Wartungshandbuch

Last Update: 2013-06-16

Projekt “Wartelistenverwaltung planbarer Operationen”

Autoren: Taylor Peer, Michael Wagner, David Stöckl

Version 1.0

Inhalt

Dieses Dokument soll der Weiterentwicklung und Wartung des Projekts “Wartelistenverwaltung planbarer Operationen” dienen. Es wird empfohlen, die hier beschriebenen Werkzeuge und Tools zu verwenden, um Konsistenz innerhalb des Projekts zu garantieren und eine möglichst einfache Wartung / Weiterentwicklung zu ermöglichen.

1. Beschreibung von IDE

Als Entwicklungsumgebung für die Anwendung ist die Spring Tool Suite (Spring STS) zu verwenden (http://www.springsource.org/sts). Diese hat das Spring Framework (V3) bereits integriert und liefert viele Grundlagen für in der Anwendung verwendete Techniken zur schnelleren / besseren Entwicklung, wobei die wichtigsten hier aufgeführt sind:

- Flexible “Dependency Injection” (IoC) mit XML (-> Austauschbarkeit der Komponenten)
- Annotation-basierte Konfigurationsmöglichkeiten
- Gut augestattete Testumgebung für Unit-/Integrationstests

Desweiteren wird für die Beschreibung der einzelnen Server-Komponenten Maven verwendet, um die Abhängigkeiten zu überwachen und weitere Grundeinstellungen für das Projekt verwenden. Maven wird über die **pom.xml-**Dateien konfiguriert, die sich im Root-Folder der jeweiligen Komponente des Projekts befinden.

2. Beschreibung der eingesetzten Frameworks

Spring Framework V3	siehe Beschreibung der Entwicklungsumgebung
RabbitMQ	RabbitMQ wird für den Austausch von Nachrichten zwischen den einzelnen Serverkomponenten verwendet (Nachrichten UI -> Allocators, Allocators -> Messenger)
Embedded MongoDB	Embedded MongoDB wird für Unit-Testing der Data Access Objects verwendet

3. Beschreibung des Build-Prozesses

Für den Build Prozess (Kompilierung, Linking) wird Apache Maven eingesetzt. Mit Maven kann der Build-Prozess weitgehend automatisiert stattfinden, die Abhängigkeiten werden in der pom.xml der jeweiligen Komponente vorgenommen.

Das Update einer Komponente erfolgt wie folgt:

1. Changes “commiten” und mit dem Master-Branch des Projektes auf Github “mergen”

2. Mithilfe des CloudFoundry Plugins Verbindung zur Cloud herstellen
3. Das aktuelle Projekt deployen und Sicherstellen, dass dem Projekt die Services RabbitMQ (für Messaging) und MongoDB (Datenbank) zur Verfügung stehen

4. Beschreibung der Unit, GUI- und Lasttests

4.1 GUI Tests

Die GUI des Projekts wird manuell getestet. Die gesamte Anwendung ist über das GUI verwaltbar. Testdaten können mit Hilfe der Admin-Backend View erstellt werden (/admin/).

4.2 Unit-Tests

Für jede Komponente gibt es eine separate Sammlung von Unit-Tests die in der Ordnerstruktur in src/tests/... zu finden sind. Um Fehler in der Anwendung zu vermeiden, sollte das Resultat sämtlicher Unit-Tests positiv ausfallen. Sie überprüfen bestimmte Methoden der Anwendung wie das Einfügen / Entfernen von Daten. Es können jederzeit weitere Unit-Tests hinzugefügt werden um die Anwendung ausgiebiger zu testen.

Die Unit-Tests sind Komponenten-spezifisch (siehe jeweilige Komponente). In der aktuellen Anwendung fallen alle Unit-Tests positiv aus.

4.3 Last Tests

Die Lasttests für das System sind einfach gehalten und testet die Reaktionszeit des Systems für eine moderat hohe Menge an Anfragen zur gleichen Zeit. Es wird lediglich die Reaktionszeit des Gesamtsystems für Anfragen an der JSON-API gemessen.

Durchschnittliche Reaktionszeit beim letzten Update dieses Dokuments:

+INFO : load.LoadTest - 100 runs: 60840047638 nanoseconds in total, averaged 608400476 nanoseconds per run.
+INFO : load.LoadTest - 50 runs: 30374037393 nanoseconds in total, averaged 607480747 nanoseconds per run.
+INFO : load.LoadTest - 10 runs: 5096445066 nanoseconds in total, averaged 509644506 nanoseconds per run.