Maxim Novichkov 797918

Simon Lischka 797307

Sebastian Dassé 791537

Duc Tung Tong 798029

Schlussbewertung Plan The Jam

Bei der Implementierung von Plan the Jam setzen wir auf Java Technologie in Kombination mit einem aktuellen JavaScript Framework für das Frontend. Ziel war es eine moderne Oberfläche mit HTML verwandten Mitteln gestalten zu können und eine im Browser ausführbare Applikation zu erstellen.

Da aktuell AngularJS und Bootstrap bei Webentwicklern in Mode ist, bestand der Anreiz die Technologie neben dem Erlernen der Java Server Programmierung auszuprobieren und eine Applikation zu erstellen, die sich neben ihrer Funktionalität dem Benutzer möglichst angenehm präsentiert.

Server
VirtualBox / Debian Linux
VirtualBox ist eine Virtualisierungssoftware, die es ermöglicht ein Gastbetriebssystem plattformunabhängig innerhalb des Hauptsystems auszuführen. Dabei wird ein Image erstellt, was mit geringem Konfigurationsaufwand auf einer anderen Maschine zum Laufen gebracht werden kann.

Wir haben VirtualBox mit folgenden Beweggründen benutzt:
- Die Möglichkeit wird offen gehalten, das System ohne großen Aufwand auf einen Server zu deployen
- Die Entwickler werden mit einem einheitlichen System versorgt. Konfigurationen müssen nicht individuell vorgenommen werden sondern können mit dem Austausch des Images verteilt werden. Auf das jeweilige Betriebssystem (in userer Gruppe Windows / OSX) musss keine Rücksicht genommen werden.

In der Anfangsphase des Projektes haben wir nicht einheitlich die VirtualBox verwendet und Technologien z.T. als Eclipse Plugins verwendet. Die nötige Abstimmung im Arbeitsprozess wurde dadurch verzögert.

Da die Entwickler das Bedürfnis hatten, den Komfort ihres lokalen Betriebssystems in der Entwicklungsumgebung zu genießen, haben wir nach und nach folgende Techniken zum Fernzugriff auf die virtuelle Maschine aufgenommen:

SSH - Verbindung zur Virtuellen Maschine: Systemkonfiguration, Starten des Servers.
SSHFS - Mounten des Dateisystems auf der lokalen Maschine, dadurch: Arbeiten mit lokalem Eclipse möglich.

Das Ausführen von Maven muss jedoch auf Kommandozeilenebene erfolgen.

Größte Einschränkung: Debugging nicht möglich.

Sehr spät im Prozesss des Projektes entdeckten wir die Möglichkeit mithilfe von Remote Debugging aus unserer lokalen Entwicklungsumgebung (Eclipse) beim Ausführen des Servers und der Tests Fehler zu diagnostizieren.

PostgreSQL / Hibernate JPA
Da uns ein performanter Datenbankzugriff wichtig war, haben wir PostgreSQL verwendet. Neben der eigentlich Installation auf dem System haben wir durch den Einsatz von JPA allerdings keine datenbankspezifische Operationen erforscht.

Zur direkten Datenbankmanipulation setzten wir PG SQL Admin ein.

Die Einarbeitung in JPA hat viel Zeit erfordert. Das Verständnis der korrekten Transaktionsverwaltung und das Persistieren von Entities, die sich in einer n:m Beziehungen befinden und Kreisreferenzen besitzen, bereitete uns besondere Schwierigkeiten. Dabei experimentieren wir mit verschiedenen FetchType-Konfigurationen. Mit der Einstellung FetchType.LAZY war es uns nicht möglich Collections oder Listen in Transaktionen korrekt anzufordern. Schließlich entschieden wir uns für ein konsequenten Einsatz von FetchType.EAGER und erzwangen mit @JsonIgnore tags, dass diese Collections nicht automatisch über die Rest Schnittstelle gesendet werden. Der Nachteil, dass geg. unnötige Daten von der Datenbank abgefragt werden, nehmen wir dabei in Kauf.

Zum Ende der Projektumsetzung setzten wir vermehrt direkte Queries ein, um komplexere Abfragen über mehrere Klassen zu realisieren.

Jetty / Jersey {Jackson.Json}

Jetty ist ein Web Server und javax.servlet Container, den wir in Kombination mit dem REST-Framework Jersey einsetzen. Ein zentraler Lernprozess beim Entwickeln der REST-Schnittstelle bestand darin, ein angemessenes Maß an Granularität für die angeforderte Daten zu finden.

