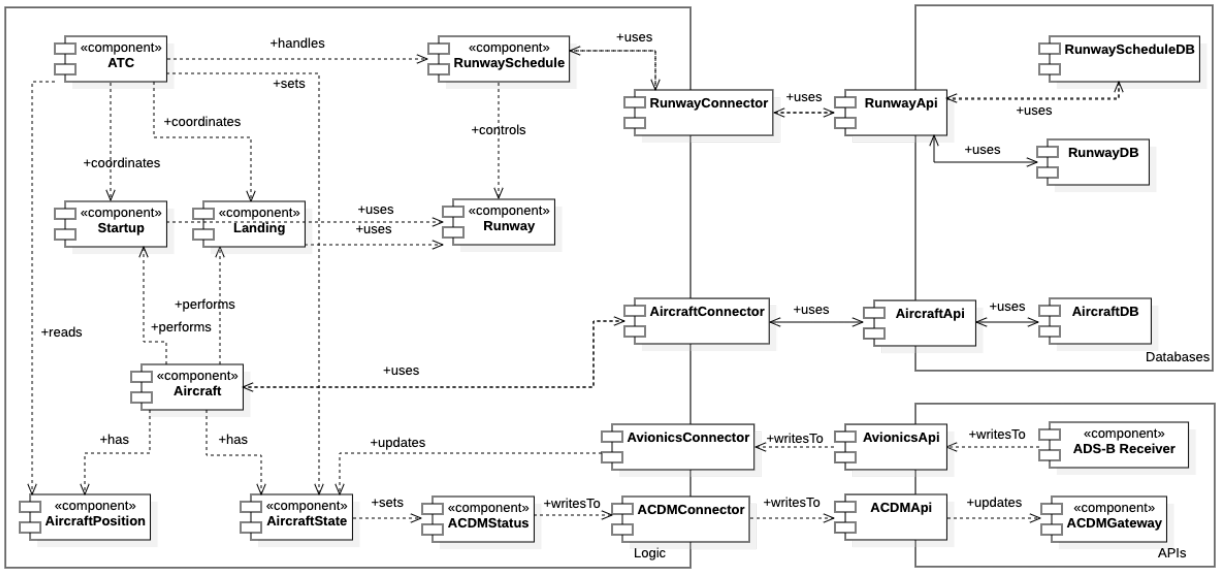
Aufgabe: Design

A) Design zu globalem Softwareprojekt

Verbessertes Komponentendiagramm zur Towersoftware:



B) Domain Driven Design

DDD Kernidee: Das Design des Systems basiert auf fachlichen zusammenhängen, die in der Anwendungsdomäne / Domänenmodell definiert sind. Bei einem Fughafen besteht das Domänenmodell beispielsweise aus: Flugzeug, Flug, Passagier, Gepäckstück, Cargostück, Bodenbewegung

Event Storming: eine Workshopmethode, bei der Domänenexperten und Entwickler gemeinsam fachliche Ereignisse im Geschäftsprozess zusammentragen und sortieren. Hieraus können dann die eigentlichen Gegschäftsprozesse, Benutzeroberflächen etc. abgeleitet werden. Dabei werden Prozesse gerne vom Ende her betrachtet und versucht herzuleiten, wie dies ermöglicht werden kann. Das daraus resultierende Domänenwissen kann für klassische User-Stories und Entwürfe verwendet werden. Für solch einen Workshop eignet sich am besten ein leerer Raum mit einer möglichst großen Wand. Geschäftsflüsse werden entlang einer Zeitlinie abgebildet, zuletzt werden kritische Bereiche (Risiken, Schlüsselfeatures, Verbesserungsmöglichkeiten) hervorgehoben.

Context Mappings

Conformist: Rolle in der Beziehung zwischen zwei bounded contexts, bei der Downstream-Kontext (Conformist) Änderungen an seinem Modell durch einen Upstream-Kontext (Supplier) übernimmt. Supplier ignorieren hierbei die Erwartungen der Conformisten. Dies kann bei sehr etablierten APIs der Fall sein.

Open-Host-Service: Basier auf dem Conformist-Ansatz. ein Upstream-Kontext implementiert ein offen dokumentiertes und versioniertes Protokoll, welches eine verallgemeinerte Nutzung des Modells ermöglicht. Dieser Service ist dann ein zentraler Zugang zu einer vielzahl spezieller Microservices innerhalb des Upstream-Kontextes, der Upstream-Kontext keine Rücksicht auf die Downstream-Kontexte nehmen muss. Downstream-Kontexte können dieses Protokoll dann konformistisch implementieren.

Published Language: Teil eines Protokolls, das ein einem Open Host Service bereitgestellt wird.

Shared Kernel: Mehrere bounded contexts teilen sich ein Modell, Sprache, Code und entwickeln sie gemeinsam weiter.

Customer Supplier: Zwei bounded contexts in Upstream/Downstream-Beziehung. Supplier liefert Modell und Sprache zugeschnitten auf Customer, behält aber die Deutungshoheit darüber.

Anti Corruption Layer: Zwei bounded contexts in Upstream/Downstream-Beziehung, bei der der Downstream-Kontext eine Schicht zwichen sich und dem Upstream-Kontext implementiert um die Objekte des Upstream-Kontextes in das eigene Modell zu übersetzen. Dies hilft die Modelle beider kontexte zu isolieren und ihre integrität zu erhalten. Beispielsweise kann damit eine neue API zu einer Legacysoftware hinzugefügt werden.

C) Clean Code Developer

Meine clean-code highlights sind, die Pfadfinderregel, dass jede Verbesserung im Kleinen beginnt, ähnlich dem *Kaizen* in der agilen Entwicklung.

Am orangen, gelben und grünen Grad gefällt mit die frühe Automatisierung von allem was automatisierbar ist (Integrations- und Unit-Tests, Continuous Integration and Delivery). Der Fokus liegt auf der Implementierung von Features und Bugfixes durch automatisierte Bereitstellung von Artefakten als Testversionen für Tester und Betas für die Stakeholder. Somit ist die Software wird sichtbar nach aussen und die Entwickler haben mehr Zeit für die wichtigen Aufgaben, wie Entscheidungen zu Code, Implementierung, Planung, Architektur.

In der Musik gab es durch die Entwicklung von elektronischen Musikinstrumenten und Sequenzern die Idee, dass von Musikern nur noch die Komponisten und Dirigenten übrig bleiben werden, da das eigentliche spielen der Musik von Automaten erledigt wird.

Entsprechend kann man beim Clean-Code-Development sagen, dass Entwickler mehr in die Rolle der Architekten und Administratoren hineinwachsen, da alle redundanten Aufgaben von Automatismen erledigt werden.