Vad är en bra projektmetod för små IT-projekt?

Ett försök att besvara frågan görs i kursen II1302 ”Projekt och projektmetoder” vid KTH EECS Kista våren 2020

Hugo Lindell[#1], Sadeq Gholami[*2], Abdirahman Abdullahi[#3], Mohamed Mahdi[#4]

#Section for ICT, Royal Institute of Science and technology

Electrum 229, 164 40 KISTA Stockholm Sweden

[1]hulind@kth.se

[2]sadeqg@kth.se

[3]aabdul@kth.se

[4]mmahdi@kth.se

***Abstract*— Kurs vid KTH ICT.**

Texten är en rapport om projekt och projektmetodik kursen som undervisas på KTH. Syftet med kursen projekt och projektmetodik är att komma fram till ett svar till frågan ”Vad är en bra projektmetod för små IT-projekt?” Frågan ska besvaras genom en undersökning och praktiska utförandet av ett litet IT-projekt. Då är målet med kursen att utveckla en IT-produkt i en liten projektgrupp och pröva de olika projektmetoder som lärs ut under kursens gång. Den mest centrala metoden som används i kursen är scrum inspirerade projektmetod och agila arbetssätt med en arbetstavla som verktyg.

***Keywords* ‗ Projektmetodik, Scrum, agil, iterativ projektmetod, arbetstavla.**

I. OM DETTA DOKUMENT OCH UNDERSÖKNING

(Sadeq Gholami)

Detta dokument innehåller rapport om en undersökning som pågått under kursen projekt och projektmetodik. I detta dokument framgår Litteraturstudier kring arbetsmetoder för små IT-projekt samt resultat av en undersökning om att ”komma fram till “en bra projektmetod för små IT-projekt”. Dokumentet innefattar dessutom undersökningsmetoder som används för att besvara frågan, samt praktiska genomförandet av arbetet utifrån olika roller i projektgruppen. Så detta dokument riktar sig mot IT-branschen och andra branscher där man jobbar i små projekt och utvecklar en produkt.

Resultatet av rapporten är av en hög validitet när det gäller små IT-projekt (vilket är syftet med kursen) eftersom den har uppnåtts efter ordentligt undersökning, inläsning av olika teorier och praktiska genomförandet av de teorierna samt diskussioner. Däremot har resultatet testats bara för små IT-projekt så det kan vara av låg validitet för stora IT-projekt och projekt i samtliga branscher.

Bilagor:

1. Projektdefinition
2. Teknisk dokument om webbapp
3. Vision
4. IOT Client Component Description
5. Use-Case Model Survey
6. Test Specification
7. Device Screen Component Description
8. Hugo Tekniskdokumentation
9. Door Display API Component Description
10. Architecture Description
11. API description
12. API component diagram
13. Door display Use Case
14. Systemarkitektur

II. INTRODUKTION

A. *Bakgrund*

Projektet är huvuddelen av kursen II1302 Projekt och Projektmetoder på KTH. Kursens projekt är ett rollspel där elever ska inta olika roller inom ett IT-projekt där läraren är en spelledare. Projektet ska utföras enligt en projektmetod och som avslutas med en gemensam Projektrapport och projekt demo där gruppen ska dokumentera och presentera sina erfarenheter att arbeta med projektmetoden. Projektgruppen ska utifrån given kurslitteratur och erfarenheter i projektarbetet lära sig om praktisk arbete i ett projekt. Kursen hade sitt slut den 1 Juni 2020.

B. *Problemformulering*

Projekt för att skapa en mjukvarulösning som är enkel att underhålla, stabil och säker, effektiv och standardiserad.

Product characteristics	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilization, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable, and compatible with other systems that they use.

Essential attributes of good software [1]

A. *Undersökningsstrategi/lösningsstrategi*

Fallstudie genom att i grupp försöka genomföra ett mindre IT-projekt med utvalda projektmetoder och där varje

gruppmedlem värderar sitt ansvarsområde i projektet för att ur detta försöka besvara den övergripande frågeställningen.

B. Relaterade arbeten

(Abdirahman Abdullahi)
Inga relaterade arbeten har hittats.

C. Avgränsningar

(Mohamed Mahdi & Hugo Lindell)
Avgränsningen som gjordes i denna rapport var att hålla sig till Scrum metoden och Git som DevOps platform.

III. TEORI OCH INGENJÖRSPRAXIS

Detta kapitel listar och i viss mån beskriver teorier och ingenjörspraxis som använts i undersökningen. Det finns två underkapitel, Litteraturstudie och Förstudie.

D. Litteraturstudie

I detta kapitel anges litteratur och andra källor som har använts för att hitta ingångar och möjligheter i undersökningen? Förutom de källor som anges finns förmodligen andra, och kanske bättre, källor som denna studie inte använts sig av.
Undersökningens görs utifrån övergripande projektmetoder men också utifrån specifika metoder och arbetssätt som används av olika kompetenser [ref Essence] i projektets team. Vilka dessa kompetenser är framgår av texten nedan.

Övergripande källor för hela projektet

- Projektets hälsa och status, (OMG, 2013)
- Scrum [ref Kniberg]
- Fasindelning
- Projektgrunder (Eklund, 2010)
- Person- och gruppdynamik (Eklund, 2010)
- Software Engineering av Ian Sommerville [4, kap. 1,2, 3]

Kundrepresentant (Abdirahman Abdullahi)

- Use Case Modeling av Kurt Bittner och Ian Spence [4, kap. 2–4]. Dessa kapitel behandlar vision, problembeskrivning, UseCase modellering och översiktlig beskrivning av kund-representant rollen.
- Software Engineering av Ian Sommerville [4, kap.4]. Behandlar kravspecifikation och hantering.
- USE-CASE 2.0 The Guide to Suceeding with Use Cases av Ivar Jakobson, Kurt Bittner och Ian Spence [5]. En guide som behandlar hur man definierar och hanterar krav.

Analytiker - kompetens enligt []

- Arkitektur (Kruchten, 1995)

Utvecklare
Denna roll antogs av hela projektgruppen och utelämnas därmed från projektet och undersökningen.

Testare (Mohamed Mahdi)

- Software Engineering version 9 av Ian Sommerville [1, kap 2,3,4,8]. De tar upp konceptet agile och teststrategier.

