**05\_BAI5-AI-Technische\_Architektur\_Backend**

0:00  
Herzlich Willkommen zur nächsten Videoeinheit. Dieses Mal verlassen wir den methodischen Bereich, wie wir zum Architektur kommen, mit dem wir uns jetzt eine ganze Weile beschäftigt haben, sondern gehen jetzt in Richtung Technik, was vielleicht den ein oder anderen von euch auch mehr gefallen könnte als die reine Methodik. Und dort fangen wir an mit dem Backend. Kurze Erinnerung was ist das Backend oder wo kommen wir da her, das Bild.

0:30  
Einfach gedacht, klassischerweise ein Informationssystem heutzutage werden wird in diesem 3. Tier, wie man sie ja dann eher auch nennt, entworfen, das heißt, typisch auf alle hat man natürlich irgendwo ne Persistenzlösung, unten eine Datenbank.

0:46  
Doch immer damit werden wir uns auch noch mal näher beschäftigen, welche Optionen es da gibt und da drüber liegt dann Big Backend als Bindeglied zwischen dem Frontend oder Benutzer dann mit dem System arbeitet und auf Daten zugreift und im Backend.

1:01  
Da passiert eine ganze Menge. Das hat so einige Aufgaben und da werden wir uns jetzt in dieser Videoeinheit mit beschäftigen, welche Möglichkeiten es gibt, dieses Backend weiter zu unterteilen.

1:14  
Wir uns zwar nicht nur in fachliche Komponenten, die wir uns ja schon angeschaut haben, sondern wirklich in unterschiedlicher.

1:24  
Technischer Aspekt oder wie diese technischen Aspekte dort behandelt werden, können kurze Erinnerungen, technische Architektur des Backends, das gibt wirklich den Rahmen vor, ne, also wir haben, wenn wir von der Methodik her kommen und eine fachliche Architektur im Groben entworfen haben, eine Bausteinsicht, eine fachliche Bausteinsicht, die hier auch mal dargestellt ist, bei der wir letztendlich fachliche Komponenten haben, die in Abhängigkeiten zueinander stehen. Wir haben eventuell sogar.

1:55  
Eine Idee für einen Schnittstellen für Schnittstellen, die wir entworfen haben.

2:01  
Und das gilt es nun aber natürlich irgendwo wirklich in einer technischen Architektur im Backend niederzulegen. Da sind noch ein paar Fragen offen, mit denen wir uns noch nicht wirklich beschäftigt haben.

2:14  
Teilweise schon, aber noch nicht ganz. Das eine war so ein Stück weit, wie bildet man denn jetzt so eine technische Komponente ab, das haben wir schon gesehen, gibt es verschiedene Möglichkeiten, man kann es über Bildunits machen, man kann es aber auch einfach auf Kundenventionale oder konventionelle Arbeit über konventionelle Konventionen machen.

2:29  
Packages oder Textschnamen festlegt.

2:33  
Und wir haben uns auch schon damit beschäftigt, wie kann man diese Komponenten zusammen an Komponenten integrieren. Da hatte ich auch spring IO C, also den Version of Control Container von Springby vorgestellt als eine technische Lösung, aber irgendwo muss man die ganzen Komponenten natürlich auch miteinander verdrahten, die Punkte haben wir schon ein Stück weit adressiert, was aber offen ist, ist letztendlich, wie implementieren wir denn jetzt eine Komponente?

3:04  
Wie kann man die denn weiter strukturieren? Da gibt es ja immer, es steckt ja relativ viel drin, was da erledigt werden muss, und deswegen wollen wir uns da jetzt in dieser Einheit mehr beschäftigen.

3:16  
Wenn ich sage, steckt da relativ viel drin.

3:20  
Können wir uns ja mal kurz vergegenwärtigen, was da die Aufgaben von so einem Back End sind. Hier nochmal das Überblicksbild auch, aber auch noch mal mit einem weiteren Element, was bislang noch nicht drin war, wir uns ja auch später noch mal eine eigene Einheit mehr mit beschäftigen werden, der Systemintegration, trotzdem ist das hier auch schon berücksichtigt werden, dass ein Backend, und das ist ja auch eine der Motivationen gewesen, auch.

3:44  
Schnittstellen bietet nicht nur für die für das eigene System, das sind für das eigene Frontend, sondern auch für Fremdsysteme oder Fremdsysteme integriert. Selber in den eigentlichen Geschäftslogiken, die da implementiert sind.

3:59  
So, wenn man uns jetzt das Bild anschauen, dann sind so ein paar Dinge gekommen, springen da sofort heraus, wofür das Backend offensichtlich verantwortlich ist.

4:06  
Das eine ist erstmal die Funktionalität, die es anbietet, über irgendwelche Schnittstellen nach draußen anbieten. Das ist offensichtlich. Dazu gehört natürlich das Fondant, vielleicht aber auch ein externes System.

4:21  
Und wenn ich über Funktionalitäten spreche, dann ist damit natürlich die Geschäftslogik gemeint. Das heißt, im Backend ist auch typischerweise die Geschäftslogik implementiert.

4:32  
Da gibt es, werden wir auch später beim Frontend sehen, jetzt auch noch verschiedene Trader of Entscheidungen, weil heutzutage natürlich auch die Fronten immer leistungsstärker geworden sind. Da kann man auch bei javascript basierten Frontense oder Touchscreens frontrins relativ gut wartbarer Logiken im Frontend fahren kann und da ist dann immer aber ein Stück weit die Frage, was tue ich jetzt ins Backend, um es auf die Weise auch dort zu Kapseln, nach dem Information hyping Prinzip.

5:02  
So dass die gleiche Logik auch von anderen Fremdsystemen abgelaufen wird und nicht man letztendlich genau diese logisch doppelt implementiert. Und deswegen ist da immer so den Trade auf tue ich manches im Frontend, weil es dort schneller ist. Es läuft ja dann nicht übers Netzwerk irgendeine Logik, die ich erst aufrufen muss, sondern kann direkter implementiert werden, das sind so Fragestellungen, die man da hat, aber klassischerweise eine irgendeine Art und Geld geschäftstheorie wird definitiv ein Backend sein oder muss dort auch sein.

5:34  
Und die Geschäftslogik ist natürlich irgendwie relativ blutleer oder inhaltsarm, wenn das nicht irgendwelche Daten hat, mit denen das arbeitet. Außerdem die natürlich über die Schnittstelle reinkommen, sondern natürlich braucht es dann halt die Einbindung an die Persistenz und Daten zu verbreiten.

5:50  
Und das sind erstmal würde ich sagen, so die offensichtlichen Aufgaben eines Backends, die ihr wahrscheinlich auch so, wenn ihr drüber nachgedacht habt, irgendwo im Kopf gehabt habt, die ich ja glaube ich auch schon so ein bisschen beschrieben hab.

6:02  
Aber passiert noch ne ganze Menge mehr. Das eine wenn wir über Daten verwalten sprechen, dann ist das eine natürlich irgendwie, dass man Daten liest, Daten verändert, Wegschreibt löscht. Ja soweit so gut, aber ein ganz wichtiges Element dabei ist natürlich auch.

6:22  
Sind oder sind dabei auch Transaktionen? Typischerweise möchte man ja Aktionen irgendwie bündeln und diese nach gewissen Kriterien ganze gar nicht festschreiben in der Datenbank und auch das ist Aufgabe des Backends, genau diese Transaktionen zu verwalten da.

6:42  
Noch mal näher mit beschäftigen.

6:45  
So neben den Transaktionen ne muss ich natürlich auch irgendwo die Daten herstellen, dass ich dass die gültig sind, das heißt ich muss auch eine Validierung gewährleisten, sowohl syntaktische Validierung das was irgendwo vom Frontend oder anderen Systemen kommt, dass das den Regeln der Schnittstelle entspricht, als auch semantische Überprüfungen, semantische Regeln, die in Kombination bei den Aktionen auftreten, also fachliche Regeln, geschäftsregeln, die dort implementiert werden müssen. Ich muss auch sicherstellen, dass die Sicherheit gewährleistet ist. Es soll ja nicht irgendjemandem beim Backend aufrufen können und Informationen rausklauen oder Unfug betreiben, sondern.

7:22  
Nur Benutzer, die das dürfen, das heißt, ich muss sie authentifizieren, bedeutet, ich muss sicherstellen, dass das diejenigen sind, die das ganze System benutzen dürfen, und ich muss auch prüfen, wenn ich das sichergestellt hab, sind wir haben diese Benutzer auch das Recht überhaupt ne, es ist ein super User.

7:39  
WRK System Approven darf oder nicht, muss ich prüfen und sicherstellen. Ansonsten kann ja irgendein Sachbereiter auch einfach seinen Auftrag, weil er schnell das machen will. So ein Einkauf approven, ohne dass er es recht hat.

7:51  
Und wenn man sie Thema Sicherheit sind, ist auch ein Thema natürlich was passiert und Fehler ne. Also ich muss ja auch sicherstellen, dass Fehler sauber abgearbeitet und abgehandhabt werden. Es darf halt nicht passieren, dass Fehler einfach letztendlich.

8:05  
Durchgereicht werden und das Backend wegen einer Exception abstürzt, das ist der Worst case. Es muss aber auf der anderen Seite auch ein paar andere Punkte berücksichtigt werden, die wir uns gleich auch noch mal anschauen werden und daneben passiert sogar noch mehr. Ne, also ich muss auch irgendwo natürlich den Betrieb sicherstellen, das heißt ich muss auch Monitoring und Logging irgendwo vereinen, kann in dem Backend bis hinzu natürlich der Integration von Drittsystemen, auch das muss ja irgendwo gewährleistet sein und auch das muss ich sicherstellen, ihr seht, es ist eine ganze Menge an Aufgaben die das Backend.

8:39  
Und die man irgendwo natürlich dann auch verorten muss innerhalb des Backends, welcher Teil des Backends das Buch für verantwortlich, damit ich bei Fehlern oder bei Punkten genau weiß, wo ich reingreifen muss und auch Änderungen durchzuführen und auch überhaupt die Arbeit sinnvoll zu gestalten.

8:55  
Wir werden uns jetzt ein Paar von diesen Aufgaben ein paar wichtige Aspekte mal rausgreifen und da mal näher darauf eingehen und auch konzeptionelle grundlegende Prinzipien dort mal darlegen.

9:04  
Das eine ist die Sicherheit, ne, also Sicherheit gehört einfach an die Grenzen des Backends. Ich muss letztendlich natürlich, bevor irgendjemand da etwas innerhalb des Backens aufruft, eine Barriere bauen, die sicherstellt, dass derjenige authentifiziert ist und auch die Autorisierung muss geprüft werden, dass derjenige, wenn ich sicherstelle, dass er es ist, über Passwort oder wir werden später auch mit Awards andere Mittel kennenlernen.

9:30  
Wenn das sichergestellt ist, auch das rechte Zeug. Und Recht ist auch etwas, was man auf 2 unterschiedlichen Weisen prüfen kann.

9:37  
Das eine ist, hat er das Recht, diese Operation aufzurufen, also erst recht eine bestimmte Aktion durchzuführen. Das andere ist datenabhängige recht, die sind nicht ganz so einfach zu verstehen, das ist im Prinzip dann, dass ich nur bestimmte Daten sehen darf, wenn ich ne laseroperation aufrufe.

9:52  
Zum Beispiel nur die Einkäufe aus meiner Abteilung oder in dem Hochschulinformationssystem vielleicht nur die von meinem Department. Die Vorlesung auf der anderen Weise Art und Weise muss ich natürlich auch sicherstellen, dass die.

10:06  
Auch beim Schreiben, dass ich zum Beispiel nicht den Vorlesungsplan von einem anderen Department einfach überschreiben darf. Ne, das sollte ich dann nicht dafür vielleicht sehen, aber nicht ändern. Das sind so Aspekte, die man da berücksichtigen muss, dann muss ich auch.

10:19  
Genau diese Punkte halt transparent handhaben, dass man wirklich dort gar nicht als Entwickler ein Fehler machen kann, sondern das ist idealerweise irgendwo in einer Schicht.

10:28  
Um den um den Kern drumherum abgefangen wird typischerweise auch konfigurierbar, dass man wirklich festlegen kann, welche Dinge welchen Regeln unterliegen und was wo abgeprüft wird.

10:41  
Und ich kann das machen, indem ich zum Beispiel auch standardifizierte Authentifizierungsprotokolle nutzen kann. Früher wurde viel Basic Authentication genutzt, mit Username und Passwort, das ist nicht mehr zeitgemäß, heutzutage nimmt man meistens ROOS 2 oder Open ID als Erweiterung davon werden wir uns später im Kapitel der Systemintegration nochmal näher anschauen.

