Woche 8
Architektur dokumentieren, prüfen und Deployment

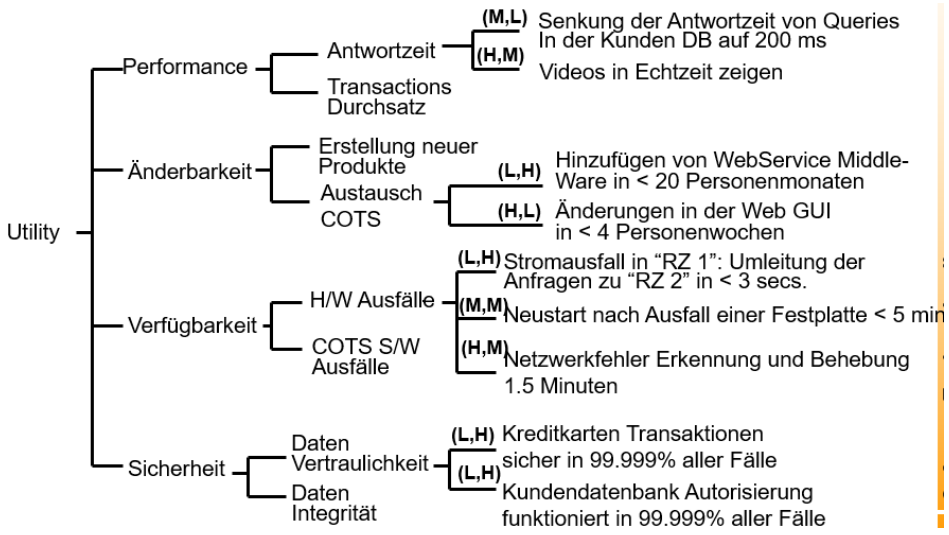
Wir dokumentieren diese Woche zentrale Architektursichten nach oder aktualisieren vorhandene Darstellungen. Wir verwenden die UML 2.5 für die meisten Diagramme. Zusätzlich prüfen wir unsere Entscheidungen gegen Architekturrisiken und juristische Auflagen.

1. Beispiele Brainstormen

Wir brainstormen gemeinsam Beispiele im Zusammenhang mit den Architekturtreibern wie Verfügbarkeit, Skalierbarkeit, Gebrauchstauglichkeit, Performance, IT-Security oder Wartbarkeit. Erstellen sie mindestens 10 Beispiele, ungefähr so:

- Wenn der Server ausfällt, soll das innerhalb von 10 Sekunden auffallen und der Server soll nach einer Minute wieder verfügbar sein. (Verfügbarkeit)
- Wenn das Netzteil des Servers durchbrennt, soll das innerhalb von 10 Sekunden auffallen und der Ersatzserver soll nach fünf Minuten verfügbar sein. (Verfügbarkeit)
- Wenn die Internetverbindung abbricht, kann der Client im Offline-Betrieb weiterarbeiten. (Verfügbarkeit)
- Ein Angreifer ist nicht in der Lage, die Session eines Benutzers zu klauen (IT-Security).
- Ein Angreifer darf nicht in der Lage sein, das Passwort eines Benutzers im Anmeldeprozess zu erforschen, z.B. über Wireshark. (IT-Security).
- Wenn ein neues Feld in der Oberfläche ergänzt wird, ist dies mit einen Aufwand von weniger als 5 Personentagen möglich. (Wartbarkeit).
- Wenn das DBMS ausgetauscht wird, ist das mit einem Aufwand von weniger als 2 Personenwochen möglich (Wartbarkeit).

Ordnen Sie die Beispiele in einem Quality Utility Tree (vgl. Video zu dem Thema, bzw. Folien). Sie können dazu eine MindMap verwenden. Die Priorisierung ist nicht erforderlich.



2. Context: Kontextdiagramm prüfen

Betrachten sie noch mal ihr Kontextdiagramm (ihr System in der Mitte, Nachbarsysteme und Nutzergruppen um umkreis angeordnet). Ist die Darstellung noch korrekt? Finden sich da wirklich die wichtigsten Elemente ihres MVP? Dieses Diagramm darf informell sein.

3. Container: Verteilungsarchitektur prüfen und nachdokumentieren

Sie haben ihre Software in verschiedene Deployment-Einheiten (Container) geschnitten und diese auf der vorhandenen Hardware verteilt. Dokumentieren sie das Trägersystem, die Container und deren Verteilung mithilfe er Verteilungsarchitektur-Sicht. Die Container können Docker-Container sein oder einfache andere Deployment-Einheiten wie z.B. Android APKs. Das Trägersystem kann Android/iOS Hardware, virtuelle Maschinen beim Cloud Provider oder eine VR-Brille sein. Alles inklusive des SW/HW-Stacks auf dem ihre Container ausgeführt werden.