Ledning och styrning - kompetens enligt []

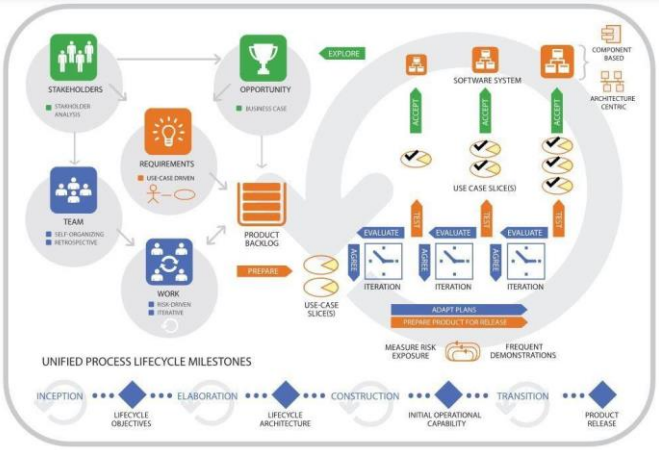
- Projektplanering
  - Projektdefinition

Miljö, hållbar utveckling, etik och jämställdhet

Arbetsmiljö

E. Förstudie

Vilka möjligheter till ansats har beaktats och prövats? Brett perspektiv.
Enligt undersökningsstrategin så skall någon projektmetod prövas i ett praktiskt projekt och utifrån de erfarenheter som fås görs en värdering av använda metoder. Frågan är då vilken ansats av projektmetod som skall användas. Eftersom erfarenheten av projektarbete hos studenterna i denna kurs är liten så fanns det ett färdigt förslag till ansats av projektmetod. Detta projektmetodförslag kan senare modifieras av projektgruppen.
Tidigare kursomgångar och lärarens förslag har mynnat ut i följande ansats. Projektmetoden framgår med god tydlighet av de arbetstavlor som definierats i ansatsen, se figurer och bilder.



Vision, projektdefinition, iterativt, Scrum ….

Resultatet av förstudien är att metoderna, som anges i följande kapitel, har valts för undersökningens genomförande.

1) Arbetstavla (vald ansats)

(Sadeq Gholami)
Arbetstavlan består av två sida, den publika sidan och arbetssidan. Publika sidan består av de delar vilka berör intressenter som kunder, entreprenörer och även teamet. I publika ingår product backlog som innehåller målet med projektet, usecases/ usecase slilces samt modellen till alla usecases. Där ingår även projektplanering som innehåller tidsplaneringen, tidsrapportetingar, projektets status/ hälsa och eventuella kostnader och budget. (Se figur 2)
Arbetstavlans arbetssida som berör främst teamet består av tasks som tillhör varje usecase. När en person i teamet börjar med en task ska hen checka ut tasken sedan när hen har testat klart tasken skall den flyttas till done delen av tavlan. Arbetssida består dessutom av systemarkitekturen, test-planen, eventuella risker, burn down chart och även oplanerade tasks och saker man tänker göra på nästa iteration. (Se figur 3-6)

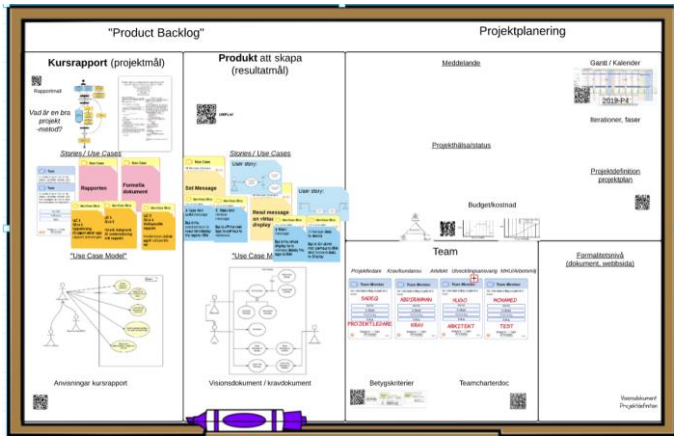


Figure 1 Arbetstavla "Whiteboard" "Publik sida"

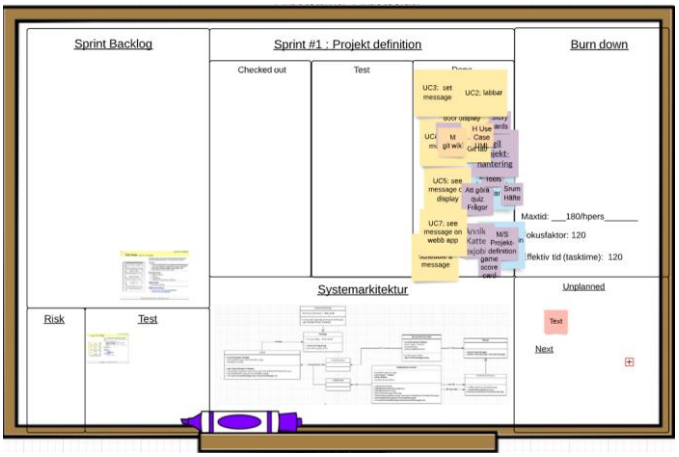


Figure 2 Arbetstavla "Whiteboard" "Arbetssida iteration 1"

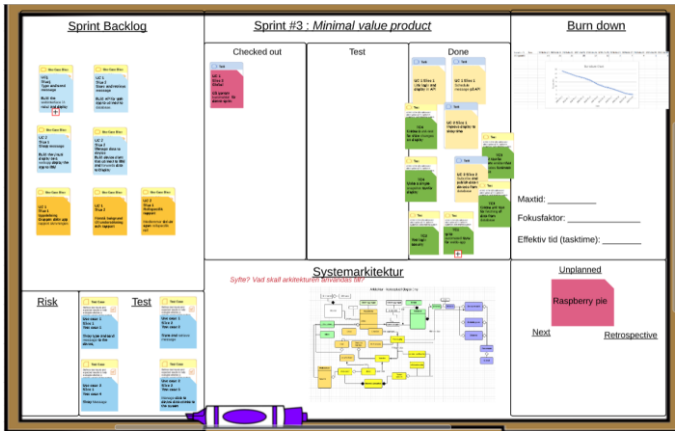


Figure 3 Arbetstavla "Whiteboard" "Arbetssida iteration 2"

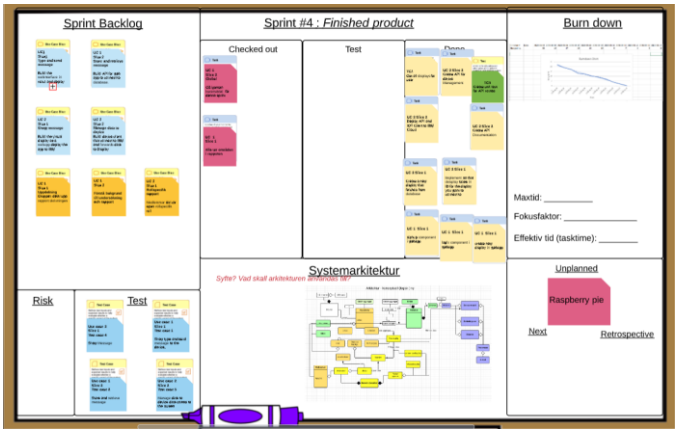


Figure 4 Arbetstavla "Whiteboard" "Arbetssida iteration 3"