11:02  
Und typischerweise macht man das über wirklich user Informationen, die an dem User dran ist. Also irgendwie ne Rolle n Attribut was da dran hängt, wie zum Beispiel zu einem Department zuweist.

11:13  
Soweit zum Thema Sicherheit. Ein weiteres Thema. Wie gesagt waren das Thema Transaktionen ne ist ja auch sicherstellt, dass die Daten sich ganz schlüssig sind und bei den Transaktionen ist es wichtig, dass wir da erstmal unterscheiden zwischen wirklich.

11:26  
Fachlichen Transaktionen, das ist das, was der Benutzer darunter begreift, wenn er einen Auftrag bearbeitet, ist das gesamte von ihm eine fachliche Transaktion wichtig, üblicherweise.

11:37  
Uns eine fachliche Transaktion, unterteilt in mehrere technische Transaktionen, eine technische Transaktion, wo ich ihn lese, den Auftrag dann bearbeitet der da was in seinem Frontend und dann speichert er, und das passiert in einer weiteren technischen Transaktion.

11:50  
Und technischer Transaktion. Damit sind dann wirklich die Transaktionen von der Persistenzlösung gemeint, also bei einer relationalen Datenbank quasi deren die Transaktion, die da ist mit Asset Kriterien, also atomal konsistent.

12:04  
Halt mit einer Isolierung gegenüber anderen Transaktionen und dass die Daten, wenn es abgeschlossen ist, auch gesichert und persistent sind.

12:13  
Wichtige Eigenschaften, die einem viel bringen, werden wir uns auch noch mal näher mit beschäftigen. Dann in der Einheit zur Persistenz Ebene, darunter zur Persistentschicht das.

12:22  
Bieten aber diese technischen Transaktionen.

12:26  
Und üblicherweise so, wenn ich ein Backend aufrufe, passiert diese transaktionssteuerung transparente Hintergrund, man öffnet quasi auch manchmal, ohne dass man es sehr stark steuert, einfach direkt eine Transaktion.

12:40  
Und bedeutet alles, was darunter abläuft, wenn es wirklich eine erste Transaktion, das ist dann für sich gesichert, wenn es einen Fehler gibt, wird auch alles zurückgerollt, man hinterlässt dadurch die Daten nicht in einem konsistenten Zustand, was einem viel Kopfschmerzen.

12:56  
Oder viel Kopfschmerzen verändert Kopfzerbrechen, wie man sonst hätte, wenn man diese Eigenschaften nicht hat.

13:02  
Typischerweise kann es natürlich trotzdem sein, dass man Konflikte hat, ne, da werden wir uns auch noch mal in der Persistenzlösung damit auseinandersetzen, wenn jemand parallel gleich einen Auftrag bearbeitet, kann es ja zu Konflikten kommen und da gibt es verschiedene Strategien, wie man damit umgeht, was auch sehr wichtig ist. Transaktionen können halt wirklich geschachtelt sein, man kann Transaktionen halt Schachteln, indem man dann 2 subtransaktionen öffnet und das macht man zum Beispiel, wenn man die eigenen Daten speichert, bevor man.

13:34  
Eine andere System aufruft und zusammen ergeben sie an die übergeordnete Transaktion ist eine Möglichkeit, das Ganze noch komplexer zu gestalten. Das kann aber auch relativ sehr oder relativ schnell sehr problematisch werden, wenn man den Überblick verliert und auch dadurch einen komplexen Geschäftsfällen halt auch in einen Deadlock laufen kann, wo sich die Transaktion gegenseitig bremsen.

13:59  
Dann kommen wir zum Thema.

14:03  
Fehlerbehandlung und bei Fehlerbehandlung hatte ich schon gesagt, es ist wichtig zum einen, dass man vermeidet, dass das System abstürzt und das Backend gar nicht mehr verfügbar ist. Ja.

14:15  
Der zweite wichtige Aspekt ist dabei, dass man auch sinnvolle Fehlermeldungen nach draußen gibt und nicht so etwas wie das hier.

14:23  
Das ist halt etwas, was tatsächlich hier vielleicht auch mal das ein oder andere mal selber gesehen hat. Ich hab es auch schon mal bei dem ein oder anderen Programm gesehen, woran Stack Tracy oder ähnliches, das ist n Haube und n Haufen gibberisch, den man gar nicht so richtig versteht.

14:35  
Der da zurückgegeben wird.

14:38  
Und wenn das passiert, dann ist üblicherweise die Reaktion gerade von so einem Experten da. Von einem Informationssystem sitzt der gerade eine halbe Stunde mühsam einen Auftrag bearbeitet hat, mit Absprachen, telefonisch, nebenbei und allem dran, und das Eingeklimpert hat doch speichern drückt und das dann hochkommt, der reagiert dann üblicherweise so, also die sind dann nicht sehr erfreut, da kann man dann auch davon ausgehen, dass da Telefone, Anrufe kommen in den Support bin ich sehr freundlich sind, das heißt man muss auch hier Fehler wirklich architekturell sinnvoll adressieren und berücksichtigen und dazu gehört erst einmal auch eine Unterscheidung von verschiedenen Fehlerarten oder Typen.

15:17  
Wichtig ist halt, dass man wirklich erst mal unter zwischen spezifisch spezifizierten Zeiten und unspezifizierten Verhalten spricht. Also ich spezifiziere das Verhalten ist das, was man halt wirklich implementiert hat mit einem bewussten Spiel, weil es so gefordert wurde.

15:31  
Wo es natürlich n normales Ergebnis gibt, was zurückgegeben wird, aber auch spezifizierte Fehler n doppelter Username zum Beispiel oder ein Einkauf übertritt ne bestimmtes Limit oder solchen Geschichten und unspezifiziertes verhalten kann dann zu unspezifizierten Fehlern führen. Das sind typischerweise Dinge, wo vielleicht die Datenbank oder das System nicht verfügbar sind, wo aber letztendlich auch vielleicht Programmierfehler einfach vorliegt und der dann zu einer Exception führt. Das sind Dinge, die da passieren können und man kann die dann halt unterschiedlich zurückgeben. Also man kann dann wirklich.

16:07  
Das normale Ergebnis und Rückgabewertung kurz ein spezifischer Fehler ist ne Spezi spezifizierte Exception, die auch gefangen werden muss. Von dem Aufruf war, weil er dann auch agieren muss und kann ne wichtig er muss es auch können wenn ein anderes System nicht verfügbar ist. Was so ein Programmierer dann da implementieren kann man nicht viel machen.

16:26  
Und unspezifizierte Fehler sollten auch unspezifische Exceptions sein, also Runtime Exceptions im Java File, die halt auch nicht gefangen werden müssen, die da durchfliegen.

16:36  
Genau.

16:39  
Und wichtig, man braucht dann halt eine transparente Behandlung in so einer Art Sicherheitsfassade vor dem eigentlichen Backend LPRI die dann halt Fehler die Hochpoppen erstmal die auffängt ne es dekoriert quasi das Backend und.

16:52  
Stellt sicher, dass das ganze generische Fehlermeldungen zurückgegeben wird und dass das auch übertragbar ist. Der Fehler über Jason oder sonst was ist ja keine Stack Tracer oder sonst was rüber fliegt. Man kann das dann auch erweitern, dass man zum Beispiel eine Fehler ID mit gibt, womit der Benutzer sich an einen Support findet wenden kann, der in den Log Meldungen dann auch genau den Ablauf findet und direkt irgendwie vielleicht auch Auskunft geben kann. Das sind so Aspekte die man da nutzen kann, das waren jetzt mal so.

17:24  
Ein bisschen Deep Dive, wo ihr seht, welche technischen Feinheiten man eigentlich in einem Backend berücksichtigen muss und ihr merkt daran, dass das ganze kann durchaus ein bisschen komplizierter sein, sprich, wir wollen die Komplexität reduzieren und das Backend unterteilen in verschiedene Teile, um halt diese ganzen verschiedenen Aufgaben, wo ich jetzt hier auch nur ein paar detaillierter dargestellt habe, sinnvoll zu platzieren und zuordnen.

17:47  
Da hab ich jetzt mal folgenden Überblick vorbereitet, wo wir uns durchhangeln werden in diesem Teil der Vorlesung. Und zwar geht es zum einen darum, diese technischen Aufgaben, die ihr eben gesehen habt, sinnvoll zu verteilen.

18:01  
Und da gibt es 2 Ansätze, die relativ populär sind. Sind sicherlich nicht alle. Es gibt wahrscheinlich noch mehr, aber das sind die, die ich zu einer Praxis meistens vorfinde. Das eine ist die Schichtarchitektur, wo man halt üblicherweise mehrere Schichten hat, die in einer bestimmten Aufreihenfolge sind, das Thema hatten wir auch schon bei dem Architekturprinzip.

18:23  
Und es gibt dann das Thema Hexagonale Architektur oder die Alternative dazu, wo man auch eine Art Schichtung hat, aber eher so wie ihr seht, mit einem Kern in der Mitte und so eine Adapterschicht drumherum. Beiden gemein ist, dass man halt geschäftslogik irgendwo implementiert hat und auch klar gesagt hat, wo die Supplementieren ist. Bei der Hexagonalen Architektur im Kern hier bei der Schichtenarchitektur in der gleichnamigen Schicht der geschäftslogik Schicht und dann kann man aber sich die Frage stellen, wie strukturiere ich jetzt wieder um die Geschäftslogik und auch da gibt es 2 unterschiedliche Sätze Ansätze.

18:58  
Das eine ist so Use Case basiert, indem man einfach Use Cases hat, die auf so ein domänenmodell arbeiten, was meistens relativ oder einfach nur ein Datenmodell ist, ist ein relativ einfacher Straight Forward Ansatz den wir kennenlernen werden und ein anderer Ansatz ist das taktische, die Idee wo man versucht das ganze wesentliche Objektorientierter zu adressieren, das sind so die 2 wesentlichen Punkte und da werden wir uns jetzt hier so Schritt für Schritt durchhangeln, damit wir dieses ganze Bild einmal gesehen haben, dass wir auch ein Verständnis für entwickelt haben.

19:29  
Wie man das Backend strukturieren kann.

19:32  
Gut.

19:36  
Dann.

19:38  
Fangen wir mit der Schichtenarchitektur an. Ihr seht hier oben so ein bisschen Überblick, wo wir uns da befinden. Die Schichtenarchitektur unterteilt jetzt das ganze Backend in verschiedene.

19:49  
Schichten in einem glücklicherweise Fassaden oder Serviceschicht in eine Logikschicht und eine Data Access Schicht ne Fassadenschicht bietet Schnittstellen nach außen an, sowohl zum Frontend als auch zur anderen Backend Systemen.

20:03  
Und ruft dann die Logikschicht auf. Und da ist die Geschäftslogik drin und damit die implementiert werden kann, ruft sie Schnittstellen an, die die Datenzugriffsschicht zur Verfügung stellt, die wiederum den Zugriff auf die konkrete Datenbank oder Datenbanklösung Kapselt.

20:18  
Dadurch hat man eigentlich schon mal eine grundlegende Verteilung der Aufgaben, wie wir sie auch schon mal eben gesehen haben bezüglich Datenmanagement bezüglich.

20:27  
Exponisation oder Exposionierung von irgendwelchen Schnittstellen bis hin zur Geschäftslogik, da seht ihr, das ist schon mal relativ klar.

20:33  
Und man sieht auch, man hat eine klare Aufrufabhängigkeit immer von oben nach unten, wo das Ganze dann entsprechend stattfindet.

20:43  
Es gibt dann aber im Detail durchaus unterschiedliche Interpretationen. Was gehört Logic Logic Schicht, wobei?

20:51  
Manche sagen in der Logikschicht, weil da das eine fachliche Entscheidung sein sollte und manche sagen, es sollte in der Fassadenschicht sein, es möglichst einfach und transparent zu haben. Das sind so wie sollen die Schnittstellen hier draußen formuliert werden, ist das im Wesentlichen.

21:05  
Wird das hier gemacht, dass man explizit noch mal ein eigenes Datenmodell definiert oder macht das auch die Logikschicht und die Fassadenschicht oder Serverschicht? Ist wirklich nur ne reine?

21:16  
Reine Technik dass man es über Netzwerkaufrufbar macht. Wo packt man die Sicherheit, also welche Autorisierungssachen getan werden sollen, da gibt es unterschiedliche Ansätze, wie man das letztendlich dann verteilen kann und es gibt auch unterschiedliche Schichtenarchitekturen natürlich, das ist jetzt die eine Interpretation, die wir bei uns auch viel in unseren Projekten angewandt haben.