Wir haben die Pfade in einem relativ späten Projektstadium noch nachgebessert. Zu Anfang wurden beliebige Pfadnamen mit reinen POST Operationen verwendet. Unsere Anpassung bestand darin uns strenger an die REST-Konventionen zu halten, die Operationen GET, PUT, POST und DELETE einheitlicher zu verwenden und die Pfade sauberer zu strukturieren. Dieser Umbau zu einem relativ späten Zeitpunkt war ein gewisser Aufwand, erlaubte aber auch eine klarer und redundanzfreiere REST-Services.

MulTEx
Die Verwendung des Ausnahmemeldungssystems MulTEx erwies sich als intuitiv und erleichterte die Notation von Exceptions.
Vor dem Einsatz von MulTEx hatten wir eine große Menge an eigenen Exception Klassen erstellt.

Probleme beim Konfigurieren von MulTEx:
- Verknüpfung MulTEx doclets mit der process-classes Build Phase.
- Zugriff auf die src/main/resources/MessageResources.properties Datei (muss erst per Hand erzeugt werden

JUnit / FestAssert
Das Testframework JUnit ist Java Standard für Unit Tests und kann durch FestAssert ergänzt werden, einer Biliothek die insbesoners die Validation von verschachtelten Objekten mit Listen erleichtert. Da u.U. verschachtelte Entity Klassen überprüft werden mussten, hat sich die Bibliothek als sehr hilfreich erleichtert und den Codeumfang maßgeblich reduziert.

Das konsequente Abbilden der Fehler auf zustände Testfälle wurde erst in einem späten Stadium der Projektentwicklung das konsequenten Schreiben von Tests insbesondere in Situationen mit größerem Zeitpunkt als gute Taktik.

Wir bemühten uns etwa beim Entwickeln des Datematching Algorithmus eine TDD-Taktik anzuwenden.

Maven
Die Erstellung der pom.xml und die Erlernung des Build-Systems hat große Zeit in der Anfangsphase des Projekts erfordert. Nachdem die Einarbeitung geklappt hatte, erwies sich Maven jedoch als sehr große Arbeitserleichterung.

JavaDoc Plugin
Zur Dokumentation der Java-Klassen- und Paketdokumentation benutzten wir das JavaDoc Plugin. Dabei tauchten keine Probleme auf.

Client
AngularJS
Für die Oberfäche entschieden wir uns für den Einsatz des JavaScript-Frameworks AngularJS. Damit ist es möglich Single-page-Webapplikationen nach dem Model-View-Controller-Pattern zu erstellen. AngularJS unterstützt die Strukturierung der Programmkomponenten in Modulen. Durch eine bidirektionale Bindung von Daten zwischen View und Controller spart man einigen Code für die Aktualisierung der View. Außerdem ist es möglich, mit Services und Direktiven eigene wiederverwendbare Komponenten zu schaffen. Der Umfang des Frameworks erforderte einige Einarbeitung. Letztlich liessen sich die einmal erlernten Mechanismen aber sehr produktiv einsetzen.

Bootstrap
Wir verwendeten das CSS-Frameworks Bootstrap und konnten auf der Basis des Grid-Systems und mit Hilfe der Gestaltungsvorlagen verhältnismäßig schnell ein einheitliches und responsives Design erreichen.

HAML
HTML ist keine unbedingt angenehm lesbare Sprache. Sehr störend ist zum Beispiel die Notwendigkeiten von schließenden Tags. Um diese und andere Unschönheiten bei der Entwicklung zu vermeiden setzten wir HAML (HTML Abstraction Language) ein. Diese verwendet Einrückungen semantisch zur Strukturierung des Dokuments und erzwingt so eine lesbare und knappe Text. Wir setzten ein Ruby-Script ein, das automatisch HAML-Dateien nach HTML übersetzt.

FullCalendar
Für die Visualisierung und möglichst intuitive Manipulation von Terminen benötigten wir eine Kalenderdarstellung. Der Versuch so etwas selbst zu implementieren hätte wahrscheinlich den Rahmen eines eigenen Projekts von gleichem Umfang gesprengt. Deshalb suchten wir ein fertige Lösung, die wir in unserem Projekt verwenden konnten. Das verwendete jQuery-Plugin FullCalendar, das wir in einer für AngularJS angepassten Form verwenden, ermöglicht es Kalenderdaten per Drag and Drop zu editieren und sogar mehrere Layer von Terminen anzuzeigen. Die Dokumentation dieser Bibliothek ist teilweise recht knapp gehalten, war aber ausreichend, um den Kalender für unsere Zwecke zu adaptieren.