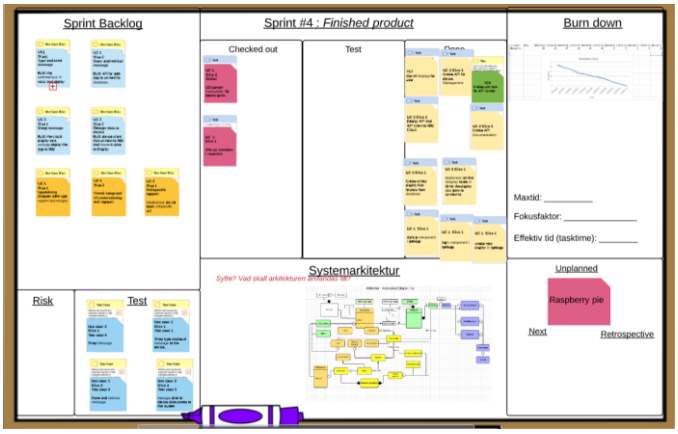
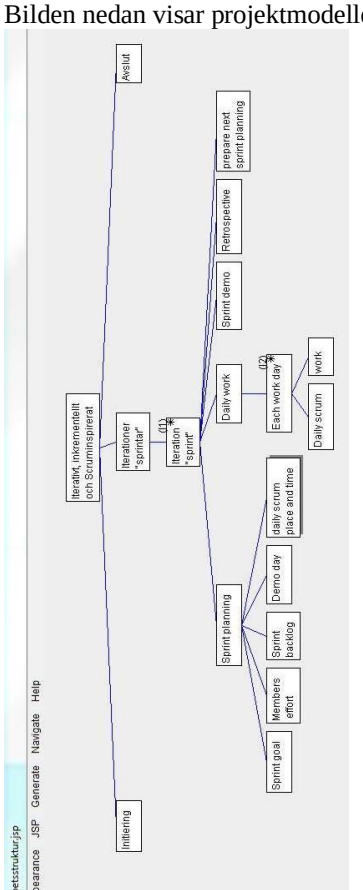


Figure 5 Arbetstavla "Whiteboard" "Arbetssida iteration 4"

2) Scruminspirerade projektaktiviteter (vald ansats)

Scrum sprinter är två veckor långa och har planerade mål (tasks) som ska genomföras vid varje sprint. Sprinten avslutas med ett demo. Projektet genomför Scrum demon inför en annan kursgrupp eller lärare eftersom projektet inte har tydliga stakeholders ger det viktig feedback vi inte fått annars.
Bilden nedan visar projektmodellens aktiviteter och ordning



Kruchten[11] beskriver 4+1 arkitektur view modellen där mjukvara delas upp i 5 vyer där varje vy är menad för olika målgrupper.

Lågisk vy: funktionaliteten som menas åt slut användaren, kan beskrivas med ett klass diagram.

Process vy: System funktioner som kommunikation och parallellisering, kan beskrivas med ett sekvensdiagram.

Utvecklar vyn: Implementationen för utvecklare, kan beskrivas med ett komponentdiagram.

Fysisk vy: Projektets fysiska implementation.

Scenarier: Modell av interaktionerna mellan de andra vyerna, kan beskrivas med en Use-case modell.

IV. UNDERSÖKNINGSMETODER

(Sadeq Gholami)
De flesta moderna projektmetoder bygger på att man jobbar inkrementell och iterativ. Det vill säga man delar hela produkten till små färdiga produkter som i början möter ett minimikrav. Efter varje iterationer skall produkten testas och levereras till kunden för eventuella förbättringar. Detta kapitel beskriver då metoder som användes för att genomföra undersökningen. Metoderna är valda och specificerade så att de skall kunna ge svar på ett antal följdfrågor som identifierats i denna undersökning. Först anges frågorna och sedan följer metodbeskrivning.

F. Frågor att besvara i undersökningen

Frågorna kategoriseras i följande kategorier …(eventuellt)

1. Hur skall man bedöma/redovisa om en delprojektmetod eller praktik är bra?
2. Hur kan man kategorisera, välja, och namnge projektmetoder (projektpraktiker) och (verklighetsbeskrivning) så att diskussionen om dito blir begreppsmässigt konsisten för ingenjörer inom IT-området (s.k. ontologi?).
3. Vilka ansvarsroller skall användas som ansats i projektet?
4. Vad består ett projekt av och vilka metoder/praxis skall användas, undersökas och bedömmas? Vilken ansats skall göras?

G. Metodbeskrivning (undersökningsmetod)

(Sadeq Gholami)
Den centrala metoden i undersökningen är att avgöra om en projektmetodik är effektiv eller inte. nedan följer två metoder som kan användas för att besvara de följdfrågorna.
Metod 1: Undersökningsmetod enligt artikeln ”Vetenskaplighet – Utvärdering av tre implementeringsprojekt inom IT Bygg och Fastighet” av Niklas Anderson och Anders Ekholm (2002) se figur 6. enligt denna metod så går det inte att beskriva vad vetenskap är. För att förstå vetenskapen skall man istället fokusera på vad är det som utmärker ett vetenskapligt arbete. Ett generellt sätt för vetenskapligt arbete är att följa ett vetenskapligt arbetssätt inom teknologisk forskning som kallas för ingenjörsmässighet. Så för att arbeta vetenskapligt enligt modellen ska man följa några steg vilka framgår av de gula fälten som är aktiviteter som kopplar till själva undersökningen.

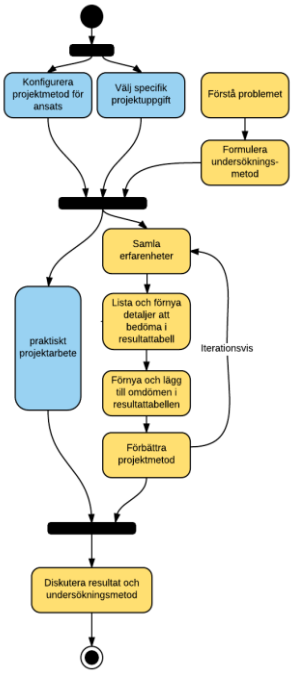


Figure 6 Undersökningsmetod för ”Vad är bra projektmetod för små IT-projekt”.

Metod 2: Begrepp Begrepp som används följer om möjligt OMGs standard *Essence - Kernel and Language for Software Engineering Methods Version 1.0* [12] .
Följande bilder listar illustrativt centrala begrepp. I denna artikel kommer de engelska begreppen att fritt översättas till svenska då risken för missförstånd anses liten.