21:36  
Dass die Idee sieht, eine andere Beschichtung vor, die habe ich hier mal aufgeführt. Das ist auch Teil von einem Tactical DWD. Ich habe hier mal so das Bild was später noch mal kommt auch mal das nochmal gekennzeichnet, da hat man die User Interface, das ist natürlich nicht Teil des Backends, also das Frontend und dann das Backend ist unterteilt in so eine Applikationsschicht wo man im Prinzip den Zugriff auf die Domänenschicht bereitstellt und Domänenschicht ist ein Stück weit die die reine fachliche Logik, die das reine fachliche Modell.

22:06  
Praxissystem Applikation schneidet das für den Sinne der Applikation? Welche Funktionalitäten werden wir nach draußen?

22:14  
Gestellt hält auch keinen Zustand oder ähnliches. Das wird ist wirklich eine Art Zugriffschicht, wie wir es eben gesehen haben. Irgendeinen anderen Namen wie ihr seht und darunter sieht man so eine Infrastrukturschicht, die auch von der User Interface genutzt wird, wo technische Querschnittsdienste zur Verfügung gestellt werden, UI Library, Persistenz oder ähnliches ist eine Alternative, die ich jedoch Vollständigkeit halber auch mal darstellen sollte, aber auch dem ist gemeint, dass man halt immer eine klare Aufrufbedingung natürlich hat.

22:45  
Und dann kommen wir zu guter Letzt auch noch zu der hexagonalen Architektur, die momentan noch sehr beliebt ist und auch bei uns immer beliebter wird.

22:56  
Ich sitze hier noch mal im Überblick. Das ist eine zweite Möglichkeit, technische Aufgaben zu verteilen ist auch ein Stück weit eh nicht zu einer Schichtenarchitektur, insofern, dass man hier drin das als eine Schicht bezeichnen kann, mit einer äußeren Schicht drumherum, deswegen wird es häufig auch so Anion Architecture genannt, also Zwiebelarchitektur mit verschiedenen Schichten im Kern.

23:14  
Gesagt, ganz klar Verortung der Geschäftslogik hier drin ist und die Idee bei der Hexagonalen Architektur ist auch das Prinzip der Trennung von Zuständigkeiten wirklich auf die Spitze zu tragen und zu sagen, dieser dieser Application Corso idealerweise gar keine Technik haben.

23:30  
Nichts. Ne, das ist so ein Stück weit der die Idee kann man natürlich auch im im Sinne eines Trade Offs ein Stück weit variieren, inwiefern man das wirklich so durchsetzen will, sind zum Beispiel springoritation Technik oder nicht, das ist so eine Frage.

23:44  
Die man, wo man häufig gesagt hat, das ist eigentlich egal, das ist fast schon das, was wir nullsoftware ja auch nennen nach dem Softwarekategorien. Das sollte eigentlich akzeptiert sein, aber heutzutage Cloud Architektur und ist das zum Beispiel gar nicht mehr so. Springen ist halt nicht so performant, im Startup zum Beispiel eine Spring Applikation Springboot zu starten, das dauert noch schon mal 10 Sekunden oder so, das ist in einer skalierbaren Welt, wo man mehrere Instanzen von einem Backend laufen lässt, in einem Cloud flexibel, die auch mal abrauchen dürfen und einem Failover nicht mehr akzeptabel da.

24:20  
Wirklich schnell sein. Deswegen sind Dinge wie Quarkos geboren, als alternative Komponenten, Rahmen, rahmen die Tiere, wenn ihr wollt, die auch den Java Code wirklich in nativen Code übersetzen, wo man.

24:34  
Guten Startup Zeiten von unter einer Sekunde hat dabei so ein unter einer halben Sekunde und teilweise sogar unter 100 Millisekunden. Das geht rasend schnell, das sind aber alle Rednerionen und andere Dinge, das heißt, wenn man da flexibel wechseln können möchte auch in Zukunft, wer weiß was die Kürze umbringt, dann müsste man da sagen, dass es dann nicht mehr teils application.

24:57  
Umgeben wird der dann von dieser Adapterschicht und die Adapterschicht unterteilt man auch in 2 Unterschichten sozusagen oder teilt die in der Mitte durch. Das eine ist die Primary oder Driving Adapters.

25:09  
Oder incoming Adapters eingehen Adapter, das heißt das sind die Sachen, die von außen unseren Application Core aufrufen wollen über Rest über WSDL, über Graved oder über eine Message, die reinkommt oder selber durch einen Batch gesteuert, was auch immer.

25:24  
Das kann alles Mögliche sein, da ist man flexibel und bietet für jede dieser Dinge einen eigenen Adapter an, der dann einen Port aufruft. Vom Port ist dann letztendlich ein Nichterseits ein Schnittstelle angeboten und implementierte Schnittstelle von Application Core, in der eingehende Adapter dann aufruft, relativ simpel und entspricht in dem Sinne, wenn man das jetzt die Service Schicht wäre und dass die Logikschicht so der Schichtenarchitektur.

25:55  
Relativ straight Forward, deswegen würden wir auch mal von Adaptern und.

26:03  
Der wesentliche Unterschied zwischen Schichtenarchitektur kommt jetzt in dem zweiten Teil, in dem Secondary Driven oder Outgoing Adapters, wo das wirklich die ausgehenden Aufrufe sind. Das heißt, wenn ich jetzt Daten mir holen will, das heißt, der Chor selber braucht auch an Daten aus anderen Systemen.

26:17  
Er will eine Nachricht verschicken oder ähnliches. Dann ist da der Unterschied, dass der Chor auch in diesem Fall einen Port definiert, also eine Schnittstelle definiert, aber wirklich nur definiert, das heißt, er exportiert nur die Schnittstellen Definition.

26:31  
Also eine angeforderte Schnittstelle. Und diese Schicht gibt dann die Implementierung davon ne und auf die Weise seht ihr, dass die Abhängigkeitsrichtung immer nur von außen nach innen sind.

26:44  
Das ist der wesentliche Unterschied zu zu der Schichtenarchitektur und man kann auf die Weise auch wesentlich flexibler das ganze gestalten. Man ist immer nicht nur auf Persistenz als Ausgehendes beschränkt, sondern da kann alles mögliche sein und dadurch kann man es auch beliebiger erweitern.

26:59  
Das Hexagon als das Teil des Namens ist einfach nur wegen der Darstellung gewählt. Bitte nicht dran denken, dass man jetzt irgendwie nur 6 Schnittstellen, die ihr dargestellt haben sollte, sondern da gibt es beliebig viele. Ne, das ist ganz wichtig zwischen und ihr merkt dadurch ne durch diese Flexibilität und da starken Trennung von Zuständigkeiten bietet das Muster gegenüber einer Schichtenarchitektur schon noch mal so ein paar Vorteile, die es sehr beliebt machen. Unter anderem findet ihr Netflix hier so einen schönen Artikel.

27:29  
Der veröffentlicht wurde, wo Netflix mal dargestellt hat, wie es ZB.

27:35  
Eine Geschäftslogik aus einem bestehenden Monolithen Rausgelöst hat extrahiert hat, und zwar wirklich nur die Logik und dann halt mit Adaptern gearbeitet hat, wo sie erstmal noch die alten Schnittstellen aus dem alten Monolithen aufgerufen hat und die Daten zu speichern und zu holen. Später hat man dann die Daten migriert, eine eigene Datenbanklösung und musste nur den Adapter anpassen auf einen db Adapter um dann halt die Daten der eigenen Datenbank zu speichern und so konnte man flexibel das ganze evolutionär migrieren.

28:04  
Und das zeigt so ein bisschen die Flexibilität, auch später in in einem weiteren Entwicklung des Backens, die man durch diese starke Trennung von Zuständigkeiten bekommt.

28:15  
Nachdem wir jetzt die 2 grundlegenden Arten kennengelernt haben, wie wir die technischen Verantwortlichkeiten verteilen können, ne mit Schichtenarchitektur und einmal die exagonale Architektur, gucken wir uns jetzt an, wie man die Fachlichkeit da drin strukturieren kann und fangen wir auch hier erst mal mit dem einfachen an und gleichen Kombination mit der Schichtenarchitektur, damit jemand ein Gesamtbild bekommt, wieso ein Backend Struktur ausschauen kann, was ihr hier seht ist im Wesentlichen.

28:44  
Die Komponenten, die wir gebildet haben, der fachliche Komponentenschnitt und darüber legen sich jetzt die verschiedenen Schichten, also die Datenzugriffsschicht.

28:55  
Dort hat man dann üblicherweise auch das Datenmodell abgebildet mit MTTS, die sich dort drin sind und man die Entities zu bekommen. Ne, also um die zu holen aber auch zu schreiben und zu speichern.

29:08  
Führt man in der Regel noch ne weitere Subjekte ein oder ein weiteres Element, das ist das sogenannte Dow Data Access Object. Beides werden wir uns gleich auch noch mal genauer anschauen, dass letztendlich diese Entity verwaltet und darüber liegt dann wie gesagt die Geschäftslogik und in diesem Use Case Driven Ansatz ist es jetzt einfach die Geschäftslogik in Use Cases bekapselt, das heißt man versucht.

29:33  
Kohärente Zusammengehörige Funktionalität in einem Artefakt zusammenzugliedern da was dann eine Schnittstelle draußen anbietet, da ist im wesentlichen Dinge sein können, wie zum Beispiel einen Auftrag platzieren.

29:47  
Einen Auftrag stornieren, das kann sein, dass man.

29:51  
Keine Ahnung, Lagerhaltung optimieren möchte oder den Lagerstand kontrollieren möchte. All diese Dinge sind.

29:57  
Mögliche Use Cases in unseren Hochschulinformationssystemen werden es halt Dinge wie ein Praktikum anlegen, einen neuen Lehrplan erstellen, für ein Semester in Lehrplanverwaltung den Mensaplan anlegen und all diese Dinge, das sind einzelne Teile, die typischerweise gebündelt werden. Meistens ist so ein Use Case Artefakt dann halt auch ein Stück mehr, dass man dort versucht die gemeinsame Funktionalität zu tun, rein zu tun, also gerade wenn es um die Verwaltung zum Beispiel von The Mensaplan geht, dass man.

30:28  
Anlegen kann, sondern dass man ihn auch löschen kann und verändern kann, lesen kann.

30:33  
Da drüber gibt es dann die Fassadenschicht oder Serviceschicht, teilweise auch genannt, wo man Services hat als Artefakte, die dort platziert sind und die auf die Use Cases zugreifen und die nach draußen und die Außenwelt exponieren.

30:47  
Und üblicherweise würde man jetzt nicht einfach einen Use Case aufrufen und die Entitäten da rausholen und die Identitäten zurückgeben an das Frontend und ähnliches, weil das letztendlich.

31:00  
Verletzungen ein Stück weit das Geheimnis Prinzip ist, als auch Probleme darstellt, technischer Natur häufig, weil die Entität ja ein viel größeres Geflecht sein können und man halt da auch ein Stück weit die Kontrolle verliert und deswegen gibt es da drum hier auch noch mal sogenannte Datentransferobjekte einzuführen, die letztendlich ein eigenes Datenmodell darstellen, was nach außen gegeben wird, das ist ein Stück weit querschnittlich über beide Schichten, insofern, dass beide Schichten das verwenden, teilweise werden die Daten.

31:33  
Data Transfer Objects nur in der Fassadenschicht hier gemappt und genutzt und die Geschäftslogik fungiert nur auf Authentizitäten oder funktioniert nur auf Entitäten, wird darin implementiert.

31:45  
Teilweise gibt es aber auch Ansätze. Wie gesagt, es gibt Unterschiede, wie man die Aufgaben verteilt, dass das Mappen von Entitäten diese Datentransferobjekte unten geschieht auch die Geschichte, wie kann ich datentransferobjekte modellieren, welche Traders gibt es da werden wir uns gleich noch mal genauer anschauen, das ist aber im wesentlichen dann für eine Komponente schon alles, das sind die Artefakte wie man das Backend für eine fachliche Komponente strukturiert und das macht man dann halt bei jeder Komponente konsistent und ähnlich, so die Idee.

32:17  
Wie spielt das Ganze zusammen? Ich habe es eben schon umgangssprachlich gesagt, hier mal ein Beispiel.

32:23  
Von.

