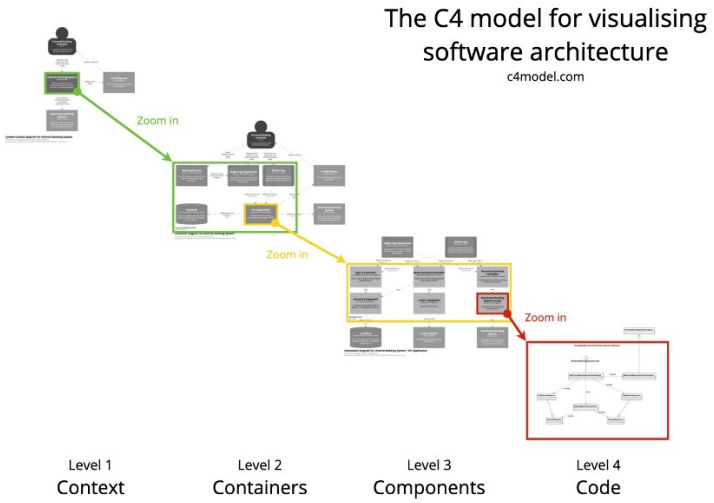
Woche 4
Architekturvorschlag

Ausblick: In Ihrem Angebot mitte November müssen Sie nicht nur die Features ihres MVP beschreiben, sondern auch wie sie dieses MVP technisch bauen wollen. Ihren Architekturvorschlag erarbeiten Sie in diesem Workshop. Spätestens nach dem Workshop erstellt ein Teammitglied bzw. das ganze Team den technsichen Durchstich, um die Machbarkeit abzusichern.

Wir folgen hier bei der Ausarbeitung dem Vorgehen von Simon Brown, dem C4-Model. https://c4model.com/
Wir erarbeiten die Architektur in vier Schritten:

1. Context (= Umgebungsdiagramm / Kontextdiagramm, alternativ Rich Picture)
2. Containers (= Deploybare Einheiten z.B. Client, Server und Datenbank, z.B. als Docker Container)
3. Components (= Aufteilung der Deploybaren Einheiten in Schichten, Frameworks, Patterns)
4. Classes (= Details, hier nicht mehr relevant)

Context, Containers und Components erscheinen auch inklusive beschreibendem Text in ihrem Angebot. Sie dokumentieren die Ergebnisse dieses Workshops zunächst mit einfachen Wiki-Seiten.



1. Umgebungsdiagramm / Rich Picture (ggf. schon vorhanden)

Für ihr Projekt ist es wichtig, dass Sie den Kontext ihres zu bauenden Systems besser verstehen. Daher erstellen sie im ersten Schritt des Workshops ein Kontextdiagramm. Wenn sie alle früheren Aufgaben erledigt haben, müssten sie bereits ein Umgebungsdiagramm haben.

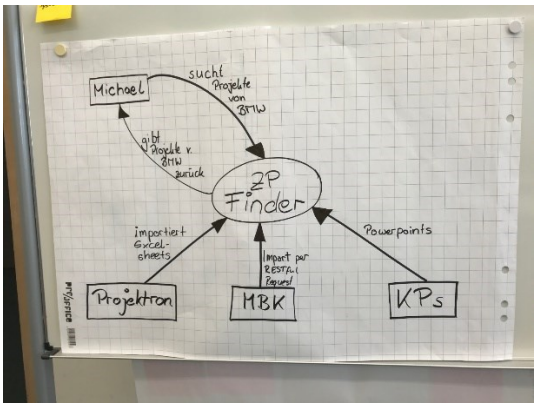


Abbildung 1: Beispiel für ein einfaches Umgebungsdiagramm

Wenn in Ihrem speziellen Kontext ein Umgebungsdiagramm nicht sinnvoll ist, bietet sich ein „Rich-Picture“ an in dem sie den allgemeinen Ablauf beschreiben in dem ihr Projekt eine Rolle spielt. Beispielsweise bauen wir im schaerholzbau-Projekt eventuell keine Software, aber interessieren uns für den Workflow vom CAD-Programm über dessen Export-Funktion bishin zum Import der Daten im Cloud-Service von schaerholzbau.

