|  |
| --- |
|  |
| Software Architecture Document |
|  |
| CaféApp |

**SYSTEMINTEGRATÖR YHSIPI17/Systemarkitektur**

den 22 maj 2018

Billy Andersson, Malin Albinsson, Muhammad Assaf, Tommy Eriksson

Software Architecture Document

CaféApp

Innehåll

[2 Inledning 3](#_Toc516042927)

[2.1 Syfte 3](#_Toc516042928)

[2.2 Definitioner och förkortningar TODO 3](#_Toc516042929)

[3 Arkitekturella mål och avgränsningar 3](#_Toc516042930)

[3.1 Övergripande mål 3](#_Toc516042931)

[3.1.1 Interaktion med omvärlden 4](#_Toc516042932)

[3.1.2 Avgränsningar 4](#_Toc516042933)

[3.2 Produkter och programspråk 4](#_Toc516042934)

[3.3 Utvecklingsmiljö och utvecklingsverktyg 4](#_Toc516042935)

[4 Arkitekturell Representation 5](#_Toc516042936)

[4.1 Arkitekturella vyer 5](#_Toc516042937)

[4.2 Use case View (användarfallsvy) 5](#_Toc516042938)

[4.2.1 Översikt 5](#_Toc516042939)

[4.2.2 Aktörer 5](#_Toc516042940)

[4.2.2.1 Kund 6](#_Toc516042941)

[4.2.2.2 Personal 7](#_Toc516042942)

[4.2.2.3 Ägare 9](#_Toc516042943)

[4.2.2.4 Betalning 9](#_Toc516042944)

[4.3 Logical View (Logisk vy) TODO 9](#_Toc516042945)

[4.3.1 Funktionalitet 10](#_Toc516042946)

[4.3.2 Implementation 10](#_Toc516042947)

[4.3.3 Tillståndsdiagram 11](#_Toc516042948)

[4.3.4 Interfacedefinitioner TODO 11](#_Toc516042949)

[4.3.5 Realisering av användarfall 11](#_Toc516042950)

[4.3.5.1 Lägg produkt i varukorgen 11](#_Toc516042951)

[4.3.5.2 Gör beställning 12](#_Toc516042952)

[4.3.5.3 Skapa rapport 13](#_Toc516042953)

[4.3.5.4 Hantera produkt 13](#_Toc516042954)

[4.4 Process View 14](#_Toc516042955)

[4.4.1 Översikt 14](#_Toc516042956)

[4.4.1.1 Beställningsprocessen 14](#_Toc516042957)

[4.4.1.2 Administrativa processer 15](#_Toc516042958)

[4.5 Data View 16](#_Toc516042959)

[4.6 Deployment View (Driftsättningsvy) 18](#_Toc516042960)

[4.6.1 Lösningsöversikt 18](#_Toc516042961)

[4.6.2 Applikationsversioner TODO 18](#_Toc516042962)

[5 Icke-funktionella krav 18](#_Toc516042963)

[5.1 Svarstider och kapacitet 18](#_Toc516042964)

[5.2 Tillgänglighet 18](#_Toc516042965)

[5.3 Tillförlitlighet 19](#_Toc516042966)

[5.4 Förvaltningsbarhet 19](#_Toc516042967)

[6 Övriga Designprinciper TODO 19](#_Toc516042968)

[6.1 Felhantering 19](#_Toc516042969)

[6.1.1 Regler 19](#_Toc516042970)

[6.1.2 Dataformatering 19](#_Toc516042971)

# Inledning

## Syfte

Software Architecture Document (SAD) ger en arkitektonisk översikt över systemet CaféApp, vars syfte är att hantera såväl beställningar som bakomliggande administration av produkter och försäljningsdata i ett café. Den presenterar ett antal olika arkitektoniska synpunkter för att visa de olika aspekterna av systemet.

## Definitioner och förkortningar TODO

DFD

JPA

postgreSQL

IDE

# Arkitekturella mål och avgränsningar

## Övergripande mål

Huvudmålet med arkitekturen är att realisera beställarens krav på ett sådant sätt att systemet vid behov enkelt går att bygga ut.