32:25  
Einem von einem anderen Beispielsystem, was in den vorherigen Vorlesungen verwendet haben, wo es darum geht, dass man halt eine BG Verwaltung hat für die Wohngemeinschaft und dort könnte man zum Beispiel so einen BG Service haben, da könnte man alle WG s finden, der hat das einfach an einen BG Maintenance Use Case weitergegeben um die zu holen.

32:46  
Dass da hat dann.

32:49  
Wurde dann als nächstes aufgerufen. Hat die Datenbankabfrage gemacht und das Ganze ging dann zurück und hier in dem Schnitt ist es dann so, dass der WG Service dann letztendlich das Mapping gemacht hat.

32:59  
Das ist jetzt ein simpler Fall hier zu veranschaulichen. Natürlich kann man hier oben in dem Service noch weitere Orchestrierungen stattfinden.

33:08  
Genauso auch hier in dem Maintenance Use Case kann noch mehr stattfinden und hier können auch andere Systeme zum Beispiel aufgerufen werden. Über AP is auch das ist möglich.

33:20  
Genau und DTOS auch hier wie gesagt unterschiedlich gehandhabt. Teilweise wird es auf Use Case Ebene gemacht, teilweise auf Service Ebene und.

33:28  
Üblicherweise macht man diese DTLDTOS. Das sind häufig sehr ähnliche Abbildungen von Identitäten, aber halt schon noch mal eine Abstraktion davon, teilweise aber wirklich baugleich. Wir wenden uns jetzt im Folgenden die einzelnen.

33:44  
Artefakte auf den verschiedenen Ebenen anschauen und hier seht ihr oben jeweils auch, wo wir uns befinden.

33:52  
Hier jetzt bei Identitäten ein Daten zudurchschicht und wir seht Entitäten in diesem Modell sind üblicherweise ein reines Datenmodell. Hier seht ihr keine Methoden, sondern wirklich nur Attribute und Assoziationen zu vielleicht anderen Entitäten oder zu Kind Entitäten und in der Regel wird das auch so eingehalten. Das ist die simpelste Regel, dass man sagt, da ist keinerlei Geschäftslogik und man legt es wirklich rein anämisch, wie man letztendlich sagt an, also blutleer. Es ist wirklich ein reines Datenmodell, trotzdem haben.

34:23  
Die natürlich einen Lebenszyklus.

34:26  
Durchaus mal angelegt, verändert, gelöscht werden, haben auch dazu häufig eine Identität, mit der man die natürlich finden muss, sowohl in fachlichen Schlüssel und auch.

34:39  
Lebens, weil der fachliche, dass er vielleicht zu komplex ist oder zu aufwendig zu verwalten oder sich vielleicht auch mal ändern kann, wie zum Beispiel ein E-Mail-Adresse.

34:51  
Genau und bezüglich des des Anämischen Datenmodells vielleicht auch noch als Ergänzung. Es gibt auch dort Abwandlungen, dass man zum Beispiel sagt, ne Entität.

35:03  
Kann Methoden anbieten und damit auch Logik, sofern es zum Beispiel das lawof.de Mieter einhält. Also wirklich nur auf den eigenen Attributen plus auf den Methoden von den benachbarten oder assoziierten Entitäten, dann arbeitet.

35:18  
Aber nicht weitergeht, kann man so definieren. Ich find es halt ne pragmatische Lösung um es auch ein bisschen objektorientierter zu gestalten, aber es ist nicht nur so ne ganz simple Regel, das heißt Entwickler müssen es auch verstehen, müssen es richtig anwenden. 2. Neue Fehlerquelle ist halt auch hier ne Trade off Entscheidung wie weit man da gehen möchte.

35:38  
Was ist jetzt mit den Assoziationen zu Entitäten von anderen Komponenten? Könnt ihr euch selber mal die Frage stellen, wie würdet ihr das handhaben?

35:47  
Zunächst einmal muss man feststellen, es ist halt notwendig, ne, wir haben halt einen häufig irgendwie Beziehungen, was weiß ich sei es zu zu irgendwie einer Kundenobjekt oder das heißt zu einem zu einem Lager in dem.

36:01  
Ne ne Ware liegt im Produkt liegt es gibt einfach fachlich durchaus Situationen, das lässt sich einfach nicht von Hand weisen. Typischerweise versucht man es aber dann halt nicht so zu machen auf der Ebene um das nicht eng zu koppeln. Ansonsten habe ich ein riesiges Datenmodell über alle fachlichen Komponenten, dass man sagt man entkoppelt es durch eine Referenz und das kann halt wie gesagt diese technische ID sein oder halt ein fachlicher Schlüssel und darüber habe ich das dann auch wirklich entkoppelt. Alternativ kann man auch drüber nachdenken dort extra.

36:32  
Interfaces einzuführen für Entitäten zum Beispiel Read only, die hat also keine Setter anbieten oder keine Methoden, die Daten verändern, einfach um die Datenhoheit zu gewährleisten. Die Datenhoheit ist einer der Hauptgründe, auch wie soll man das versucht zu vermeiden, das ist schon eine relativ enge Kopplung, nicht nur von dem Wissen, was Überschwappt, sondern auch über die Möglichkeit, wirklich Daten ineinanderitäten zu ändern, die eventuell dann auch untergeschrieben werden auf Basis von Veränderungen, die aus den anderen Komponenten kommt, worst Case sinaio.

37:02  
Man hat völlig den Überblick verliert, wer eigentlich welche Daten ändert. Das ist ja gerade hier die Idee bei der Komponentenorientierung, dass man diese strikte Trennung und kapselung halt einschätzt.

37:15  
Gut, kommen wir zu den Artefakt. Die Frage war ja, wie komme ich jetzt an die Ethik dran, damit ich damit arbeiten kann und das passiert halt über dieser Data Access object patterns ne.

37:27  
Da habe ich dann letztendlich.

37:30  
Einen dao winterface ne.

37:33  
Mit den üblichen Methoden wie zum Beispiel Find ID, find All find Buy Update und Lead. Das sind so die klassischen Craft Methoden und das wird dann irgendwo implementiert und nutz bei der Implementierung irgendeine Persistenz Library wie zum Beispiel High by Nate, Eclipse Link Hypernate enden im dotnet Bereich und so weiter und sofort Ziel ist es von diesem Pattern, das ist wirklich ein Entwurfsmuster, dass man die Persistenz auch austauschen könnte. Also wenn ich einen anderen datastore habe, Low xq ldatenbank vielleicht sogar im extremen Fall oder einfach nur einen Dateisystem wo ich irgendwie Dateien ablege, dann kann ich das halt mit dem Pattern austauschbar machen, hab dann.

38:12  
Flexibilität welche Lösung ich wähle.

38:15  
Ist auch meistens gegeben. Man muss aber auch da aufpassen im im im Detail.

38:23  
Hat man dort, muss man aufpassen, dass die Operation, die man anbietet, weil das sind hier so die Standardoperationen, aber die werden natürlich erweitert durch bestimmte Queries, ne, wenn ich bestimmte Aufträge aus einem Land haben will, oder ich möchte alle offenen Aufträge, die mir zugewiesen sind.

38:37  
Das sind Currys, die auf der Datenbank abgelaufen werden. Und das kann üblicherweise nicht durch jede Implementierung.

38:45  
Performant implementiert werden. Da muss man dann schon schauen, ob das überhaupt möglich ist. Das ist die einzige Einschränkung bezüglich der Anwendbarkeit, würd ich sagen.

38:54  
Die Vorteile, die sich dadurch ergeben, sind wirklich die, dass man diese Flexibilität bekommt. Man hat auch eine klare Trennung von Zuständigkeiten, ne hier die Simplementierung kümmert sich genau um die konkrete und persistenzlösung und im Corpelt dann davon Nachteile.

39:11  
Ist ein Stück weit, dass der ER mapper ja eigentlich auch schon so etwas versucht. Wenn man jetzt wirklich relationale Datenbank hat. Aber da liegt genau der Punkt ne. Also wenn ich jetzt nenne nen Airmaper nutze und auch ne relationale Datenbank, dann fühlt sich teilweise das also wie man so schön sagt umgangssprachlich doppelt gemoppelt an als würde man so 2 Abstraktion hintereinander und in gewisser Weise ist das auch so, weil der er Mapper ermöglicht es mir so ein halber net auch die konkrete relationale Datenbank einfach auszutauschen, das sind aber 2 unterschiedliche Ebenen wenn ich wirklich versuche dann halt eine ganz andere Lösung dahinter zu tun.

39:46  
Wird es dann nicht ganz so einfach, dann würde ich auch keine ehrenamtlich mehr einsetzen.

39:52  
Dieses Data Access Object Pattern ermöglicht es auch ganz schön, den Aspekt zu adressieren, den wir bisher noch nicht so genau angeschaut haben oder nicht im Detail geschaut haben, nämlich diese datenabhängige Autorisierung. Das lässt sich Dortmund nicht ganz schön verankern und auch transparent behandeln.

40:13  
Und das ist so eine schöne Erweiterung, finde ich.

40:17  
Dort Verankere in diesem Dow, dann liefert mir diese Stau ne wenn ich styleplementiert habe auch nur die Entitäten zurück, die ich als angemeldete User überhaupt sehen darf oder verändern darf. Ne, das ist letztendlich so die Idee, das heißt ich muss das irgendwie hinkriegen, dass wenn ich hier ein find all aufrufe oder auch ein Find bei id ich, dass dieses Dao prüft, ob ich dazu überhaupt berechtigt bin, das ist da letztendlich die Idee und das kann man halt wirklich generisch implementieren.

40:47  
Und das hatte ich auch schon in der Vergangenheit gesehen. Häufig macht man das zum Beispiel, indem man bei so einer Entität halt auch eine Interface definiert, dass diese Entität dann implementieren kann, das gibt dann halt so ein Get restricted Attribute zurück, das heißt, das Attribut, was sagt, wie es eingeschränkt ist, also ein bestimmtes zum Beispiel in den Abteilungskürzel könnte das sein als Wert ergibt die Sensität zurück, dagegen kann ich dann prüfen, ob ich selber.

41:15  
Das Recht habe auf diese Abteil oder zu dieser Abteilung gehöre und damit auch die Sentität sehen darf.

41:21  
Dazu muss die Implementierung allerdings natürlich durch diese querschnittliche Element zugreifen, so eine User Session Provider ne, also irgendwie ein Singleton muss da Zugriff drauf haben, das kann man in Frameworks ganz gut implementieren.

41:35  
Das heißt, das ruft das auf und holt sich die Usersession und die Usersession gibt dann halt die Attribute zurück, die auf die der Benutzer das Recht haben kann.

41:43  
Und dann kann ich halt zum Beispiel bei find by d, ich hol mir das Objekt aus der Datenbank, ich frag das Objekt, was ist dein restricted Attribute und ich kann dann halt auch bei diesem in der User Session die die die die die erlaubten Attribute abfragen vergleicht ob das passt. Wenn es passt gebe ich es zurück. Ansonsten gebe ich Neid zurück oder werfen Exception, das ist dort letztendlich die Idee.

42:07  
Und man kann es sogar soweit generisch implementieren, dass man wirklich Currys erweitert. Also wenn ich jetzt hier einen find all habe, kann ich in dieser down Implementierung immer automatisch eine worklos Antwort hinzufügen, eine Bedingung in der SQL Abfrage zum Beispiel oder auch bei anderen Lösungen ähnlich wo ich genau diese Suchparameter einschränke und aus dieser Session die erlaubt Attribute hole und das dann einschränke das Attribut auf der Entität wo es gespeichert ist. Also den Attribut, dass das irgendwo in dieser Liste von erlaubten Abteilungskürzel jetzt in diesem Beispiel ist.

42:44  
Das funktioniert so lange gut, solange man halt ein bestimmtes Set auch nur erlaubt. Ne. Also wenn es jetzt keine Ahnung üblicherweise Benutzer hat immer nur eine Abteilung zu der zugehört, da gibt es vielleicht ein paar super User, die haben vielleicht 1020 Abteilungen, geht auch noch, dann ist das jetzt eine SQL, zum Beispiel eine Einschränkung einer Inklusion mit 20 Attributen, das kriegen die meisten Datenbanken dann auch noch mit entsprechenden Indizes gut hin, schwieriger wird es dann, wenn ich das wenn das mehr wird muss.

43:15  
In der Praxis auch tatsächlich mal passiert. Bei einem Kunden hatten wir den Fall.

43:21  
23 Jahre, nachdem wir das eingeführt haben, das System, das es schon im Betrieb war, das ist ne ne Änderung gab in der Organisation, dass die so Shared Service Center eingeführt haben, das waren dann 3 Shared Service Center weltweit.