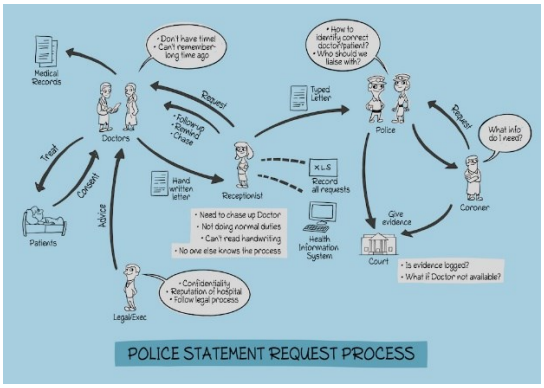


Abbildung 2: Beispiel für Rich-Picture, siehe https://www.betterevaluation.org/en/evaluation-options/richpictures

Die Diagramme erstellen sie im Team, gemeinsam am Whiteboard, ggf. haben Sie als Moderator den Stift in der Hand. Oder Sie delegieren das Zeichnen an ein Teammitglied mit mehr Erfahrung.

2. Deploybare Einheiten und Verteilung definieren

Sie entscheiden im Team über eine gemeinsame Zeichnung (UML-Verteilungsdiagramm, ggf. Box and Arrow + Legende!) in welche Deployment-Einheiten ihr System aufgeteilt wird. Eine recht einfache Aufteilung wäre Client (z.B. NginX+JavaScript), Server (z.B. SpringBoot / Quarkus) und Datenbank. Damit würden sie ihr System in Form von drei Docker-Containern liefern können. Diese Architektur (-sicht) wird auch Verteilungsarchitektur genannt. Hier würden sie bei größeren Systemen zwischen Monolith, EDA und Microservice entscheiden.

In das Diagramm zeichnen sie bitte auch die verwendeten Protokolle zwischen den deployten Einheiten ein, z.B. HTTPS / REST oder HTTPS / gRPC oder …

Achtung: Sobald sie die deploybaren Einheiten entschieden haben, können sie das Build-System / die CI-Pipeline aufsetzen.

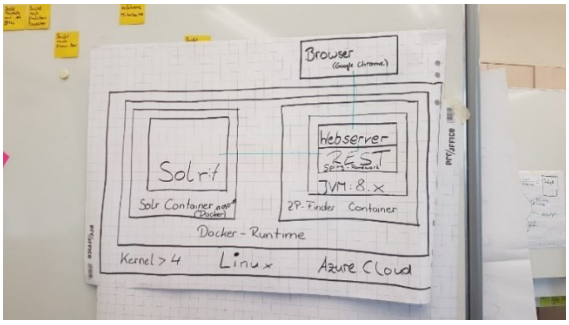


Abbildung 3: Beispiel für eine Verteilungsarchitektur mit zwei Containern: Solr und Webserver

3. Deploybare Einheiten strukturieren

Hier werden wir später (Ende November) noch etwas mehr Zeit investieren. Erarbeiten sie für den Moment mit ihrem Team

1. Welche **Schichten** sollen die deploybaren Einheiten haben? Ein Spring Boot Server hat beispielsweise drei Schichten: Controller (mit REST-interface), Service (User Storys) und Model (die Daten.) oder ihr Client könnte z.B. nach dem MVVM-Muster aufgebaut sein und eine View, ein View Model und ein Model (= Schnittstelle zum Server) haben. Hierfür genügt eine einfache Zeichnung.

2. Welche **Frameworks** sollen verwendet werden. Hier dokumentieren sie als **Tabelle,** den Namen des Frameworks (z.B. Vue, Angular, React oder SpringBoot, Quarkus, Micronaut), die Versionsnummer, die Lizenz (MIT, Apache 2) und die Programmiersprache. Die „kleinen" Bibliotheken z.B. zum Logging oder für Drag&Drop bestimmen sie später.

Um beides besser zu verstehen bietet sich eine besondere Laufzeitsicht an, die „T-Architektur" (vgl. Siedersleben „Moderne Software Architektur, dpunkt, 2004).