Arkitekturen designas för att uppfylla målet att underlätta hanteringen av beställningar för kund och personal på ett café. Den designas också för att underlätta produkt- och orderadministrationen. Figur 1 visar på en övergripande nivå hur detta är tänkt att fungera.

Kunden skall enkelt vid besök kunna sätta sig ned, och via kundversionen av appen, endera i plattan som finns vid bordet, eller nedladdad till kundens privata mobila enhet, knappa in den produkt denne vill beställa och göra en betalning, allt i applikationen, för att sedan få sin produkt.   
Personalens arbete underlättas då dessa slipper sköta betalningen. Personalen får via en personalversion av appen en notifikation om beställd produkt och till vilket bord som produkten skall levereras.   
Ägaren kan via ägarversionen av appen administrera produktutbudet och ta ut rapporter om försäljningen.



Figur DFD Level 0 – Systemet och dess context

### Interaktion med omvärlden

CaféApp ska vara en applikation som ska finnas för iOS och Android. Vid varje bord finns en platta med applikationen, men den ska också gå att ladda ner till privata mobila enheter.

När kunden lagt sin beställning skickas en notifikation till personalen som via sin personalversion av applikationen kan se vad som beställts och till vilket bord.

### Avgränsningar

De huvudsakliga avgränsningarna med avseende på design och implementation är flexibilitet och enkelhet.

## Produkter och programspråk

Tabell 1 visar produkter som ingår i systemet.

Tabell Produkter och programspråk

|  |  |  |  |
| --- | --- | --- | --- |
| Produkttyp | Produkt | Version | Beskrivning |
| Driftskomponent | TeamCity | 2017.2 | Bygg ledning & Kontinuerligt integrationsverktyg |
| Databashanterare | pgAdmin | 3 | administration och utveckling plattform |
| Programspråk | Java | 8 | Programmeringsspråk för iOS-versionen |
| Programspråk | Swift | 4 | Programmeringsspråk för android-versionen |
| Programspråk | Java EE | 8 | Programmeringsspråk API:et |

## Utvecklingsmiljö och utvecklingsverktyg

Tabell 2 visar de miljöer och verktyg som kommer att användas för realisering, byggande och tester.

Tabell Miljöer och verktyg

|  |  |
| --- | --- |
| Produkt | Ansvar |
| Eclipse | Programmeringsverktyg (IDE) |
| Android studio | Programmeringsverktyg för android-versionen |
| Xcode | Programmeringsverktyg för iOS-versionen |
| JUnit | Enhetstestning av kod |
| Google analytics | Byggstöd och rapporteringsverktyg |
| JPA | Anslutning och hantering av databasen |
| PostgreSQL | Relationsdatabas |

# Arkitekturell Representation

## Arkitekturella vyer

Vid modellering, dokumentation och implementering av ett system är det viktigt att systemet kan betraktas ur olika perspektiv. Vi har därför valt att presentera arkitekturen i fem vyer: Användarfallsvy, Logisk vy, Processvy, Datavy och Driftsättningsvy.

* Use case view har som primärt syfte att gestalta grundläggande användningsfall och därmed identifiera de huvudsakliga systemkraven, samt illustrera och validera den valda designen. Denna vy riktar sig till alla, inklusive applikationens slutanvändare.
* Logical view skall definiera systemets komponenter och interfaces för interaktion och kommunikation, således den funktionalitet som systemet ger slutanvändaren. Detta görs genom klassdiagram och tillståndsdiagram. Primär målgrupp: designers/utvecklare.
* Process view ska förklara systemprocesserna och hur de kommunicerar och interagerar, fokuserar på runtime, vi använde aktivitetsdiagram. Målgrupp: Integratörer och utvecklare.
* Data view visar dataflöden och datastruktur genom dataflödesdiagram och databasmodeller och är främst avsedd för dataspecialister och databasadministratörer.
* Deployment view definierar den fysiska miljön som systemet ska driftsättas i. Deploymentdiagram används för detta. Denna vy riktar sig till de som ansvarar för driften.