43:33  
Diese Standardaufgaben im System übernommen haben, das heißt, das Waren typischerweise dann Mitarbeiter, die hatten Rechte auf wesentlich mehr.

43:45  
Abteilungen auch nicht nur Abteilungen, auch teilweise unterschiedlichen Ländern.

43:50  
Sogar Abteilungen, so dass dann die Query die Resultierende plötzlich riesengroß wurde. Wenn das dann passiert, dann ist man dagegen natürlich nicht gewappnet, dann kriegen plötzlich die Performance in den Chat Service Center in den Keller, hat teilweise auch sogar Performance Auswirkungen auf die anderen Abteilungen gehabt, das ist so ein Beispiel wo so eine Intransparenz halt auch.

44:11  
Damit rechnet man nicht mehr unbedingt ne, weil das lief alles. Das läuft dann im Hintergrund ab, das haben nicht alle immer auf dem Schirm und plötzlich merken, ah ja, das gab es ja auch noch, diese diese generischen Implementierung und dann muss man halt da entsprechend tätig werden und ne andere Lösung finden oder die Lösung erweitern haben wir auch in den Griff bekommen, aber das ist mal so ein Beispiel aus der Praxis wo halt so Magic die im Hintergrund passiert auch vielleicht mal unerwartete Probleme in der Produktion im Betrieb verursachen kann.

44:42  
So, und wir kommen jetzt zum eigentlichen Kern, den Muse Cases, also der, wo wirklich die Geschäftslogik implementiert ist. Und ihr seht es in dem Beispiel, die das Prinzip dahinter ist relativ.

44:55  
Einfach, das sind wirklich halt ne Bündelung von von Geschäftslogik in einem ad Fakten namens Use Case, das typischerweise zusammengehöriger.

45:06  
Operationen mit hoher Kohäsion bündelt.

45:10  
Zum Beispiel Halt, wenn man Auftrag anlegen möchte oder auch den Auftrag mit Default werten für den Benutzer oder den Kunden anlegen möchte. Statussendungen machen will. Bündelt man das dann in einem Use Case, so dass idealerweise fachliche Regeln, die zusammen gehören, auch dort drin implementiert sind. Das ist die hohe Kohäsion, weil diese fachlichen Regeln, und das ist dann wichtig, dass man das gemeinsame Verständnis auch mit dem Fachbereich hat, sollten dann halt auch gemeinsam geändert werden oder in Beziehung stehen, das ist

45:41  
und dazu gehört es dann natürlich auch, diese diesen Use Case wirklich.

45:47  
Gut zu strukturieren und zu schneiden ist eine Designaufgabe an sich. Das ist häufig das, was dann halt nicht ganz so akkurat gemacht wird. Häufig wird es doch relativ datenzentrisch gemacht, da kommen wir gleich noch dazu, was vielleicht nicht ganz immer dem besten Vorgehen entspricht, man sollte es wirklich versuchen in dem Sinne zu bündeln, wie es halt auch der Fachbereich machen würde, wo der Fachbereich auch gemeinsame Änderungen sehen würde, das wäre ein guter Schnitt.

46:15  
Der Use Case ist an sicher, dass man auch gar keine technischen Abhängigkeiten nutzt. Diese Dao Schnittstelle, das ist erstmal frei von irgendeiner Technik, das sind einfach reine Methoden die er aufrufen kann um Entitätsobjekte wirklich zu bekommen oder er nutzt Halt Use Cases von anderen Komponenten oder auch Services und können die dann entsprechend orchestrieren und kann dort in dieser Ebene natürlich das Ganze auch noch dynamischer gestalten, gerade wenn die Geschäfte relativ dynamisch dynamisch sind und vielleicht auch mal sich ändern.

46:47  
Sollen, ohne dass man jedes Mal dann die Implementierung ändert, kann man auch über sowohl engions nachdenken.

46:55  
Das ist dann, das sind regeln, die man halt irgendwo in einer Domain specific language halt auch konfigurierbar im System ablegt.

47:02  
Und die man auch zur Live Zeit ändern kann. Ein Beispiel dafür sind so.

47:09  
Entscheidungsbäume, die man zum Beispiel modellieren kann oder Entscheidungstabellen, wo verschiedene Bedingungen geprüft werden zur Laufzeit und man auf die Weise zum Beispiel bestimmen kann, welches welche Abteilung es für einen Auftrag zuständig und.

47:26  
Kann dann entsprechend das Routen. Das wäre so ein Beispiel dafür, das kann man dynamisch machen, dann kann man es auch dynamisch kultivieren.

47:35  
Mehr gibt es dazu aber eigentlich gar nichts zu sagen. Das heißt, ihr seht an diesem sehr einfachen Vorgehen, wie man Use Cases strukturieren kann oder wie man die Geschäftstheorie strukturieren kann. Das ist letztendlich etwas, was es wirklich stark vereinfacht, auch für den Entwickler.

47:49  
Klar, den Schnitt für Use Cases zu finden ist noch mal ne Herausforderung. Dann allerdings arbeiten die Use Cases rein auf Datenmodellen von Entitäten und die Logik wird dort implementiert. Das Vorgehen ist ein relativ straight Forward.

48:02  
Und man muss nicht so viel darüber nachdenken.

48:06  
Nicht so viel Designaufwand reinstecken hat aber auch den Nachteil, dass es natürlich nicht sehr expressiv ist. Es ist nicht wirklich eine Anwendung von Objekten. Orientierung auf der Ebene und nutzt dadurch auch nicht die Vorteile, die sich da ergeben, die wir später dann noch einmal beim Tactical Etiketten.

48:24  
Zum Abschluss kommen wir jetzt noch mal zu den Services hier oben in der Fassadenschicht.

48:30  
Zumindest was die Schichtenartefakte angeht. Und dort sieht ja so ein Service, das ist dann halt eine dedizierte Schnittstelle für verschiedene Clients oder auch für andere Backends oder so für andere als Systeme und üblicherweise beinhaltet so ein Service auch so eine transparente Zugriffsschicht Service Access, hier mal genannt, die vielleicht auch von Framework rein injiziert wird, die der Entwickler sogar nicht zwingend mitbekommen, da kann man mit dem Java Bereich mit Aspekte Programmierung viel machen.

49:00  
Bei anderen Frameworks für vielleicht andere Möglichkeiten. Das wird dann quasi injiziert und anstatt das direkte die Plimitierung aufgerufen wird, wird halt erst der Service Access aufgerufen und dort kann man auf die Weise halt auch ein Paar von den Querschnittlichen belangen, die ich auch schon erwähnt hatte und Aufgaben dort fahren kann. Man kann hier zum Beispiel so ein grundlegendes Logging einführen, dass man sieht okay hier wird dieser Service von dem Benutzer aufgerufen zu dieser Uhrzeit und mit welchen Attributen und hat so schon mal ein grundlegender Login Funktionalität oder?

49:32  
Security ja üblicherweise, dass man prüft, ist der Benutzer auch authentifiziert, dann hat er vielleicht auch dem Recht diesen Service und diese Operation aufzurufen.

49:41  
Bis hinzu den Transaktionen, die dort teilweise angesteuert werden. Das heißt, da kann man Transaktionen technischer erstellen und sagen, alles was jetzt da drunter passiert, zumindest bei einer relationalen Asset.

49:56  
Irgendwo n Problem gibt Nowback kann ich alles zurückrollen und hab keine inkonsistenten Zustand und dort tritt es häufig auch drin. Die serialisierung ne wenn ich das ganze nach draußen zum Frontend oder aber zu einem anderen Fremdsystem geht und muss das selber in einem Netzwerkleitung, das heißt dort findet halt auch wirklich die Anbindung die technische Anwendung auf der Netzwerkebene statt, üblicherweise aber durch Frameworks implementiert, das ist so das was der Service dort letztendlich macht, ansonsten Delegierte mehr oder weniger nur auf ein oder mehrere Use Cases nachdem er diese technischen Aufgaben abgehandelt hat.

50:29  
Um das auch mal ein Stück weit anschaulicher zu machen und auch nochmal auf ein paar der Herausforderungen, die ich ganz am Anfang der Einhaltung genannt habe, mal einzugehen in meinem Detail hier mal wieder das Beispiel aus dem Springframework wie man da so eine Zugriffsschicht realisieren kann und dass man ein Beispielservice.

50:48  
Einen Service, den würde man dann auch hier mit dieser Serviceorganisation als Java Bean deklarieren. Dadurch ist man den Komponentenrahmen bekannt, man kann ihn dann halt auch in anderen.

50:58  
Beams, die für Springen bekannt sind, halt einfach einwirken mit Transactional, wenn dann der.

51:08  
Der IOC Konzerns kennt und das ICE Control ist, kann man mit Transaction einfach sagen. Alle Methoden die aufgerufen werden werden Gewappt in einer technischen Transaktion.

51:18  
Das ist etwas mehr muss man da auch nicht machen. Man führt nur diese Annotation hinzu. Man muss noch ein paar Konfigurationen im Hintergrund machen, die typischerweise von von Dee Dee Developern oder der Architekten gemacht werden, den Victor selber fügen einfach nur diese Anstation hin und per Magie fast im Hintergrund, bevor die Methode wirklich ausgeführt wird, die ich da drin habe. Die Public Methode wird eine Transaktion, sofern sie noch nicht existiert, geöffnet und am Ende der Methode auch geschlossen, bis hinzu robec verhalten, was ich im Fehlerfall bei einer Exception auch dort sein kann.

51:49  
Dass ich sagen kann, wenn ein Rollback passiert, möchte ich bitte, dass das zurück oder wenn Exception hochfliegt, dann möchte ich bitte, dass das Zurückgerollt wird, was da passiert ist. Da haben wir hier diesen Customer Service und um die Anbindung oder die Exponierung als Rest Service zu machen kann man hier mit so einem Postmapping zum Beispiel arbeiten wo man sagt unter API Customer ist das steht das zur Verfügung, ich kann dort mit Pre authorize wirklich prüfen ob er auch die das Recht dazu hat diese Methode aufzurufen.

52:19  
In dem Fall Edit ne dieses Update customer. Irgendein Objekt wird Reingegeben ist jetzt hier mal simplifiziert dargestellt.

52:27  
Dann passiert dort die Delegation und die Use Cases und dann ist das ganze dann entsprechend abgehandelt und man kann dann auch hier fehlerbehandlung transparent handhaben, indem man Exception Händler definiert, ne?

52:41  
Und auch einschränkt auf bestimmte Exceptions, wenn die Auftreten. Dass diese Exception plant, da aktiv wird und dann.

52:52  
Man dort eine transparende Fehlerbehandlung machen. Es sind eine generische umwandeln, man kann dort ein RID erzeugen, die RID loggen.

53:02  
Und ähnliches. Üblicherweise macht man das schon vorher, dass man wirklich halt so eine Corelation ID immer hat fürs Logging, die immer mitgeloggt wird, die dann aber auch an der Stelle zurückgegeben wird, das heißt in die Fehlermeldung an den Client zurückgeliefert wird und so sieht dann halt oder kann ein Service dort aussehen. Dazu gehört aber auch eine gewisse Konfiguration, nicht nur bezüglich der Datenbank, die habe ich jetzt hier mal nicht aufgeführt, aber bezüglich der Security, wie kenne ich denn da Sicherheit gewährleisten und das macht man auch hier bei so einem Configuration Beam. Das ist auch ein Java Beam The Configuration Assistiv Spring Framework bekannt gemacht.

53:39  
Und dort kann man zum Beispiel dann mit Enable Global Method Security genau das hier aktivieren. Dass das möglich ist mit diesem Pre authorized zu arbeiten, mit Web Security aktiviere ich generell die Security und dann kann ich dort eine Security Konfiguration definieren.

53:56  
Indem ich von einer von einer so einem standardadapter Erbe und dort halt entsprechend eine Methode überschreibe die configured Methode, die HTTP security dann halt erweitert.

54:07  
Und da kann ich dann so nach einem großen API sagen, zum Beispiel für HTTP mit dem Session Management, das möchte ich statemess halten und ich möchte auch ein Request autorisieren und zwar alle ne und sagen alle.

54:21  
Brauchen mindestens die Autorität Display.

54:26  
Ansonsten wird jeder Request Deniz und ich kann auch zum Beispiel auch aus 2 relativ dort einfach dort konfigurieren. Da gehört noch ein bisschen mehr dazu im Hintergrund, aber in der Regel ist seht ihr, dass es eine relativ einfache Konfiguration programmatisch liegt, vor dem kann um Halt so eine Security zu konfigurieren und dann letztendlich diesen http.