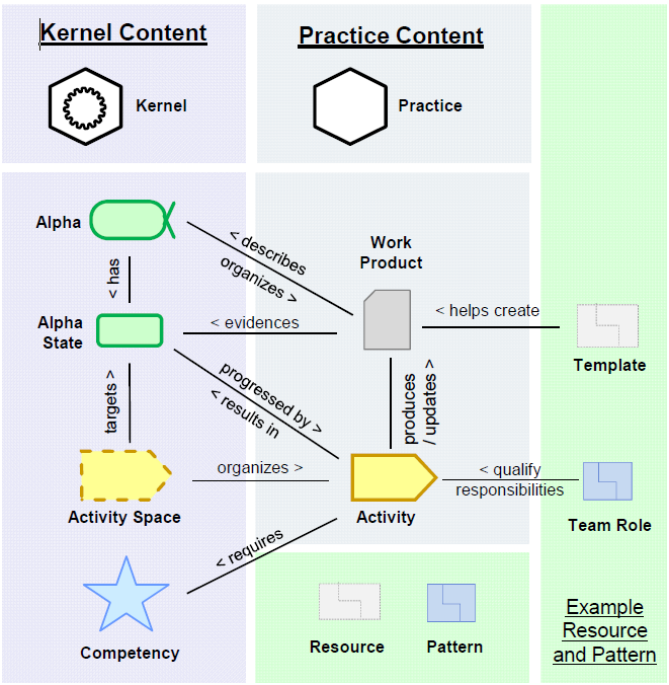


Figure 7 Begrepp (Elvesæter, Benguria, & Ilieva, 2013)

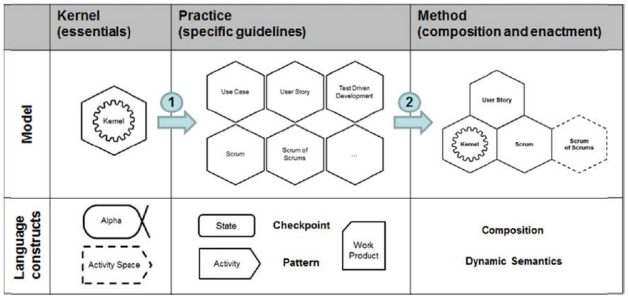


Figure 8 Practice och Methods (Elvesæter, Striewe, McNeile, & Berre, 2012)

(Hugo Lindell)

För att bedöma vilken projektmetod som passar bäst för ett projekt måste man först etablera kriterier som ger ett framgångsrikt projektarbete. Effektivitet, att ingen i projektet går utan arbetsuppgift under för lång tid. Samarbete, hur bra anpassad är metoden för samarbete mellan flera utvecklare och främjar bäst kommunikation. Integrering, hur enkelt kan arbeta från flera utvecklare integreras tillsamman.

I de flesta större företag har utvecklingsprojekt är inte bara grupperna större men de är oftast omgiven supportpersonal som designers, kund-specialister, företagsledare, dedikerade testare och ekonomer som alla tillsammans arbetar med kärngruppen att ta fram en produkt. Projektets projektgrupp är relativt liten på bara 4 personer därför är det mycket viktigt att ansvarsroller tilldelas effektivt så att ingen person blir över- eller under belastad genom någon av projektets delar. Vattenfallsmetoden följer ett strikt linjärt utvecklingsmönster, där projektet passerar olika steg på sin väg till målet.

1. Kravspecifikation
2. Design
3. Implementering
4. Integration
5. Verifiering
6. Installation
7. Underhåll

[3]

I en vattenfallsmetod har oftast varje person ett eget syfte och varje roll är isolerad, detta innebär varje person kan bli högt-specialiserad på ett område, men samtidigt behövs det då fler personer per projekt och risken föra att vissa personer har stora variationer i arbetsbelastning under olika delar av projektet är hög. Varje steg i projektcykeln som projektet passerar har olika ansvariga personer.

Alternativet är att arbeta agilt med DevOps där utveckling sker cykliskt och varje person har ansvar för utveckling, testning och distribuering. Det agila arbetssättet är också väldigt bra för sammanställningen av arbete som utgjorts av olika utvecklare där kodbasen sammanställs och testas tillsammans efter varje cykel (sprint) och då kan eventuella problem med samarbetet upptäckas tidigt och fixas direkt istället för att dessa problem uppstår i slutet av projektarbetet som det kan i vattenfallsmetoden. Den cykliska naturen av agilt med DevOps tillsammans med det bredare individuella ansvaret gör att det agila arbetssättet passar bättre för den typen av projekt och produkt som har få utvecklare.

V. GENOMFÖRANDE

I följande kapitel redovisas viktiga beslut, förändringar och anpassningar som gjorts i projektmetod, projektpraktiker, värderingar, beslut mm som gjorts under studiens genomförande.

H. Projektledning (Sadeq Gholami)

Denna roll i en projektgrupp innefattar den organisatoriska och administrativa förmåga vilket innebär att en projektledare ska se till att projektarbetet ska göras så effektivt som möjligt. Detta nås genom att ledaren har förmågan att på ett effektivt sätt planera och samordna arbetet som utförs av teamet. Dessutom bör en ledare ha förmågan att inspirera, motivera och leda teamet så att gruppen når målet på ett effektivt sätt. Ledaren bör också ha förmågan att lösa olika konflikter som uppstår under arbetets gång så att de inte påverkar gruppens arbete att uppfylla målet på ett effektivt sätt. Det är projektledaren som ska se till att alla medlemmar ska känna sig trygga och inkluderade.

När det gäller mjukvaruutveckling så ska projektledaren se till att produkten är färdig enligt den planerade tiden, att kostnaden inte ska överstiga budgeten, att produkten uppfyller kundens kravspecifikation och att medlemmar i gruppen mår bra och jobbar effektivt. [1]

När det gäller genomförandet av projektarbetet så har projektledaren sett till att alla i gruppen känna sig trygga och inkluderade i viktiga beslutsfattanden. Projekt metoden som användes för genomförandet av arbetet var som tidigare nämnts Scrum inspirerade projektmetod och agila arbetssätt. Men ledaren var öppet till olika idéer och förslag från gruppmedlemmar vad gäller arbetsmetoden. Detta för att

arbetet inte ska vara beroende på en enda arbetsmetod och att gruppen ska testa de olika metoder och komma fram till det som fungerar bäst för dem. Ett exempel kan vara att man fick använda sig av vattenfallsmodellen för att göra sin egna enskilda task. De viktigaste ändringar har gjorts efter diskussioner och omröstningar så att majoriteten skulle bestämma ändringar i projektmetoden. För att alla ska känna sig trygga med den metod som användes. Däremot så stora ändringar inte har gjorts från den centrala arbetsmetoden som var scrum och agila arbetssätt.
När det gäller projekt praktiker så har ledaren sett till produkten skall uppnå minimikravet i första hand och i mån av tid (budget) skall den uppfylla övriga krav. arbetsuppgifter var fördelade på ett rättvist och demokratiskt sätt i början av varje sprint. Det vill säga arbetsuppgifter till varje "use-case" var delade till olika tasks, de tankarna var fördelade främst volontärt.
Att anta denna roll har varit väldigt inspirerande och utvecklande för projektledaren samtidigt som den hade sina svårigheter i vissa fall. Den har hjälpt projektledaren att bli ansvarsfull, och att planera och använda sin tid på ett effektivt sätt. Projektledaren har bland annat lärt sig att leda och inspirera medlemmar och att tydliggöra målet för dem.

I. Kundrepresentant (Abdirahman Abdullahi)

I praktiken arbetar en kundrepresentant med att identifiera produktens funktioner utifrån intressenters (projektgruppen spelar rollen som intressenter och utvecklare) krav, prioritera dessa för utveckling och se till att produktbackloggen möter och speglar intressenters krav [1].
I iteration 1 av projektet var kundrepresentanten ansvarig för att definiera och precisera problemet och se till att projektet övergick från intressenters krav och behov (problem domänen) till en specifikation på en produkt som uppfyller dessa krav (lösningsdomänen).