## Use case View (användarfallsvy)

### Översikt

Syftet med användarvyn är att ge ytterligare sammanhang kring användningen av systemet och interaktionerna mellan dess komponenter. I detta dokument betraktas varje komponent som en användargruppsaktör.

En övergripande beskrivning av systemet ur ett funktionellt perspektiv är att en kund kommer in på caféet, använder kundversionen av appen för att titta på menyn, eventuellt granska innehållsförteckningen för någon produkt närmare, väljer produkter att lägga i varukorgen, skickar och betalar beställningen. En notifikation går då via personalversionen av appen till personalen som leverar beställningen till rätt bord. Samtidigt uppdateras lagerstatus för produkterna och ordern registreras i en försäljningsdatabas.   
Ägaren har i sin version av appen möjlighet att skapa rapporter så som försäljningsdata och lagerstatus. Vid behov kan ägaren även uppdatera eller lägga till ny produktdata.

### Aktörer

Systemet har fyra olika externa aktörer. Kund, Personal, Ägare och Betalning (exempelvis en bank).

#### Kund

Kunden kan titta på menyn, granska innehållsförteckningen, välja produkter att lägga i varukorgen, titta och ändra i varukorgen och slutligen skicka beställningen, betala och få kvitto.



Figur Use cases Kund



Figur Några use cases för Kund i tabellform

#### Personal

Personalen hanterar beställningen och ser till att kunden får sin mat och dryck. Eftersom kunderna också kan ställa frågor om produkterna direkt till personalen istället för att söka information i systemet ligger *Få fråga om produkt* utanför själva systemet, och då kan de vid behov ta fram information om produkten från systemet.



Figur 4 Use Cases Personal

#### Ägare

Ägaren står för det administrativa, lägger till, tar bort och ändrar produkter på menyn och deras pris. Denne kan också ta ut rapporter ur systemet.