54:49  
Zugriff, der über Springer auch ene wird, wird überhaupt auch abzusichern, dass da nichts Dummes passiert, solange zumindest ein springcan Buckes gut.

55:00  
Das so einmal rundum durch diese ganzen verschiedenen.

55:07  
Ein Begriffe und.

55:11  
Damit ihr da ein Bild entsprechend habt.

55:15  
Dann können wir jetzt noch zu dem letzten Artefakt, was eine gewisse Relevanz hier hat, nämlich die Data Transformcts. Die Idee bei den DTOS ist üblicherweise, dass man versucht, von den konkreten Datenmodellen innerhalb der Komponente zu abstreben.

55:30  
Und so wirklich auch Schnittstellen sauber benutzerorientiert.

55:37  
Für den, der es verwendet, aufzubereiten, mit einem eigenen Datenmodell, sodass man nicht immer das komplette Datenmodell, was man intern hat, nach draußen preisgibt und nicht weiß, was die anderen damit machen. Das ist zu viel wissen, was nach draußen definiert.

55:51  
Neben der Schnittstelle selber von einem Service, den man vielleicht dann nutzt, ist halt auch das Dto wirklich eine Abstraktion. Dann eine wichtige Teil der Schnittstelle, das Datenmodell, was überhaupt noch draußen geben will, ist im Wesentlichen auch wieder die Entität, nichts anderes als ein reiner Datencontainer dient auch dem Transfer der Daten, das heißt bei dem Dto kann man dann zum Beispiel den Aspekt auch berücksichtigen, dass die Daten signalisierbar sein sollten, das ist vielleicht nicht überall der Fall, wenn ich dann halt bestimmte komplexe Daten habe mit irgendwelchen.

56:23  
Character Lobs von irgendwelchen relationalen Datenbanken oder ähnlichem, dann kann das vielleicht problematisch sein und je nachdem was ich dann auch benutze.

56:31  
Um die Daten auf die Leitung zu bringen, ne, also im Fall von der Rest API vielleicht irgendwie so n jsonmapper, dann muss ich da auch vielleicht annotationen nutzen um genau das Signalisierbar zu machen und diesen Aspekt dieser Aufgabe kann ich dort in das DTO verlagern.

56:47  
Es soll also sein.

56:49  
Silber sein soll auch keine Logik enthalten, auch keine Methoden. Die kann das Frontend ja auch nicht nutzen, es sei denn man hat wirklich irgendwie so eine native Method Interface von Java. RMI japaces glaube ich.

57:02  
Womit man wirklich Objekte übergeben konnte, das ist tatsächlich möglich, aber da braucht man auch die gleiche Klasse. Auf der anderen Seite und sonst hat jetzt andere Nachwirkungen, also in der Regel Logik macht dort keinen Sinn, das ist ganz ganz klar und sollte auch wenig Abhängigkeiten haben zu irgendwelchen Services oder Use Cases, sondern sollte sie eigentlich auf Dtos beschreiben und sollte halt auch ganz oder gar nicht befüllt sein beziehungsweise es soll klar definiert sein im Dto. Welche Attribute kann derjenige der es nutzt erwarten?

57:33  
Dass sie auch noch gesetzt sind, damit man wirklich weiß, was man in der Hand hat.

57:39  
Und hier auch noch mal so ein paar Punkte. Wieso nicht? Wieso nicht einfach die Entität nach draußen neben dem Aspekt, dass man halt das im Koppeln möchte?

57:47  
Das ist hier der Punkt. Ne. Also man möchte wirklich die Komponenten Schnittstellen entkoppeln von der Umwelt und wenig wissen nach draußen geben, ne auch die Datenhoheit behalten n ganz wichtiger Punkt bin ich beim DTO irgendwie n Setter Aufrufe oder sowas interessiert die Komponente aus dem das stammt gar nichts. Bei der Entität hatten wir bei dem Prinzip Datenhoheit, bei den Horrorstiken kann es dann wenn man halber net nur wirklich dazu führen, dass das auch in der Datenbank persistiert wird man auch irgendwie volle Kontrolle über das was nach draußen geht. Über einen Datentransfer kann dadurch auch ein Stück weit steuern, dass nicht zu viel Daten rüber gehen, die Interformant sind oder ähnliches.

58:23  
Man kann auch wirklich die Persistenz im Top, also gerade ist eine Entität von halber net, die dort gemanagt wird, die kann so Dinge enthalten wie lazy Collections, also Listen von assoziierten anderen Entitäten, die erst dann aufgelöst werden, wenn man darauf zugreift.

58:40  
Dazu wird halt wirklich dann ne Praxis Proxy davor geschaltet, transparent über.

58:49  
Die das ermöglichen, das sieht der Entwickler nicht. Das sind versteckte Details, die aber dann spätestens wenn man es rübergibt, halt Probleme verursachen kann, weil vielleicht die Serialisierung nach Abschluss der Transaktion passiert und man dort gar keine Abfragen an die Daten machen kann. Das sind so einige der Gründe, wieso man das entsprechend machen möchte und da kommt es jetzt natürlich darauf an, wie ist eigentlich das Dto und.

59:14  
Dort gibt es eigentlich 2 verschiedene Herangehensweisen. 2. Extreme sage ich mal von einem Trade off her, nämlich das eine ist rein Datenmodell orientiert, eine Entität ist ein 1 dto. Man hat da 11 Mapping von den Anti teats Attributen, hat dadurch eine Sicht auf die Daten allerdings auch wenig Kontrolle.

59:34  
Auf der anderen Seite hat man auch wenig Aufwand dahinter. Das Ganze einfach kann man relativ einfach generieren und hat trotzdem Abstraktion. Im Notfall kann man immer noch von der bei der Entität was anders machen und das tio so lassen.

59:48  
Die Freiheit hat man dann, das heißt, Entkopplung ist auf jeden Fall da etwas besser. Es ist ein Use Case orientiert, das ist ein Stück weit eher in die Richtung von expliziten Schnittstellen, wenn ihr euch erinnert bei den Schnittstellenarten wo man dann.

1:00:04  
Im Prinzip pro Use Case eine bestimmte dt. Os wirklich spezifisch definiert, maßgeschneidert und performant. Für den Use Case nur das was er wirklich benötigt, vielleicht auch mit komplizierteren Mappings und benötigt dabei ein explizites Design dafür und natürlich dadurch auch ein Stück weit mehr Aufwand.

1:00:21  
Es gibt so ein Stück weit nen Versuch Mischform davon auch zu zu hinzubekommen und das sind die Data Transfer Structures, DTS teilweise genannt.

1:00:30  
Oder halt auch Composite DTOS, teilweise also 10, CTOS oder CDTOS. Ich hab es alles schon gesehen, das ist n Stück weit n Mittelweg. Man hat zum Beispiel das Modell N Kunde der ne Adresse hat als zweite Identität, dann gibt es 2 DTOSN Customer DTO und n Address DTO und das Customer Address dts verknüpft die beiden zusammen und auf die Weise habe ich so ein 11 Mapping, teilweise kann ich auch ein Subset wählen von den von den Daten der Identität.

1:01:01  
Und was man auf jeden Fall nicht macht, sind die Assoziationsanalytik, natürlich zu merken auf der Theorie eben. Das macht keinen Sinn.

1:01:09  
Und die DTS? Die können selber auch noch mal abgeleitet, Attribute zum Beispiel enthalten, und so habe ich eine relativ große Flexibilität und das sollte man aber auch wirklich nutzen. Also ich bin von dem da rein Datenmodell orientierten Vorgehen, das ist gefährlich, einfach auf dem Grund, weil das dazu tendiert auch zu wachsen, die Entitäten wachsen und es wird dann immer einfach die die To s erweitert, obwohl es für manche Use Cases gar nicht relevant sind, dann wird man dts sein, die dts erwachsen also.

1:01:39  
Das Design eines Datenmodells für die Schnittstelle ist eine explizite Designaufgabe.

1:01:45  
Im einfachen Fall für einfachen Komponenten, Stammdaten oder sowas kann das relativ einfach sein, aber ich würde davor warnen, dass wirklich immer nur ohne das Gehirn irgendwie einfach stumpf zu generieren, dann braucht man es eigentlich auch fast gar nicht, diese Abstraktion und ich habe tatsächlich auch Fälle in der Praxis gesehen, wo man dann irgendwo in einer anderen Komponente DTS übergeben wird und das DTS ist riesig, enthält auf oberster Ebene allein schon 10 Attribute und 3 davon sind irgendwie assoziations zu weiteren dts und das gefälligst dann weiter aus über.

1:02:16  
Ebenen und der Entwickler, in dem ein Use Case den ich dort mal reviewed habe, braucht tatsächlich irgendwo auf Ebene 3 oder 4 nur ein einzelnes Attribut plus ein Attribut auf oberster Ebene, mehr nicht. Bis er das gefunden hat und dass du Datenmodell verstanden hat.

1:02:31  
Ist eigentlich nur, dass das überhaupt funktioniert, aber von der Praxis klappt es dann trotzdem irgendwo. Aber es ist natürlich nicht das, was man haben möchte, es ist nicht wirklich lose Kopplung, viel zu viel wissen diffundiert danach außen. Also ich kann euch nur mit auf den Weg geben, denkt darüber nach, wenn ihr zwischen verschiedenen Komponenten oder auch mit Kontext arbeitet, da sägst du zieht im Design auch das Datenmodell zu sein, um eine wirkliche Entkopplung auch hinzubekommen.

1:02:55  
Wir sind jetzt in dem eigentlich auch eine ganz gute Überleitung. Jetzt sprechen wir schon darüber, wie kommunizieren eigentlich Komponenten miteinander und was geht da jeweils drüber und das können wir uns ja auch noch mal anschauen, was für Abhängigkeiten es da wirklich gibt oder erlaubt sein sollten und das sind hier im Wesentlichen auch nur die folgenden, das heißt, ihr habt immer eine Komponente A und Komponente B und üblicherweise ist es so, dass man natürlich nur in einer Richtung eine Abhängigkeit haben möchte, hier von A nach B. Und das sollte idealerweise auch nur von dem Service zu dem anderen Service sein, das ist dann so lose gekoppelt wie es irgendwie geht, denn man kennt nur die Datentransferobjekte von anderen Objekte, die Use Cases da aus Entitäten alles versteckt.

1:03:34  
Auf jeden Fall lose Kopplung.

1:03:37  
Und kann man so verstehen. Es gibt Fälle, wo man drüber nachdenken kann, das Ganze auch enger zu koppeln.

1:03:44  
Indem man zum Beispiel auch zulässt, dass man die Use Cases oder Use Cases untereinander aufruft und auch die Identitäten teilweise übergibt. Dann geht man sehr schnell in eine enge Kopplung und da muss man vorsichtig sein, ob man das wirklich möchte oder ob das Sinn ergibt.

1:04:01  
Kann ich nur verwarnen das zu extensiv zu betreiben. Dann verliert man eigentlich die Kontrolle über das Design.

1:04:09  
Aber auch hier gibt es halt wie gesagt verschiedene Spielarten, wie man das Macht.

1:04:16  
Gut.

1:04:18  
Zum Abschluss von diesem Case Driven.

1:04:24  
Schichtenarchitekturansatz vielleicht nochmal auch hier so eine Heuristik, die auch relativ häufig benutzt wird. Ich würde sie kritisch hinterfragen, ob sie wirklich sinnvoll ist. Aber auch hier zur Gesamtübersichtlich auch so haben wir teilweise sind wir vorgegangen beim Design, dass wir gesagt haben, wenn ich jetzt wirklich versuche die Komponenten zum Beispiel zu identifizieren.

1:04:47  
Dann kann ich sagen Okay der Server ist jetzt erst mal am Anfang eine Komponente und ich versuche dann in diesem Server unterschiedliche Gruppierungen von Entitäten und Use Cases zu haben. Also wirklich Use cases, Entitäten irgendwie zusammenführen, also sehr Daten modellgetrieben auf der Weise auch und sage.

1:05:05  
Wenn ich diese Gruppierung habe, OK, jede Gruppe ist erstmal eine Komponente.