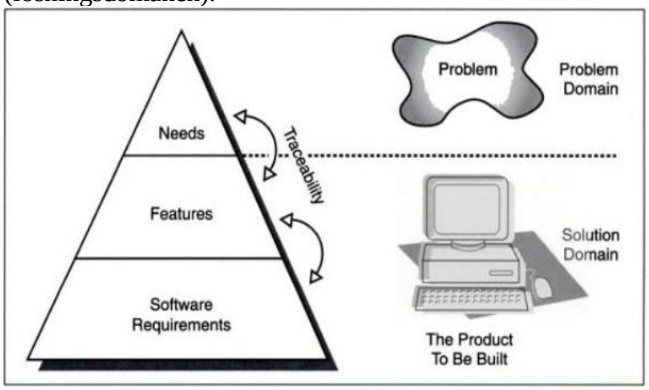


Figure 9 The requirements pyramid (Bittner & Spence, 2002)

Att tydligt definiera och precisera problemet minskar risken att projektet fokuserar på icke kritiska och icke nödvändiga delar av problem domänen [4]. Detta förenklar även övergången från intressenters krav till specificerade produktfunktioner.
I detta projekt definierades problemet genom att formulera en problembeskrivning. Problembeskrivningen redogjorde fyra aspekter av problemet:

1. Beskrivning av ideala scenariot för intressenter. Det vill sägas deras önskat tillstånd och slutgiltiga mål som de anser vara viktiga och relevanta för problemet.
2. Beskrivning av intressenters nuvarande verklighet. Det vill sägas de faktorer som hindrar intressenters från att uppnå deras ideala scenario 3
3. Beskrivning av konsekvenser som uppstår om inte en förbättring från intressenters nuvarande verklighet sker.
4. Beskrivning på en föreslagen lösning till problemet.

Intressenters krav tillhör en av två huvudkategorier; funktionella och icke-funktionella krav. Funktionella krav som uppfyller intressenternas grundläggande krav omvandlades till produktfunktioner. Enligt Bittner och Spence är produktfunktioner en fundamental faktor för att definiera produkten och hantera dess omfång [4]. Produkt-funktionerna prioriterades enligt MoSCoW modellen (Must, Should, Could och Won't).

Need	Priority	Features	Planned Release
Secure and private access for users to their door displays.	SHOULD	Secure Login	Sprint 3-4
Set the information to be displayed on door display in real-time.	MUST	Set Message	Sprint 2-3
See the information currently displayed on door display	MUST	Display on device, Display on web application	Sprint 3
Convey the information audibly for visitors with disabilities	COULD	Audible message	Sprint 4

Figure 10 Produktens Funktioner

Iteration 1 avlutades med att konstruera ett Use Case diagram enligt Use Case 2.0 [5]. En risk som kan inträffa när man konstruerar en Use Case diagram är att man glömmer bort problemet systemet egentligen ska lösa och börjar uppfinna nya problem [4]. I vårt fall undveks risken då vi definierade problemet på förhand och speglade intressenters krav i produktens funktioner. Use Case diagrammet kompletterade på så vis intressenters krav.

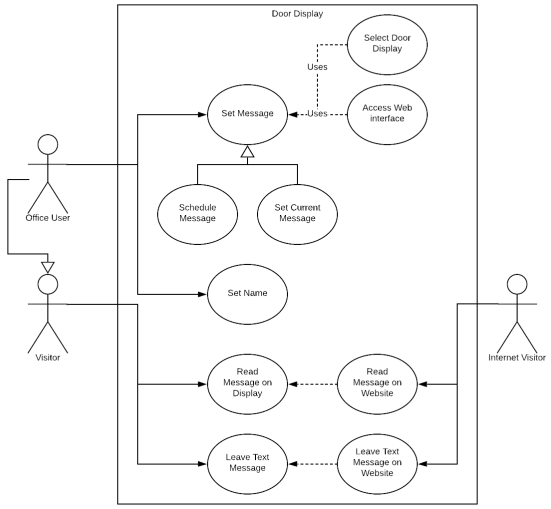


Figure 11 Use Case diagram

I början av denna sektion redogjordes det att en kundrepresentant identifierar intressenters krav, prioriterar dessa för utveckling och ser till att produktbackloggen speglar dessa krav. I resterande iterationer av projektet skapades det endast nya Use Case slices till de olika Use Case. Detta då allt grundläggande planering var i princip klar i iteration 1. Intressenters krav var prioriterade för utveckling enligt MoSCoW metoden. Intressenters krav var speglade i produktbackloggen med Use Case diagrammet och representerades i utvecklingen av produkten med de olika Use Case som valdes i början av varje iteration.

J. Arkitekt (Hugo)

Arkitektens uppgift är att utformar en systemarkitektur som resten av utvecklarna följer för att alla delar som utvecklas separat ska fungera tillsammans med varandra och med mjukvara från en tredje part.
Produktintressenterna (stakeholders) är själva gruppen och för gruppen är den viktigaste frågan är underhållbarhet (maintainability), vilket innebär bland annat:
Kostnadseffektivitet, de plattformar och tjänster (MongoDB: Atlas & Heroku) som används i projektet är helt gratis att använda. Om man har behovet att agera egen värd för tjänsterna finns den möjligheten också (Mongoose DB är open source och Heroku kan vara värd lokalt).
Kodbas och databas stabilitet, Med plattformar och tjänster som kan integreras med automatiserade tester försäkrar ökad kodstabilitet efter deployment. Arkitekturen måste också försäkra att ingen väntad eller oväntad operation kan korruptera databasen och därmed utsätta utvecklarna med uppgiften att återskapa databasen eller i värsta fall medge förlorad data.
Modularitet, lätthet i vidareutvecklings processen; REST API-arkitekturen har utformats för att vara helt åtskild från alla gränssnitt i front-end och kan utan förändringar i sig själv integreras i vilken lösning som helst med en internetuppkoppling som till exempel en smart IOT enhet. Denna arkitektur öppnar möjligheten för alla parter att använda API: n och bygga ett skräddarsytt användargränssnitt eller automatiserat gränssnitt.

Product characteristics	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilization, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable, and compatible with other systems that they use.

Essential attributes of good software [1]

Arkitekturens planeringsfas började arkitektur designen genom att konstruera ett användningsfallsdiagram [use case-diagram]. För att konstruera användningsfallsmodell diagrammet börjar man på användarsidan och extrapolerar sedan en modell utifrån systemets syfte och aktörens förväntade interaktioner med systemet. När Use case-modellen har slutfört valdes funktionerna och funktionerna i applikationen som behövs för att göra en MVP (minimivärde-produkt) som är den samling interaktioner med systemet (funktioner) som krävs för att göra en komplett lösning på problemet och uppnår syftet men utan allt som inte är absolut nödvändigt.
användningsfall diagrammet används sedan för att skapa en plan för implementationen.
De viktigaste funktionerna för att göra en MVP var följande:

- Visa meddelande / namn från databasen.
- Ställ in meddelande / namn i databasen.
- Öppet gränssnitt mot webben.