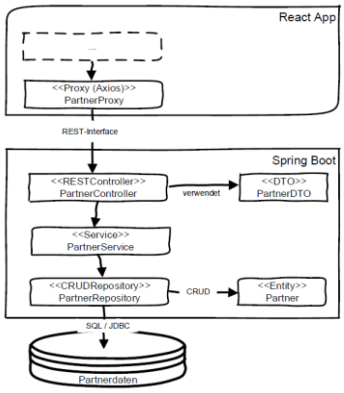
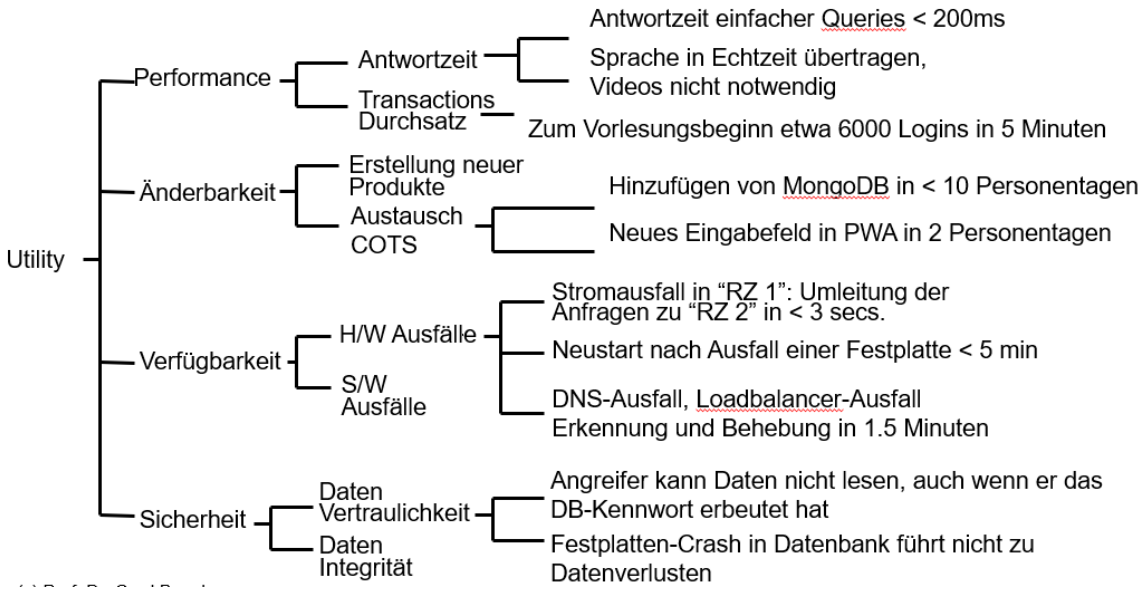


Abbildung 4: Beispiel für eine T-Architektur, dort sind Framework und Schichten erkennbar

4. Brainstorming: Qualitätsanforderungen

Sammeln Sie im Team als Brainstorming Qualitätsanforderungen in Form von Beispielen. Sie haben in den Aufgaben 1. bis 3. bereits Architekturentscheidungen getroffen. Prüfen sie jede gefundene Anforderung noch mal gegen die von ihnen entschiedene Architektur. Ist mit Ihrer Architektur die Anforderung umsetzbar?

Als Moderator:in können sie das Brainstorming über eine Mindmap durchführen lassen. Im Zentrum steht „Utility" oder „Qualitätsanforderungen". Auf der ersten Ebene kommen dann grobe allgemeine Themen wie Performance oder Verfügbarkeit. Die Blätter bilden dann Beispiele, z.B. „den Ausfall des Servers bemerken wir nach 5s". „nach Absturz des Servers ist dieser in 10s neu gestartet", …



5. Technischen Durchstich (Spike, Tracer Bullet) erstellen

In agilen Methoden versuchen wir durch schnelles Feedback die Projektrisiken zu vermindern und unser Wissen schnell zu erweitern. Besonders wichtig ist hier die technische Machbarkeit. Dazu bauen sie die für ihr Projekt

notwendige Infrastruktur prototypisch zuammen. Diesen Prototypen bezeichnen wir als technischen Durchstich. Ab der ersten Zeile Source-Code sollten sie auch den Build-Prozess automatisieren und eine wenigstens rudimentäre Buildpipeline haben.

Erstellen Sie –wenn nicht schon geschehen- einen technischen Durchstich mit Ihrer Zieltechnologie! Ein Durchstich versucht, alle wichtigen technischen Elemente Ihrer Software / Hardware so früh wie möglich zu integrieren. Dieser erste technische Durchstich dient zunächst nur dazu, dass Sie sich mit der Technologie vertraut machen und die Entwicklungsumgebung bei sich auf der Maschine installieren. Als Durchstich genügt zunächst ein besseres „HelloWorld“

- Erste zwei REST-Services mit Symphony oder mit Spring Boot
- Erster Button auf einem Android Telefon, erste Buttons in ihrer React / Angular Applikation
- Erster detektierter Mitarbeiter, der sich mit seinem Smartphone in die Nähe ihres WLAN Knotens bewegt

Ihre Architektur muss noch nicht fertig sein, sie sollten aber die wichtigsten Frameworks entschieden haben. Notfalls bauen sie mit den aktuell heißesten Kandidaten den Durchstich. Wenn sich die Auswahl später ändert, müssen sie den Durchstich entsprechend anpassen.

Wenn es ihre Technologie erlaubt, sorgen sie für eine laufende Build-Pipeline durch die ihr Technischer Durchstich bereits durchläuft. Gitlab bietet hierzu Unterstützung an (.gitlab-ci.yml). Für Projekte, in denen PWA, Server oder einfache Web-Clients entstehen, sollte die Pipeline kein Problem darstellen. AR/VR Projekte benötigen keine Build-Pipeline.