1:05:10  
Und dann kann es Fälle geben, wo die Zuordnung noch nicht ganz klar ist. Das eine Entität vielleicht mal zu der einen Gruppe oder zu der anderen geht. Da muss uns vielleicht noch mal verfeinern und irgendwann ist man so halbwegs final. Dann legt man die Abhängigkeiten fest zwischen diesen Komponenten, das kann auch irgendwo sein, dass man dann Zyklen hat, weil man dann doch die eine Information noch aus der anderen Komponente hat und das bedeutet dann aber ein Zyklus und dann muss man noch mal ran und das anders kopieren und Anstrukturieren und auf die Weise durch mehr Reparation kommt man irgendwann zu einem zyklischen.

1:05:43  
Regraphen von Komponenten und hat dann letztendlich.

1:05:49  
Einen Komponenten Design, das ist anders als wir es jetzt gemacht haben, mit strategischen Dlb und Eventstorming. Aber etwas, was halt auch relativ Straight Forward geht, wenn man schon mal so ein fachliches Datenmodell zur Hand hat oder eine Idee davon hat. Auf der Weise kann man auch so fachliche Zusammenhänge versuchen anhand von Assoziationen und Gemeinsamkeiten die Lebenszyklus zu sehen.

1:06:09  
Kann dann diese architekturprinzipien Separation of concernce, Minimierung der Abhängigkeiten, Homogenität Datenurhalt ist ne leicht große Gruppen hat anwenden um so zu einem Komponenten Architektur zu kommen.

1:06:21  
Und man muss dann aber immer halt auch prüfen, ob die Zuordnung Sinn ergibt. Kann ich die Verantwortlichkeit von einer Komponente in ein 2 Sätzen beschreiben, ist immer so ein Lackmustest, den man am Ende machen kann und auch die die Abhängigkeiten sind halt dann eine letzte Validierung ob das klappt und wenn ich es dann noch eine dynamische Laufzeit sehe, dann heranziehe und dann halt zu gucken wie in der Haupt Use Cases läuft das gut ab dann habe ich eine einigermaßen Sicherheit aber.

1:06:47  
Wichtig dabei, das führt dann wirklich zu einer sehr datenzentralen Entwurf, wo man sich sehr vom Datenmodell treiben lassen lässt. Das ist etwas, wo das strategische DDDN Stück weit flexibler oder auch anders denkt, sondern eher vom Prozess her kommt. Ne von diesem Big Picture eventstorming and zu Ende Prozess dort halt der dynamischen Sicht während dem Ablauf, der in der realen Welt passiert, dort unter Gruppen zu finden und nicht so sehr auf das Datenmodell zu schauen, sondern ganz im Gegenteil, Ärzte sagen.

1:07:18  
Ich habe vielleicht sogar dasselbe Datenobjekt mehrfach, aber mit anderen sichten, mit anderen Modellen dahinter, in unterschiedlichen Kontext.

1:07:27  
Aus meiner Sicht stellt sich so in der Praxis immer mehr heraus, dass das das bessere Vorgehen ist. Ich wollte euch trotzdem auch hier mal so einen anwesenden Alternative dort mal vorstellen, weil es halt viele auch praktiziert wurde, teilweise auch praktiziert wird. Was ich aber finde, sollte man kritisch hinterfragen, kann aber auch durchaus mal Sinn ergeben, weil es vielleicht auch relativ schnell und einfach ist, das auf die Weise zu machen.

1:07:52  
Gut.

1:07:54  
Wir werden dann in der Präsenzveranstaltung weiß noch nicht, ob in der nächsten, nächste Woche, vielleicht auch in der übernächsten, uns auch mal so ein Beispiel anschauen. Ne wo man so den Service usw Becken mal macht. Das ist hier von devener B und so unserem Framework, was wir im Open Source gestellt haben, was genau so eine Architektur, wie ich sie dargestellt habe auch grob implementiert, da sind die Verantwortlichkeiten auch ein bisschen klarer definiert, ein bisschen strenger, da wird sehr viel in diese Logikgeschichte auch verlagert, die auch die GTO s definiert und alles nach draußen kippt, wo auch Transaktionen und Rechte teilweise gemanagt werden.

1:08:28  
Aber es ist eine Interpretation der Schichten Architektur, und da haben wir so ein Mi fi Star, also eine Restaurantverwaltung, mal gebaut als Beispiel Anwendung, wie das aussehen kann, und da haben wir zum Beispiel auch folgende Bausteinsicht.

1:08:42  
Dann letztendlich, und genau die werden wir uns dann aber auch noch mal gemeinsam anschauen, mal durch den Code gehen, gemeinsam Präsenz, damit ihr auch noch mal Fragen stellen kann und auch mal wirklich ein Beispiel real sieht.

1:08:58  
Gut.

1:09:02  
So viel erst einmal dazu. Jetzt haben wir auch das Use Case getrieben angeschaut, noch die die technischen Wartlöchern es bleibt noch ein letzter Ecke von diesem Übersicht von Strukturierungsmöglichkeiten im Backend übrig und das ist genau das taktische Design von Domain Iron Design und da ist jetzt so ein hat der Eric Evans letztendlich auch verschiedene Patterns definiert, die man anwenden kann oder Building Blogs wird auch teilweise genannt.

1:09:32  
Das Ganze basiert auf der Annahme, wenn ich im Kontext habe, dann soll ein Bound it context letztendlich ja auch eine You bit critice Language haben. Das gilt ja innerhalb nur eines Bonus Kontext, da ist jedem klar was mit einem Kunden gemeint ist oder einem Auftrag oder ähnlichem und die Idee von diesem Model Design ist jetzt diese Bitcoins language die dort existiert zu formulieren in einem modernen.

1:09:59  
Und dieses Modell wird ausgedrückt, da seht ihr mit verschiedenen Dingen, mit Services, mit Entitäten.

1:10:09  
Also wirklich ein objektorientiertes Modell.

1:10:13  
Und da.

1:10:15  
Weitere Punkte, wie zum Beispiel Repositories, das ähnelt oder ist fast dasselbe wie unsere Dowos, die wir eben gesehen haben und auch Egregets und Egregets als Kapselung und und Wurzel von Entitäten. Egregats ist lange Zeit, wie ich finde, unterschätzter, unterschätztes Petter gewesen. Ist 1 eigentlich der Kern Dinge sprechen wir auch gleich noch mal im Detail darüber, das bündelt im Prinzip mehrere Entitäten.

1:10:42  
Gibt es, sind halt nur Wertobjekte, nicht veränderbar. Von Entitäten kommen wir auch noch mal zu Factories, sind Dinge, die halt wirklich dazu dienen um letztendlich Entitäten als auch hauptsächlich Aggregate wirklich zu erstellen.

1:10:59  
Das ist so im Prinzip, das sind die wesentlichen Elemente dabei. Und im Ursprungsdokument war hier auch die Layoute Architecteits, eine technische Unterstrukturierung des Backends, gedacht, was interessant ist, das seht ihr hier an dem links unteren Teil, das ist wirklich.

1:11:15  
Eine alternative Entwurf zu einer Smart UI, das ist der Punkt, den ich ganz am Anfang mal erwähnt habe. Die Fragestellung, wieviel Logik packe ich in die Oberfläche, wie ich sie ins Backend, auch da gibt es Spielarten und hier ganz klar die Herangehensweise, sozusagen das Backend kapselt alle geschäftslogik, es gibt also keine Smart UI, das Smart Uri bildet eigentlich nur das die Funktionalitäten des Backends ab.

1:11:39  
Gut BW schauen da diese Patterns jetzt im Detail aus, also fangen wir mit Entitäten an, die sind jetzt anders als Entitäten, wie wir sie eben gesehen haben in diesem Use casement basierten Ansatz.

1:11:54  
Dort ist letztendlich die Idee, dass Entitäten ein Objekt in der realen fachlichen Domäne entsprechen und auch identifizierbar sind. Ne, also das ist auch klar, man hat auch eine Identität und der wichtige oder Hauptunterschied ist, dass dort wirklich nach Objektorientierung auch die die Geschäftslogik drinsteckt, das heißt, Entitäten bieten auch Methoden an, die Dinge tun, also geschäftslogik wird dort reingetan und dort auch implementiert und das enthält dann halt Value Objects als Attribute.

1:12:26  
Value Objects wirklich die Wertobjekte kommen wir gleich noch zu, was es genau ist und kann auch andere Entitäten innerhalb eines Engels assoziieren.

1:12:34  
Und modelliert dann halt wirklich n kontinuierlichen Lebezyklus über die Realität. Ne aus dem EAK ne wo wir da mal eben schauen wie wir als Beispiel zugrunde liegen, das hätte man zum Beispiel ja so n so n Kunden, der hat ne ID ne und?

1:12:50  
Der hat dann Name, Kontakt, Phone und Address und das sind Dinge, die sich auch ändern können im Laufe der Zeit. Ne, das heißt das hat n Lebenszyklus n Kunde wird vielleicht auch kein Kunde mehr sein, dann wird er gelöscht aus dem System und der hat dann wertobjekte wieso ne Business Line? Hier hat keiner ID zum Beispiel oder N Sales Contact, die werden dann entsprechend ja auch noch mit assoziiert.

1:13:13  
Und im IHK kann zum Beispiel so ein Einkauf wochenendität sein. Der Einkauf selber, der verläuft Klärungsstatusänderungen hat ein paar Wertobjekte.

1:13:23  
Alternativ kann man aber auch sagen, pro Status macht man eine eigene Entität. Das heißt, ihr merkt allein schon so wie ich es beschreibe.

1:13:30  
Dass dieses Pattern an sich auch nicht wirklich.

1:13:35  
Eine Anleitung bietet, was gehört da jetzt an. Logik rein und was nicht. Das heißt, da ist es eine Meriade von verschiedenen kleinen Würfentscheidungen schon enthalten. Welche Entitäten habe ich, das kommt schon so ein Stück weit aus der Language natürlich raus, ne das habe ich aber da muss ich ja immer noch gucken.

1:13:50  
Wo packe ich welche Logik rein? Und die Requirement ist hier auch wirklich der Master ist Stab wie man das Design voranschreiben kann.

1:13:59  
Ich hatte ja dann bei der Erklärung der fachlichen Anforderungen mal so das Domain Storytelling erwähnt, wo man wirklich mal quasi so Diagramme zeichnet und beschreibt wie Szenario es ablaufen. Das ist auch so ein Beispiel, wo man wirklich diese Dinge dann halt Entitäten identifizieren kann, teilweise und auch methodiken und das versuchen kann auch zu überführen. Dieses Domain Story Channel in ein Modell, das ist ein Beispiel, weil das Beispiel aus dem IHK wäre zum Beispiel ein Supplier, der im Kontext gepflegt wird, der vielleicht gelöscht werden kann.

1:14:32  
Der hat also einen Lebenszyklus und eine Idee. Es ist auch eine Entität gut.

1:14:37  
Value Objects sind jetzt die dienen der fachlichen Beschreibung von so einem Objekt. Die haben keinen Lebenszyklus und keine Entität. Das bedeutet, wenn wir kein Leben Zyklus sind sie im Youtube, wir sind nicht veränderbar, wenn ich jetzt hier in dem vorherigen Beispiel den den Sales Contact irgendwie von einem Customer ändern möchte, dann muss ich ein neues Value Objekt erzeugen und den.

1:14:59  
Und setzen. Mich ersetze immer komplett das das Wertobjekt das ist da die Idee ne ist immer so ein Teil der Entität und beschreibt, dass es ein fachlicher Datentyp kann man es auch letztendlich nennen. Man kann auch teilweise eine E-Mail-Adresse, obwohl es nur ein String enthält schon als Value Object.

1:15:15  
Das macht man in der Regel auch bei Tactical.

1:15:20  
Und wichtig auch das Value Objekt kann fachliche Logik enthalten, zum Beispiel Validierung wird da ganz viel reingepackt. Ob das dann soweit gültig ist. Diese Logik kann natürlich keine Änderung vornehmen, aber das kann halt Dinge auswerten über das Value Object und preisgeben.

1:15:39  
Und ihr merkt auch oder was auch wichtig ist, was teilweise sehr verwirrend sein ist. So ein Value Objekt aus einem Boundlet context kann in einem anderen boundlet Kontext einfach ein.

1:15:53  
In der Entität sein.

1:15:55  
Das sind Dinge, die halt, weil es unterschiedliche Dinge in unterschiedlichen Kontexten sind, obwohl es das gleiche reale Objekt dahinter ist. In einem Fall ist irgendwie vielleicht eine Adresse einfach nur ein Werttyp, in einem anderen Fall kann es eine Entität sein, weil es sich vielleicht irgendwie der Straße Name ändert oder ähnliches und das dort verwaltet wird. Das sind so Dinge, die man halt da entsprechend auch wirklich berücksichtigen muss, in jedem Kontext kann es unterschiedliche Dinge wirklich bedeuten.