Funktioner som skulle göra den färdiga produkten men inte ansågs vara kritisk för MVP är följande:

- Planera meddelanden
- Inloggningsskydd och användarkonton.
- Ändra displayens visningsnamn.
- Display ägarskap.

Konstruktionsfasen påbörjades genom att etablera en kontakt mellan databasen och nodejs servern och sedan öppna en rutt för en http förbindelse med servern och en webclient. De individuella installations guiderna för Heroku och mongoDB följdes och någon exempelkod användes för att verifiera att en anslutning mellan Heroku-servern och mongoose-databasen kunde upprättas. Varje funktion/rutt konstruerades som planerat i den ordningen av mest betydelse för MVP. Innan varje publicering (deployment) uppdaterades API dokumentation så att projektgruppen kan implementera funktionaliteten i gränssnittet.

K. Testansvarig (Mohamed Mahdi)

Rollen som testansvarig ska se till att projektet ska verifieras och valideras regelbundet för att den slutliga mjuk eller

hårdvaran ska fungera som den planerades och ha god kvalitet. En person i ett scrumprojekt som har rollen testansvarig ska se till att projektet följer en testdriven utveckling. Detta kan göras genom att fastställa en testplan. Testplanen ska specificera strukturen projektgruppen ska följa. Detta underlättar ifall nya personer går med i projektgruppen.

Under projektets flera sprints så inleds de med att skapa en Test Case för varje Use Case Slice.

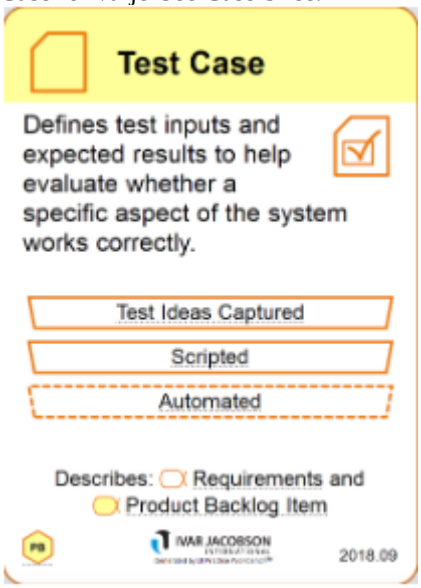


Figure 12 Test Case kort som används på arbetstavlan. [#]

Denna figur är den som används för att binda till de existerande Use Case Slices. Varje Use Case slice bestod av ett delområde av det stora systemet och va separat kod. Med att följa den förgjorda delning av systemet blev de lättare att veta vilken del av system som Test Caset tillhörde. För att bidra mer information till kortet så las notes på de som informerade vilken Use Case slice den tillhörde även självaste unika numret på testcaset.

Till varje Test case kort fanns det även Test kort. Deras syften var att kunna ha flera test som skiljer varandra mycket fast man ska kunna hålla koll vart testet hör ihop med i systemet. Detta görs genom att Test kortet som sätts upp av varje person i projektgruppen skriver kort text vad testet handlar om och vilken Test Case numer den tillhör till, detta skrivs på en note som sätts på kortet. När testet inte har utförts en så har noten som sitter över färgen röd men efter att den är godkänd att klara uppgiften byts färgen till grön och sätter kortet på Done i arbetstavlan.

VI. RESULTAT

Tabellen nedan listar bedömda ”områden” och subjektiv värdering gentemot deras bidrag till Åstadkommer/skapar projektet rätt saker (validitet) och konstrueras lösningar på bästa sätt (reliabilitet)? Alternativa arbetssätt och del-projektmetoder anges också.

Fullständiga tabeller finns i bilagor. Där finns också den evidens som ligger till grund för listade omdömen. Positiva omdömen anges med grönt, negativa med rött och neutrala omdömen med gult.

TABELLER II
”KEEP – PROBLEMS –TRY”
ROLL - STUDENT

”Keep”			
”Keep”	**Motivation**	**Förbättringar**	**Referenser**
Vision dokument	Gav en övergripande bild av projektet och möjlighet att påbörja utvecklingen och färdiggöra produkten snabbt.		
Use cases, Use case slices och task	Assisterade I uppdelning av utvecklingsuppgifter. Var även till nytta för att avgränsa projektet behålla fokuset på grundkraven.		Jakobson [5]
Komponentdiagram	Väldigt bra sätt att få en övergripande blick över hur alla delar ska passa med varandra och undvika cirkulära beroenden		
MVP	Minimum value product underlättade processen för att bestämma vad som var nästa logiska mål för varje sprint		
Teskkort	Det skapade tydlig ordning om vilka test som är skapade och vart exakt det tillhörde i det stora systemet	Skulle lägga lite mer tydliga instruktioner om deras användning. Eftersom deras riktiga nytta	

		visades bara i sista sprinten.	

”Problems – Try”			
”Problem ”	**”Try what?”**	**Motivering**	**Referenser**
Use case slicestatus.	Kanban tavla i lucid charts för att visualisera status av slices.	Det var svårt att veta nuvarande statusen för alla slices som tillhörde en use case. Man fick oftast reda på statusen mot slutet av varje iteration.	Jakobson [5]
Testplan	Få igång test-planen innan arbetet kommer igång.	Att skapa den i efterhand kommer bara strula då det leder till oklarhet	

”Problems – Skip”			
”Problem ”	**”Skip or replace, why”**	**Motivering**	**Referenser**
Riskhantering	Inte spendera så mycket tid på det.	Inte spendera så mycket tid på det.	

VII. ANALYS / FÖRBÄTTRINGSFÖRSLAG

L. Testansvarig (Mohamed Mahdi)

Som testansvarig så var uppgiften lite flummig över hur tester skulle utföras, detta leder till att andra medlemmar inte har någon aning om hur de ska lösas då de har andra uppgifter. Detta kan underlättas med hjälp av att studera teori bakom testing tidigt under projektet. Då kommer självaste test-planen redan va etablerad innan kod skrivs. I detta fall så blev självaste projektets uppgift löst innan test-planen blev klar. Som förbättring skulle man även tillägga kontroll på tester som är klara istället för att ta developers ord att testerna funkar.

M. Kundrepresentant (Abdirahman Abdullahi)

Enligt Jakobson ska man definiera vad det egentligen betyder att framgångsrikt implementera en specifik Use Case Slice och fokusera utvecklingen av Use Case Slicen mot att uppfylla tester man definierat på förhand [5]. I vårt fall kom litteraturen och teorin om teststrategier och tester i iteration 3. Produktens grundläggande funktioner var färdiga och delvis deployade i en Cloud miljö i iteration 2.