Nutzen Sie dazu ein ***UML-Verteilungsdiagramm (PlantUML oder Draw.io)*** Wenn sie schon eine derartige Darstellung im UML haben, müssen Sie nichts mehr tun, ggf. aktualisieren. Sie war ja schon beim technischen Durchstich gefragt. Achten Sie darauf, dass sie ein syntaktisch / semantisch korrektes Verteilungsdiagramm in UML modellieren! Das ist sperriger als die informelle Darstellung.

Spielen sie bitte die unter 1. gefundenen Szenarios durch. Besteht irgendwo Handlungsbedarf? Müssen sie beispielsweise schnell erkennen, wenn ihr Server abstürzt? Eventuell müssen Sie dafür Komponenten wie einen Watchdog ergänzen. Haben sie genügend Monitoring-Informationen sodass ihr System betreibbar ist?

4. Components: Komponenten-Struktur, Frameworks, Bibliotheken

Eventuell haben sie ihre Container in Schichten oder weitere Komponenten aufgeteilt. Sollte das der Fall sein, dokumentieren sie bitte diese Komponenten mithilfe eines **UML-Kompositions-Struktur-Diagramm**s (bzw. Komponentendiagramms). Die Komponenten-Darstellung ist dann wichtig, wenn sie intern bestimmte Interfaces nutzen und bei ihren Komponenten zwischen Innen- und Außensicht unterscheiden. Dokumentieren sie in dieser Schicht auch, welche Frameworks und Bibliotheken sie verwenden. Beispielsweise SpringBoot 3.0 am Server und Vue.js am Client. Prüfen Sie ihre Grafik bitte gegen die „Dependencies“ aus ihrem Build-Skript.

5. Lizenzen dokumentieren

Wenn Sie sich für ein Frameworks verbaut haben, ist es für ihren Kunden relevant, welche Lizenzmodelle dort verwendet werden. Er muss sich an die Auflagen der jeweiligen Lizenzen halten, wenn er ihre Ergebnisse weiterverkaufen will. Schauen sie z.B. noch mal bei https://openhub.net/ oder auf github, dort sehen sie unter welcher Lizenz ihre Framework laufen. Erstellen sie eine Tabelle mit den genutzten Lizenzmodellen, eventuell kann die Build-Pipeline diese Tabelle auch für Sie erzeugen:

Framework	Beschreibung	Version	Lizenz
React	Für die Entwicklung der Frontend-Anwendungen wird das React Framework verwendet. Dies entspricht der Vorgabe der Auftraggeber.	17.0.2	MIT
Spring	Für die Entwicklung des Java Backends wird das Spring Framework verwendet. Spring bietet sich an, da es weit verbreitet ist und	5.3.15	Apache 2.0

6. Build-Pipeline anpassen

Lizenzscanner: In gitlab können sie mithilfe der Build-Pipeline die Lizenzen ermitteln lassen. GitLab hat dazu einen Lizenzscanner, bitte bauen Sie diesen ein, sofern das ihre Technologie zulässt. Bei den Sprachen Java, C#, JavaScript / TypeScript oder Python sollte es keine Probleme geben.

Vulnerabilities: Der Fehler in der Log4J Implementierung hat sehr schön gezeigt, dass selbst in alten und häufig verwendeten Bibliotheken schwere Sicherheitsmängel auftreten können. Diese Mängel werden in der CVE-Datenbank gesammelt. Gitlab bietet auch diese Prüfungen Unterstützung an. Bauen sie diese bitte in ihre Pipeline ein.

Wir haben jetzt aufwendige Prüfungen ergänzt, diese sollten eventuell nicht bei jedem Push auf das Repository ausgeführt werden. Teilen Sie daher die Pipeline in zwei Stufen auf. Die erste Stufe „Commit-Stage“ wird bei jedem Push ausgeführt. Die zweite Stufe „Deployment-Stage“ wird nur dann ausgeführt, wenn auf den Develop / Main-Branch gemerged wird.

7. Deployment vorbereiten

In der Verteilungssicht unter Aufgabe 3 haben sie ja schon die Deployment-Einheiten definiert. Diese Deployment-Einheiten sollten von ihrer Build-Pipeline oder von ihren Pipelines erzeugt werden. Am Ende ihrer Pipeline(s) entstehen also Docker-Images, Java-Archive, Android-APKs oder ähnliches.

Ändern sie ihre Deployment-Stage vom Build-System möglichst so, dass diese Docker-Images oder andere Artefakte erzeugt und im Artefakt-Repository abgelegt werden. Die Aufgabe ist abgeschlossen, wenn ein oder mehrere Bibliotheken und / oder Container im Artefakt-Repository sind.

Wenn sie Zugriff auf das Trägersystem (z.B. eine Staging-Maschine) haben, könnten sie als letzten Schritt das Deployment automatisch durchführen. Das bitte mit ihrem Auftraggeber abstimmen.