1:16:26  
Genau.

1:16:31  
Genau. Weiteres Beispiel ist auch so aus einem Buch Ne in der Bücherei den Ausleihe, wenn ich eine Ausleihe habe, dann habe ich in einem Kontext n Buch ein Buch in einem Katalog, das ist dann das Buch die Entität.

1:16:45  
Oder vielleicht auch nur ein Wertobjekt und dort in dem Sinne und anderenfalls wirklich eine Instanz von dem Buch. Und ich kann auch vielleicht mal gelöscht werden, wenn das Buch kaputt ist oder ähnliches. Ne, also es ist ein unterschiedliche sichten einfach und unterschiedliche Designentscheidungen, die müssen pro bauen und Kontext getroffen werden.

1:17:05  
Genau.

1:17:07  
Außerdem im RAK könnt ihr da zum Beispiel Adresse von einem Supplier kann das sein oder seine Kontonummer, ne auch ne Ratenposition mit Max. Anzahl und Preis kann das sein ne? Und da kann man auch n schönes Beispiel nehmen ne Ratenposition wo man sagt.

1:17:21  
Was ich für.

1:17:24  
Hotelzimmer mit mindestens 7 Tagen und maximal 14 Tagen.

1:17:29  
In diesem Range habe ich diesen Preis und ich kann dort eine Methode ranlangen, dass ich den Preis mir berechne. Für was weiß ich 9 Tage oder innerhalb dieses Rangers. Ich kann auch gucken, dass ich dann in der Methode drin habe, dass ich natürlich nur innerhalb von diesem Minmax den Preis Anfrage, den Gesamtpreis, das sind Dinge, die ich da dann wirklich fragen haben würde und hier seht ihr auch nochmal das Beispiel mit dem mit dem Kunden und seiner Adresse.

1:17:53  
So, jetzt kommen wir noch zu dem wichtigen Objekt, was häufig vergessen wird und das ist wirklich die Aggregats und die Aggregats sind so n Stück weit eigentlich über den den Entitäten und Value Objects zu sehen, weil sie volerisieren eigentlich nochmal.

1:18:10  
Das fachlich Komponenten bonbontext, das ist eigentlich die nächste Ebene der Strukturierung, wenn ihr so wollt. Das heißt die werden und repräsentieren häufig so eine Art Wurzel Entität und definieren auch dort eine Transaktionsklammer. Man kann dort sogar eine technische Transaktionsklammer draußen machen, wenn man möchte, aber ist erstmal eine fachliche Transaktion in dem Sinne das was ich beim agrard Anbieter Methoden die hinterlassen alle Entitäten in diesem Aggregat in einen konsistenten Zustand im Anschluss, das ist eben das heißt da werden bestimmte W regeln und Invarianten.

1:18:42  
Gekapselt und sichergestellt in diesem Regett das ist also die die Idee ist also wirklich eine Art Wächter oder Fassade zu diesem Geschlecht.

1:18:52  
Repräsentiert dann eine Kapselung von fachlichen Logiken die Zusammengehabt und genauso wie bei Use Cases ist es auch hier wichtig, dass man da die richtige Kapselung findet. Mit dem Fachbereich abgesprochen.

1:19:04  
Und das ist etwas weiß nicht, ob wir zu der Übung noch kommen, aber man kann zum Beispiel auch Design Level eventstorming machen, um solche Aggregates zu identifizieren.

1:19:15  
Genau, ist auch häufig unterschätzt. Ne wird häufig nicht so genau angewandt, man hat dann wirklich hier so Entitäten und üblicherweise hat auch ein eckige nur 1 bis 2, vielleicht 3 Entitäten ne das ist kein riesen objektgeflecht was dahinter steckt, wenn die Täter dann die Value Objects und üblicherweise die die Wurzel Entitäten, die können dann halt andere Regeln jetzt aufrufen. Also man hat dann immer einen bestimmten Zugriffspunkt, aber wenn dann weitere Entity drin ist, die wird dann auch hier nach draußen gegeben, manche sagen sogar zwischen Engelwert sollen eigentlich nur Value Objects ausgetauscht werden, kann man auch machen.

1:19:48  
Aber da ist auch das Tactical die die relativ schwammig und gibt keine großen Vorgaben.

1:19:56  
Im RK kann man zum Beispiel den User als Aggregat definieren und kann halt dort Überprüfung lassen, ob bestimmte Rechte halt passend zusammen und auch ob das Recht ist Setup in sich nimmst, stimmig ist und keinen Widerspruch hat oder ob der Einkauf kann halt zum Beispiel Simulationen anbieten für bestimmte Auslastung und dann dort entsprechend.

1:20:20  
Hat es darunter Einkaufspositionen als Souveränitäten und vielleicht bestimmte Auslastungssituation? Und dann kann man so etwas als ein Every Gate bezeichnen oder auch hier eine Purchase Order, also ein Auftrag mit einem Approved Limit und die Regel die, dass diese Purchase Order als Reggae hat sicherstellt ist, dass die Summe der einzelnen Positionen in dem Auftrag nicht größer ist als dieses Limit und da steckt dann halt wirklich die Logik drin, wie man das dann halt sicherstellt und man verteilt dann auch wirklich dieses Geschäftslogik. Zum Beispiel würde man jetzt hier nicht über

1:20:53  
Items holen. Alle paar Zonen und Price und alle Berechnungen hier oben packen. Nein, stattdessen würde man wirklich erstmal sagen zu dem Purchase Order Line Image, gib mir deinen Preis.

1:21:04  
Da wir eine Methode Gap Price oder Calculate Price, der würde dann wirklich die Quantität nehmen und den Part und dort den Preis sich holen und das Berechnen und den Preis zurückgeben. Die Purchase Order summiert dann wirklich nur noch die einzelnen Line Item Preise.

1:21:18  
Und Vergleich das gegen das Approvedlimit man kommt dann so zu einem wirklichen Verteilung der der Geschäftslogik in einem reichhaltigen Domänenmodell.

1:21:28  
Gibt noch so n paar andere Elemente. Das eine sind Repositories, ne das ist wirklich die Idee, dass man Zugriff auf Entitäten dort entsprechend ermöglicht. Ist eigentlich ein wesentlich gleiche Idee wie das Dao.

1:21:41  
Abtritt, aber vielleicht noch stärker von einer konkreten Technologie. Ein Service ist letztendlich dann noch ein weiteres Element, manchmal kann man diese Bitcoins language auch wenn sie dort ist. Der kann man nicht so die, die die Punkte finden, manche Logiken, die ganz klar einem Aggregate oder Identität zugeordnet sind, teilweise ist es eine Orchestrierung oder ein Workflow der ablaufen muss, das sind dann die Stellen wo so ein Service genutzt wird und genau das anzubieten, das heißt da kann man fachliche Aktionen auf verschiedene Aggregaten Entparken.

1:22:13  
Gibt es dann auch in verschiedenen Schichten, sowohl auf der Domäne und Application Schicht werden wir auch gleich noch mal ein anderes Beispiel sehen.

1:22:21  
Und dann haben wir die Factory und die Kapselt einfach die Erstellung von Entität oder so eine Aggregat. Das wird dann da entsprechend gemacht.

1:22:33  
Alternativ kann man auch das Bilder Pattern dann anwenden. Ne dann ja kann stellt das sicher, dass die konsistenten Regeln eingehalten sind und was man saubere und erstelltes AG Rad zurückbekommt?

1:22:44  
So, dann kommen wir zum Abschluss noch kurz zu diesem, zu dieser Idee, dass man dort wirklich auch häufig dieses taktische DB mit der hexagonalen Architektur ne verknüpft.

1:22:57  
Das heißt, hier schließt sich dann noch mal der Kreis zu dem ersten Teil. Hier seht ihr dann auch mal ne Möglichkeit, wie man es machen kann, wo ihr die verschiedenen Elemente destactical.de DS seht mit den Aggregaten hier die Entitäten haben Factories, es gibt auch noch, das ist im Prinzip Domäne Event, das hab ich jetzt hier nicht erwähnt, also fachliche Ereignisse kann man auch modellieren und das gleiche Objekt, das ist so der Domain Core, das sind die Dinge die man dort drin hat, man kann auch so Domain Services noch anbieten, das ist jetzt hier vereinfacht dargestellt.

1:23:26  
Services nach draußen die Applikationsservices das sind dann wirklich Dinge, die ja in der Adapterschicht sind, in einem Coming Teil, die irgendwie ein Import Tor darstellen und implementieren und dann auf Aggregates oder Domain Services zugreifen.

1:23:41  
Ne und die Repositories sind in Implementierungen von so einem Output Port von einem outgoing Adapter, wo man zum Beispiel die konkrete Resistenzlösung oder Datenbank anbietet und auch so n so n Messaging Adapter wenn man Daten rausschickt. Fachliche Ereignisse ist ein weiteres Beispiel dafür.

1:23:58  
Und auf die Weise kann man wirklich diesen Domain Core mit einem reichhaltigen objektorientierten Modell modellieren, wo wirklich diese Bitcoins language gemeinsam mit den Fachbereichen worfen wird auch das zum Modell gemeinsam mit dem Fachbereich entworfen wird.

1:24:13  
Die auch ein grundlegendes Verständnis dafür haben. Das heißt, auf die Weise kann man wirklich n sehr gutes Design, passgenaues Design finden, was wirklich sowohl für n Fachbereich als auch für N, für N für die IT gut verständlich ist und halt n super Basis bietet, auch Änderungen passgenau wieder anzubauen und auch zu wissen, wo man sie einbaut. Das reflektiert eigentlich so, wie der Fachbereich denkt, das ist das Idealmodell dahinter braucht natürlich auch einige Iterationen, bis man dahin kommt, das wird man nicht sofort von heute auf morgen so entsprechend hinbekommen.

1:24:47  
Auch hier werden wir uns dann in der Präsenz, um am Beispiel anschauen aus dem Hands on domain Design by example von Michael blöd, da ist es so eine Kreditapplikation für eine Bank, wo so ein Kunde sich für einen Immobilienkredit bewerben darf und da wird dann auch eine Bewertung und ein Scoring für den Kunden gemacht, ob er halt diesen Kredit, ob er dafür würdig ist oder nicht findet ihr auch Open Source? Hier werden wir uns auch nochmal gemeinsam anschauen.

1:25:14  
Gut, dann kommen wir kurz noch zu deinem zusammen. Zusammenfassung des Backend Entwurfs. Also ihr merkt noch mal was solltet ihr mitnehmen. Es gibt 2 grundlegende Entscheidungen die eigentlich treffen müsst. Wenn ihr das Backend strukturiert, ne also auch wenn ihr die Komponenten schon habt, die fachlichen wie strukturiert die Komponenten in sich, das heißt da müsst ihr über eine technische Architektur nachdenken wie ihr die technischen Aufgaben verteilt in Unterelemente sei es eine Schichtenarchitektur, sei es hexagonal, ihr müsst aber auch.

1:25:44  
Und diese querschnittleiste Aspekte entsprechend zuordnen. Zweiter Punkt ist die Struktur in der Fachlichkeit, auch hier gibt es mehrere Wahlmöglichkeiten, es gibt.

1:25:55  
Einerseits die diesen Use Case driven Ansatz sehr simpel, einfach Use Cases irgendwo definieren auf dem reinen Datenmodell arbeiten, gut ist. Das ist relativ straight for about für Entwickler leicht verständlich, da muss man auch nicht so viel Entwurf zur Arbeit reinstecken, ein bisschen sollte man reinstecken, ansonsten wird es auch da nicht wartbar, aber kann für manche Fälle einfacher sein als wirklich dieses.

1:26:16  
Objektorientierte Modell zu entwerfen, das erfordert wesentlichen Aufwand. Der ist aber natürlich wert, ist gerade in dem Fall, wo man im Core ne Core subdomäne ist, wo man sich differenzieren will, wo man schnell und gut liefern will.

1:26:30  
Und wo auch die Privatbarkeit die ganz besondere Rolle spielt. In den Fällen macht sowas natürlich wesentlich mehr Sinn und da macht dann vielleicht auch eine Kombination mit einer hexagonalen Architektur, einer starken Trennung.

1:26:42  
Der Zuständigkeitentechnik raus, rein fachlich orientierter Chor mit Villa Java im Zentrum natürlich auch Sinn.

1:26:52  
Gut, das soll es dann zu dem Kapitel Backendentwurf gewesen sein, als.

1:27:00  
Nächstes werden wir uns dann eine Ebene tiefer mit der Persistenz beschäftigen.