Då problembeskrivning var definierad och intressenters krav prioriterade och färdiga för utveckling i iteration 1 så bestämdes det att projektgruppen skulle gå vidare med utvecklingen av produkten även om möjligheten och relevanta informationen för att skapa tester ännu inte fanns. Detta beslut hade påföljden att resultatet av en implementerad Use Case Slices bedömdes subjektivt istället för att det skulle bedömas objektivt med tester. Mitt förslag för framtida projekt är att teori och litteraturen för testning läggs ut i iteration 2 eller att projektgrupper ges möjligheten att få tillgång till dessa under iteration 2

N. Arkitekt (Hugo Lindell)

Att bygga arkitekturen modulärt gjorde processen att samarbete med flera utvecklare väldigt enkel då arkitekturen väldefinierat hur de olika modulerna (databas, server, gränssnitt) skulle kommunicera med varandra. Att prioritera arbetsuppgifterna efter MVP underlätta beslutsprocessen för vilka delar av arkitekturen var kritiska för grundfunktionaliteten. En MVP är också ett väldigt bra mål för något att redovisa under en sprintdemo där hela principen av projektet kan redovisas för intressenter.

VIII. DISKUSSION

O. Metoddiskussion

(Abdirahman Abdullahi)

Validiteten av undersökningsmetoden beskriver huruvida metoden verkligen undersöker och utvärderar den ursprungliga frågeställningen. Projektgruppens undersökningsmetod är baserad på metoden illustrerat i figur 4. Validiteten av undersökningsmetoden granskas genom att beskriva olika problem som kan orsakas med av metoden gentemot dess empiriska utvärdering av undersökningsfrågan.

Undersökningsmetodens datainsamlingsprocess var retrospektiv och bestod av projektmedlemmarnas subjektiva utvärdering av deras erfarenheter för varje iteration. Denna process var induktiv och krävde att projektgruppen länkade en projekt-metodmässig orsak till deras negativa eller positiva erfarenheter.

Ett problem som kan uppstå är att erfarenheterna inte är relevanta till undersöknings-frågan. Ett exempel kan vara att en projektmedlem ändrar om frågan ”Vilka projekt metoder eller praktiker skulle du vilja behålla till nästa iteration” och istället besvarar frågan ”Vilka projekt metoder eller praktiker gillade jag och krävde minst arbete”. Data som dessa frågor genererar skiljer sig markant. Ena genererar data som är relevant och tillräcklig generell för att kunna användas till utvärdering av undersökningsfrågan medan den andra genererar data som är alldeles för specifik och inte användbart. Detta motverkades genom att ställa en följdfråga som krävde att varje projektmedlem skulle motiverar varför de skulle vilja behålla projekt metoden eller praktiken.

Reliabiliteten av undersökningsmetoden beskriver tillförlitligheten hos data metoden genererar. Reliabiliteten utvärderades genom att granska olika problem som kan minska tillförlitligheten av data.
Ett problem som kan uppstå är att projektmedlemmarna blir alldeles för bekanta med frågorna i den retrospektiva möte under undersökningens tidsutsträckning. Detta kan påverka kvalitén av data insamlad negativt då projektmedlemmarna kan bli trötta på de upprepade frågorna. Detta motverkades till en viss grad i undersökningen genom att ha möten vid tider som passade medlemmarna bäst. Undersökning av hur stor påverkan detta problem hade i kvalitén av data insamlad förblir till framtida studier.

Sammanfattningsvis kan man dra slutsatsen att undersökningsmetoden har en relativ hög grad validitet och att reliabiliteten är för tillfället tvivelaktigt. Reliabiliteten av undersökningsmetoden kan granskas genom att repetera undersökningsmetoden i framtida studier.

P. Resultatdiskussion

Testansvarig (Mohamed Mahdi)
Detta var inte en så himla stor uppgift med tanke om hur mycket kod den slutliga produkten blev. Detta medför att ifall saker går snett är den ganska enkelt att manuellt gå in och se exakt vart det kan strula till. Detta var metoden som gruppen starta med nät det gäller testning av systemet. Man skulle till och med kunna argumentera att tester kanske inte var så nödvändiga om produkten skulle sluta utvecklas efter att gruppen blev klara. Däremot så är den en självklarhet att tester är en viktig faktor om projektet skulle skalas upp eller fortsättas utvecklas.
Efter att teststrategier infördes började flera test skapas dessa kunde bestå av unit test, frontendramverk test. Dessa test blev sedan implementerade på pipelinen de deployas på. Ett sådant upplägg är väldigt bra då en kod som förstör något tidigare inte kommer igenom utan stoppas till det har blivit lösta. Detta minimerar rummet som saker kan gå snett i ett projekt och är en nyttig strategi för framtida projekt.
När det gäller Test Case och Test korten som skulle sättas på arbetstavlan så bidrog de stor nytta när de gäller struktur fast stor osäkerhet i början av projektet. Det tog flera sprint innan de blev kortens nytta blev behärskad då varje sprint bidrog med mer erfarenhet. Rollen som testansvarig har varit en unik erfaren som bidragit med mycket utveckling när det gäller arbete i en projektgrupp.

Arkitekt (Hugo Lindell)
Det huvudsakliga syftet med att etablera en stark arkitektur i projektet är så att alla utvecklare i gruppen kan enkelt sammanfoga sina delar av koden och göra samarbetsprocessen så enkel som möjligt. Att alla delar kommunicerar med http gör det även enkelt för andra tredje parter att ansluta egna tjänster som bygger ovanpå vår. Att använda Use case model tillsammans med målet av en MVP tidigt i projektet fokuserade arbetet och gjorde att vi hade en färdig prototyp tidigt i projektarbetet.

Kundrepresentant (Abdirahman Abdullahi)
Baserad på mina erfarenheter utifrån som kundrepresentant är jag helt övertygad om att en bra projektmetod kräver tydligt definierade och specificerade krav som hanteras löpande under projektets gång.
I vårt fall anser jag att detta var en av de viktigaste orsakerna till att vi snabbt kom igång med produkten och lyckades snabbt färdigställa en MVP. Detta då alla i projektgruppen alltid visste vilka de viktigaste produkt funktioner vi behövde utveckla var.

Att vi lyckades utveckla en MVP tidseffektiv kan såklart även bero på andra orsaker såsom våra tidigare tekniska färdigheter, vana vid projektmetoder och graden av komplexiteten av projektet. Jag anser att dessa faktorer hade en påverkan med inte till samma grad som krav hanteringen.
Detta stöds av en vetenskaplig studie av Nasir och Sahibuddin där de analyserade 43 olika vetenskapliga studier som var och en undersökte vilka faktorer som leder till ett framgångsrikt projekt [6]. Resultatet Nasir och Sahibuddin fick var att 60.5% av de 43 vetenskapliga studier påstod att tydliga definierade och specificerade krav var en viktigaste och kritiska faktorer som leder till en framgångsrik projekt [6]. Medan endast 23.3% ansåg att projektets komplexitet var en kritisk faktor och endast 34.9% ansåg att tekniska färdigheter var en kritisk faktor för framgången av en projekt [6].