Figur Use Cases Ägare

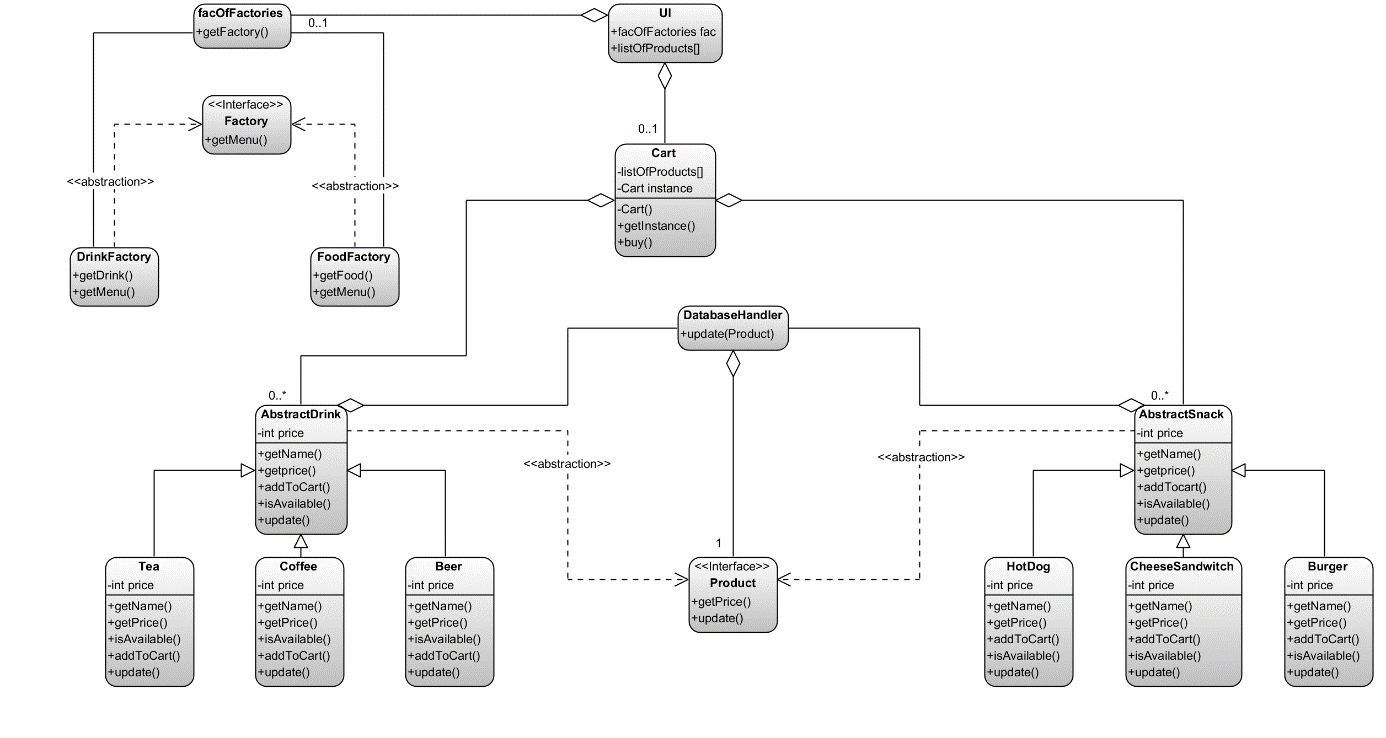
#### Betalning

Betalningen är en aktör utanför applikationen.

När en betalning sker skickas den till aktören som ger en konfirmation om betalningen gick igenom eller ej.

## Logical View (Logisk vy) TODO

Klassdiagrammet i Figur 6 visar komponenterna i systemet och hur de är utformade för att fungera med varandra. Det ger idén om hur systemet är utformat och riktlinjerna för hur det kan utvecklas i framtiden.



Figur Klassdiagram

### Funktionalitet

Arkitekturen som används har tre olika designmönster som ger dess funktionalitet.

**Abstract factory-mönster:** är ett designmönster som används för att underlätta skapandet av objekt utan att visa klienten någon detaljerad information. Vi använder det här mönstret för att skapa produkterna. Detta mönster är också lätt att förlänga i framtiden när man vill lägga till nya kategorier eller nya produkter i befintliga kategorier.

**Observer-mönster:** Med hjälp av detta mönster kan vi uppdatera informationen relaterad till produkterna när det gäller deras existens eller information relaterad till dem som beskrivningen eller priset.

**Singelton-mönster**: vi använde detta designmönster i objekten som vi behöver ha en instans av på varje instans av applikationen som till exempel Cart-klassen (Varukorgen).

### Implementation

Varje kategori implementeras som en abstrakt klass som implementerar Product-interfacet och objekten i kategorin ärver den klassen. Det gör det lättare att ange varje kategoris specifikationer och att nya kategorier kan läggas till.

Varje kategori har sin egen Factory-klass som implementerar factory-interfacet och är ansvarig för att skapa objekten. Klassen FacOfFactories används för att välja vilket Factory som ska anropas.

Databasehandler-klassen är den klass som ansvarar för att uppdatera produkterna i databaserna.

### Tillståndsdiagram

Diagrammet i Figur 7 beskriver de olika tillstånd som en beställning går igenom.



Figur Tillståndsdiagram Gör beställning

### Interfacedefinitioner TODO

### Realisering av användarfall

Här visas användarfallsrealiseringar för några av de viktigaste användningsfallen.

#### Lägg produkt i varukorgen

Kunden lägger en produkt i varukorgen.



Figur Sekvensdiagram addProductToShoppingBag

#### Gör beställning

Kunden gör en beställning.



Figur Sekvensdiagram makeOrder

#### Skapa rapport

Ägaren skapar en rapport.



Figur Sekvensdiagram createReport

#### Hantera produkt

Ägaren uppdaterar eller lägger till en produkt.



