Softwareentwicklungsumgebung

Allgemein

Da wir bis zu dem Projekt alle zum größten Teil selbständig gearbeitet haben, hat jeder seine bevorzugte Entwicklungsumgebung, in der man am besten zurecht kommt. Das wollten wir nicht einfach zerstören, da man in dem bekannten Umfeld um einiges besser arbeiten kann, als in einem unbekannten, neuem Umfeld. Wir haben zwar eine bevorzugte Umgebung angegeben, es war uns aber wichtig, dass es am Ende keinen Unterschied macht, welche man benutzt. Somit war am Anfang die Empfehlung von uns Eclipse zu nutzen aber nach einiger Zeit wurde auch vermehrt IntelliJ IDEA verwendet. Um zu garantieren, dass es möglich ist mit jeder Entwicklungsumgebung am Projekt zu arbeiten, haben wir vieles auf IDE-Fremde bzw. unabhängige Tools gesetzt. Dazu gehören:

- Externes CI
- Build-Tools
- IDE-unabhängiges Checkstyle
- Versionskontrolle

Da wir nicht an IDEs und somit dessen Einstellungen gebunden sind, ist ein passiver Nebeneffekt, dass wir das Projekt zu jedem Zeitpunkt wiederherstellen können, falls nötig, und jeder die Möglichkeit dazu hat. Ein weiterer Punkt war, dass so wenig wie möglich am Anfang installiert werden muss, um den Einstig zu erleichtern. Da wir nur begrenzt Zeit haben, ist dies ein großer Vorteil, denn wir können schneller anfangen richtig in das Projekt einzusteigen und müssen uns nicht so viel damit beschäftigen, dass bei jedem die Umgebung läuft. Es wird deswegen, bis auf die IDE und Versionskontrolle, nichts weiter benötigt um mit dem Projekt zu arbeiten, es zu starten oder zu testen.

Build Tools

Wir haben zwei Build-Tools im Einsatz: Gradle und SBT für Back- und Frontend entsprechend. Beide fungieren nicht nur als Build-Tool, sondern auch zum Dependency-Management und Testläufer, wodurch wir gleich mehrere Anforderungen mit wenig Komplexität abdecken.

Gradle ist sehr stark verbreitet und kann als defacto Standard für Java Projekte angesehen werden. Es nutzt die Maven Repositories und es ermöglicht somit den Zugriff auf jede noch so kleine Bibliothek, die man brauchen könnte. Ein weiterer Vorteil ist, dass man in Gradle nicht nur eine Beschreibung des Projekts dem Tool gibt, sondern eigentlich ein Programm an sich, welches eigene Schritte selbst ausführen kann, was eine große Flexibilität mit sich bringt. Es ist möglich das Tool in ein portables Packet zu packen, damit es direkt, ohne Installation, einsatzbereit ist.

SBT im Frontend ist der einzige Teil des Projekts, den wir uns nicht aussuchen konnten, denn SBT wird vom Framework, welches wir benutzen, vorgegeben und wir sind somit auch daran gebunden. Aber auch für dieses wird keine Installation benötigt, denn es setzt sich selber beim ersten Starten selber auf. Auch SBT verwendet die Maven Repositories und hat somit Zugang zu allen möglichen Bibliotheken.

Checkstyle

Wir benutzen ein separates Style-Checking Tool um auch hier wieder nicht auf die IDE Einstellungen angewiesen sein zu müssen. Auch falls man eben gerade nicht in seiner IDE ist und trotzdem eine Änderung anbringen will, dann kann es sein, dass man aus Versehen einen Fehler beim Stil macht. Checkstyle zeigt dann direkt den Fehler, ohne seine Umgebung vorher anpassen zu müssen. Ein weiterer Vorteil liegt darin, dass wir einen zentralen Punkt zur Verwaltung des Stils haben. Falls es kleine Änderungen im laufe der Zeit gibt, könnten diese sehr einfach eingepflegt werden ohne bei jedem die Umgebung zu verändern zu müssen.

Continuous Integration

Um garantieren zu können, dass wir zu jeder Zeit ein laufendes Projekt haben, läuft nach jeder Änderung ein externer Build-Prozess, der das Projekt einmal erstellet und alle Tests durchlaufen lässt. In unserem Fall gibt uns Travis CI nach jedem Build eine Rückmeldung ob alles erfolgreich gelaufen ist, oder die letzte Änderung fehlerhaft war. Außerdem wird geprüft, ob der Coding Style eingehalten wird, wobei eine Verletzung des Coding-Styles einen Fehler im Build hervorruft. Dies fördert das Einhalten des Stils, da so ein Fehler nun gleichgesetzt wird mit einem Fehler in dem Program selbst.

Versionsverwaltung

Im Projekt setzten wir für die Versionskontrolle Git ein, da es wie geschaffen ist für unser Umfeld. Das ist deswegen so, da Git darauf ausgelegt ist, mit verteilten Benutzern zurecht zu kommen. Da wir nicht alle in einem Gebäude arbeiten und somit keinen gemeinsamen Zugang zu einem Server haben, ist es einfacher die Verwaltung mit Git aufzuteilen, wodurch jeder seinen eigenen Git-Server hat und es nur hin und wieder auf eine zentrale gemeinsame Instanz synchronisiert wird. Ein weiterer Vorteil, der uns auch stark zugute kommt, ist, dass wir dadurch nicht gezwungen sind einen Internetzugang zu haben um Änderungen anbringen zu können. Da wir viel zwischen Universität und Zuhause pendeln können wir somit zu jederzeit arbeiten, falls wir das müssten, da immer eine lokale Kopie des Servers besteht. Git fördert auch das Branchen und Mergen, wodurch es einfacher ist gleichzeitig an den gleichen Teilen des Projektes zu arbeiten, ohne Probleme bei Änderungen untereinander zu bekommen. Deswegen lagern wir auch jedes Feature und größere Issue in einen Branch aus um dort in Ruhe arbeiten zu können, ohne den Rest gegebenenfalls von ihrer Arbeit abzuhalten. Auch wenn es nur etwas unscheinbares ist, ist der Fakt, dass Git keine Revisionsnummern hat eine kleine Belastung weniger, da man nicht gedrängt ist, die Nummer möglichst gering zu halten und somit um einiges bereiter ist jedes noch so kleine Teil zu committen. Dies ist auch ein Ziel von uns, denn um so kleiner die Commits, um so einfacher ist es diese zu verstehen und später nachzuvollziehen. Da im Hintergrund die CI nach jedem Commit mitläuft, kann bei kleineren Commits schneller ein Problem festgestellt werden.