Jasmine
Zum Testen verwendeten wir an der Oberfläche das Framework Jasmine. Die Syntax für diese Tests ist angenehm lesbar und die Einarbeitung in die Arbeit mit Jasmine gelingt gut. Problematisch war es, die AngularJS-spezifischen Konstrukte wie AngularJS-Module, Controller und Services testbar zu machen. Hier fehlte einfach die Zeit, schon zu Beginn des Projekts gründlicher die Testbarkeit von AngularJS-Anwendungen zu erforschen und von Anfang an entsprechende Tests zu schreiben. Besonders das Testen von asynchronen Aufrufen ist anscheinend nicht ohne weiteres möglich. Reines JavaScript lässt sich mit Jasmine aber gut testen.

npm, Bower
Für die Verwaltung und Installation der fürs Frontend benötigten Dependencies und zum Starten der Tests wurden der Paketmanager npm und Bower verwendet. Diese lassen sich mit jeweils mit einer JSON-Datei konfigurieren. Beide waren in der Einarbeitung unproblematisch und in Handhabung sehr angenehm.

YUIDoc
Die JavaScript-Dateien dokumentierten mithilfe YUIDoc, das aus Kommentaren im JavaDoc-Stil automatisch eine API-Dokumentation generiert. Durch Annotationen können zum Beispiel functions als Methoden gekennzeichnet und Parameter dokumentiert werden. Hierbei werden aber keine AngularJS-spezifischen Konstrukte wie Controller, Services oder Direktiven unterstützt. Das stärker auf AngularJS zugeschnittenes Dokumentationswerkzeug ngdoc erwies sich als in der Einarbeitung zu aufwändig für den Umfang des Projekts, wäre hier aber sonst die erste Wahl gewesen.

Weitere Werkzeuge
Eclipse, Sublime Text
Zur Entwicklung arbeiteten wir mit Eclipse für Java und mit Sublime Text für JavaScript und HTML.

git
Zur Versionierung setzten wir das Werkzeug git ein, das wir schon aus früheren Projekten kannten. Es hat sich wieder bewährt, insbesondere bei der Synchronisation der verschiedenen Arbeitsstände der Teammitglieder. In der zweiten Hälfte des Projektes sind wir dazu übergegangen, für bestimmte Issues auf eigenen Arbeits-Branches zu arbeiten und diese nur einem arbeitsfähig wieder in den Master-Branch zu mergen. Wir hatten nicht immer die Disziplin, diese Routine einzuhalten. Wenn wir es taten, funktionierte dieses System aber sehr gut.

Fazit
Bei der Umsetzung des Projektes hatten wir ersten Semester des Softwareprojektes insbesonders mit Schwierigkeiten in der Konfiguration zu kämpfen. Erst im zweiten Semester erreichten wir den Punkt, wo die Konfiguration und der Umgang mit Buildsystemen selbstverständlich wurde und so der Arbeitsprozess beschleunigt wurde. Generell galt es aber immer wieder Schwierigkeiten in der Konfiguration zu bekämpfen.

Wir erlernten erst im Verlaufe der Entwicklung uns als Teammitglieder geschickt zu koordinieren. Zum Ende des Projektes waren die Rollen gefestigt, so dass jedes Mitglied einen festen Zuständigkeitsbereich hatte. Wir erreichten, dass eine Selbstverständlichkeit darin enstand, Hilfe oder Reviews bei anderen Mitgliedern "zu beantragen". Insbesonderes durch das gezielte einsetzen von Pair programming erreichten wir sehr viel. Desweiteren gewöhnten wir es uns an, komplette Arbeitstage nur für das Spezifieren vorzusehen. Dadurch entstand eine wesentlich rationalere Herangehensweise an die Projektarbeit.

Wir haben viel über vorher unbekannte Techniken gelernt und uns in unseren Fachgebieten weiterentwickelt. In der Betreuung haben wir sehr davon profitiert, eine realistische Einschätzung über die Machbarkeit einzelnder Vorhaben zu bekommen. Oft waren Tips in Bezug auf die Vorgehensweise und Hinweise, bestimmte Technologien zu verwenden oder wegzulassen maßgebend für die Lösung.

Sowohl fachlich und besonders in Bezug auf Arbeitsprozesse und Teamarbeit haben wir viel bei dem Projekt gelernt. Die gesammelten Erfahrungen werden uns bei einem Einstieg in die Industrie von großem Nutzen sein.