Figur Sekvensdiagram handleProduct

## Process View

### Översikt

I processvyn visar vi hur huvudprocesserna fungerar och hur de interagerar med varandra. Vi använde aktivitetsdiagram för att visa vad som händer när huvudprocesserna anropas.

#### Beställningsprocessen

Figur 12 visar hur beställningen fungerar när användaren (kunden) gör en beställning via appen, och Figur 13 visar hur personalen hanterar beställningen som kunden gjort.



Figur Aktivitetsdiagram Gör beställning



Figur Aktivitetsdiagram Hantera beställningen

#### Administrativa processer

Figur 14 visar de administrativa processer som ägaren sköter och hur de fungerar.



Figur Aktivitetsdiagram administrativa processer (hantera produktdata, ta ut rapporter)

## Data View

Dataflödesdiagrammet level 1 i Figur 15 illustrerar dataflödet mellan de externa enheterna (Kund, Personal, Ägare och Betalning) och Caféapp-systemet. Systemet inkluderar två databaser, en över de produkter caféet hanterar, och en över försäljningsdata. På den här nivån ser vi fyra processer, en för orderhantering, en för att hämta produktinformation, en för att administrera produkterna och en för att skapa rapporter.



Figur DFD Level 1

Figur 16 och Figur 17 visar databasmodellerna för databasen Produkter (lagrar produktdata) respektive databasen Försäljning (lagrar orderdata). Relationerna mellan tabellerna visas med muliplicitetsförhållandena angivna. Primärnyckeln är understruken.



Figur Databasmodell för databasen Produkter



Figur Databasmodell för databasen Försäljning

## Deployment View (Driftsättningsvy)

### Lösningsöversikt

Systemet består av tre fysiska noder, själva applikationen, applikationsservern och databasservern, vilket visas i Figur 18. Som applikationsserver används Linux Ubuntu och för databaserna postgreSQL. Kommunikationen dem emellan sker via jpa.



Figur Deployment view diagram

### Applikationsversioner TODO

CaféApp finns i tre olika driftsversioner, vilka har olika funktioner upplåsta i användarinterfacet. Endast versionen för Kund är allmänt tillgänglig. I kundversionen är de funktioner som beskrivs i Sektion 4.2.2.1 åtkomliga, medan personalversionen omfattar funktionaliteten i Sektion 4.2.2.2 och ägarversionen endast har åtkomst till de administrativa funktionerna, se Sektion 4.2.2.3.

# Icke-funktionella krav

## Svarstider och kapacitet

Från kundens perspektiv ska svarstiden normalt understiga 2 sekunder vid normalbelastning (20 samtidiga kunder). I övriga fall ska svarstiden vara mindre än 8 sekunder.

Systemet ska klara att hantera minst 50 samtidiga användare.

## Tillgänglighet

För kunden ska systemet vara tillgängligt minst under caféets öppettider. För personal och ägare ytterligare minst från två timmar innan caféet öppnar, samt fyra timmar efter stängning för att möjliggöra administrativa uppgifter.

Användargränssnittet ska vara intuitivt.

## Tillförlitlighet

Systemet bör gå att drifta med redundans.

## Förvaltningsbarhet

Det ska finnas separata miljöer för utveckling, test och produktion.

# Övriga Designprinciper

## Felhantering

Javas Exceptions används för felhantering i android-varianten av applikationen. Swifts Error handling används för felhantering i iOS-varianten.

### Regler

När ett fel inträffar ska problemet i första hand åtgärdas. Om felhanteringsrutinen inte kan åtgärda problemet ska det kastas vidare.

De fel som kastas ska i första hand vara egendefinierade, för att dölja implentationsdetaljer som inte bör framgå i det publika användargränssnittet. På så vis förbättras också systemets underhållsbarhet.

### Dataformatering

Datum skall formateras till ÅÅÅÅ-MM-DD format, t ex 2018-07-12.

Data som saknar värde, exempelvis en tom sträng, ska returneras som *null*.