Anwendung
- Beobachtet die Konsole und die dort geloggten Tests
- Stoppt den Server und untersucht die generierten Tests im target-Verzeichnis

- Keine Sorge wenn die Tests noch nicht laufen ...
- Wir beheben das später

GLOBALE ABHÄNGIGKEITEN ELIMINIEREN

10 MIN

- Der Grund für die fehlschlagenden Tests ist eine Statische Variable, die nicht aufgezeichnet wurde
  - Statische/Globale Variablen müssen explizit konfiguriert werden
- Findet die globale Variable
- Annotiert sie mit `@Global`

- Startet das Launchscript
- Verwendet die Anwendung und generiert ein paar Tests
- Stoppt den Server und untersucht die Tests

- Einige Tests laufen hoffentlich ...
- Auch um die übrigen Tests kümmern wir uns noch.

DEFINITION VON INPUT-ABHÄNGIGKEITEN

10 MIN

- Der Grund für die fehlschlagenden Tests ist eine nicht definierte Quelle von Input
  - Input sind Zustände die den Programmfluss beeinflussen, die aber nicht unmittelbar vom Programm (sondern von externen Quellen) gesteuert werden.
    - Beispiele: Ergebnisse von Webservice-Aufrufen, Dateizugriff, Zufallszahlen, Zeitanfragen
    - Input muss explizit definiert werden
- Findet die Input Methode(n) und annotiert sie mit `@Input`
- Der Aufruf der Input-Methode muss von `AgentConfig.getPackages` abgedeckt werden

- Startet das Launchscript
- Verwendet die Anwendung und generiert ein paar Tests
- Stoppt den Server und untersucht die Tests

- Die Tests sollten laufen

OPTIONAL: EXCEPTIONS

- Die Methode `PriceCalculator.computePrice` kann Exceptions auslösen
- Findet heraus wie man das herausfordern kann
- Startet den Server und produziert die Exception
- Untersucht den Test der das aufgezeichnet hat

BEHEBEN VON FEHLERN

15 MIN

- Vermutlich hat der eine oder andere schon Fehler im Code gefunden
  - Der Kunde hat das Gefühl dass die Preise nach einem Regentag dauerhaft höher sind ...
  - Ein Tester meinte das manche Aktionen dazu führen, dass sich alle Produkte in Zukunft preislich anders verhalten
- Findet die Variable in der die Produkte abgebildet werden
- Findet die Änderung an dem Produkt die vermutlich nicht beabsichtigt war

- Findet Tests, die die unbeabsichtigte Änderung fixieren
- Wir gehen nun testgetrieben vor:
  - korrigiert die Tests in der Art und Weise, dass sie das wirklich erwartete Verhalten reflektieren
  - korrigiert den Code
- Startet die Tests erneut
  - Vielleicht habt ihr Tests übersehen, wenn sie ebenfalls eine unbeabsichtigen Änderung spezifizieren -> korrigiert sie
  - Es gibt vielleicht auch andere Tests, die jetzt fehlschlagen -> Diskutiert wie hier eine Lösung aussehen könnte
  - Im Zweifelsfall (und wenn man sicher ist, dass die eigene Implementierung korrekt ist) kann man natürlich auch einfach neue Tests generieren

FEEDBACK