Q. Bidrag till vetenskaplighet, ingenjörserfarenhet (studenterfarenhet?)

(Mohamed Mahdi)
Detta IT- projekt har varit grundläggande för ett par studenter att testa på projektmetodiker. Dessa studenter befinner sig inte i ett sådan avancerat fällt att de kan utveckla vetenskapligheten med deras resultat. Men att resonera kring en befintlig projektmetoden kan klassas som ett bidrag då förbättringar kan tillkomma på grund av det.
Denna utforskning krig projektmetoderna har utvecklat gruppens medlemmar till att samarbete för att lösa större problem. Det har även givit medlemmarna en blick på hur framtida projektarbete kan se ut vilket är en stor nytta för framtiden

SLUTORD

(Mohamed Mahdi)
Under kursen gång var produkten som skapades en Door Display. Denna har två hemsidor, första som meddelande kan skapas och skickas och andra en skärm som visar upp meddelande.
Länk till första hemsidan:
https://projekt10.eu-gb.mybluemix.net/
Länk till andra hemsidan:
https://ii1302-screen.eu-gb.mybluemix.net/

REFERENSER

[1] Sommerville, I (2011), Software Engineering 9th ed. USA: Addison Wesley Professional.

[2] Scrum and experience from the trenches. Published by Henrik Kniberg, 2007.

[3] Tillgängligt projekt web, Agile och vattenfallsmodellen, sources from tillgangligt projekt.se/agile-och-vattenfallsmodellen/ at 2020-05-30.

[4] Bittner, K & Spence, I (2002). Use Case Modeling. USA, Addison-Wesley Professional.

[5] Jakobson, I, Spence, I & Bittner K (2011). USE-CASE 2.0 The Guide to Succeeding with Use Cases [Online]. Ivar Jakobson International SA. Available at: http//:www.ivarjakobson.cpm/sites/default/files/field iji file/article/use-case 2 0 jan11.pdf.

[6] Nasir, M, & Sahibuddin, S (2011). Critical success factors for software projects: A comparative study [Online]. Scientific Research and Essays Vol. 6(10). Available at: http://www.academicjournals.org/SRE.

[7] Andersson, N., & Ekholm, A. (2002). Vetenskaplighet - Utvärdering av tre implementeringsprojekt inom IT Bygg & Fastighet 2002.

[8] Eklund, S. (2010). Arbeta i projekt: individen, gruppen, ledaren: Studentlitteratur.

[9] Elvesæter, B., Benguria, G., & Ilieva, S. (2013). A comparison of the Essence 1.0 and SPEM 2.0 specifications for software engineering methods. Paper presented at the Proceedings of the Third Workshop on Process-Based Approaches for Model-Driven Engineering.

[10] Elvesæter, B., Striewe, M., McNeile, A., & Berre, A.-J. (2012). Towards an Agile Foundation for the Creation and Enactment of Software Engineering Methods: The SEMAT Approach. Second Workshop on Process-based approaches for Model-Driven Engineering (PMDE 2012).

[11] Kruchten, P. B. (1995). The 4+ 1 view model of architecture. Software, IEEE, 12(6), 42-50.

[12] OMG. (2013). Kernel and Language for Software Engineering Methods (Essence). 1.0.

Bilaga1

Projektgrupp 10
Projektdefinition

Abstract

Detta dokument är en projektdefinition (Eklund, 2010) för studentprojekt eller examensarbete vid KTH ICT.
En projektdefinition är inte en projektplan utan föregår ofta en sådan. Projektdefinitionen kan vid behov utvecklas till en projektplan. För examensarbetet är det lämpligt att projektdefinitionen fungera som ”överenskommelse” mellan projektets huvudintressenter vilka oftast är ett företag, studenten som gör arbetet och akademin varifrån studenten kommer. Förändras projektet i något viktigt avseende så uppdateras och förankras projektdefinitionen.

Dokumentversion, senaste överst

Datum	Version	Författare	Beskrivning
2020-03-20	<0.1 >	Sadeq Gholami	En första version av projektdefinitionen inför VT20 P4.
2020-05-25	<0.2 >	Sadeq Gholami	Final version

Innehållsförteckning

1 5
1.1 12
1.2 15
1.3 15
2 16
2.1 17
2.2 17
2.3 17
2.4 17
3 18
3.1 18
3.2 18
3.3 18
3.4 19
3.5 19
3.6 19
4 20
4.1 20
4.2 20
4.2.120
4.3 24
4.4 25
4.5 25
4.6 **Error! Bookmark not defined.**
5 27
6 27
7 28
8 29
9 30
Refferenser 30

Introduktion

1.1 Dokumentets syfte

Dokumentets syfte är att ge en helhetsbild på projekt arbetet ur olika synpunkter/ uppgifter. Dessutom ska dokumentet innehålla en grovplanering av hela arbetet.

1.2 Dokumentets omfattning

Detta dokument behandlar följande:

1. En kort översikt över projektarbetet/ uppgifter
2. En underlag till fortsatta planering
3. Vilka resurser har gruppen tillgång till
4. hur ser organisationen ut och vilka roller och ansvar finns i organisationen

Detta dokument behandlar *inte* följande:

5. En detaljerad plan och arbetsfördelning vilket alltså kommer att framgå i arbetstavlan och gantt schemat.

1.3 Dokumentöversikt

Detta dokument innehåller följande delar: (Eklund, 2010)

6. **Projekt- eller uppgiftsbeskrivning** – projektet innehåller två olika moment; dels ska man utveckla en produkt (resultat mål) dels ska gruppen tillsammans försöka och komma fram till en metod och svar till frågan “vad är en bra projektmetod för små IT-projekt”.
7. **Organisation** – Alla gruppmedlemmar har sina egna roll som de är ansvariga att spela enlig rollspel model. Uppgifterna är fördelade efter olika roller. Gruppen kommer att också ha gemensamma arbetspass då diskuteras de olika moment i kursen, förbättring av planeringar, och rapportering av de individuella uppgifter, och att ge feedback till varandras arbete.
8. **Projektmål** – Mål med detta projekt är att komma fram till ett bra projektmetod för små IT-projekt och kunna praktiskt genomföra det under kursens gång.
9. **Fas- och tidsplan** – Detta projekts olika moment ska klargöras under en period vilken består av 10 veckor. varje person i gruppen ska lägga 180/200 timmar på arbetet. Så kostnaden för projekten är atbetsresursern dvs de timmarna gruppmedlemmar lägger på arbetet. Så för att utnyttja resursen på ett effektiv sätt ska gruppen göra en Scrum inspirerade arbetstavlan, och iterations planeringen i Gantt schemat.
10. **Intressenter** – Intressenter till detta projekt är examinator som kommer att bedöma uppgiften och kursrapportet. Hans förväntning är att gruppen ska skriva en bra rapport där de svarar på frågan “vad är en bra projektmetod för små IT-projekt”. Dessutom ska alla i gruppen uppfyller kursens krav och når kursmålen. Ambitionen för att uppnå denna förväntning är att alla i gruppen förstår kursmålen, planerar genom att använda resurser på ett effektivt sätt.

    En annan intressent är användaren av den specifika produkten som kommer att utvecklas under arbetet. Denna intressentens förväntning är att produkten är färdig i god tid och uppfyller kravspecifikationen och är flexibelt för förändringar.

11. **Riskanalys** – En risk som finns just i denna speciella situation som alla befinner sig är att någon i gruppen eller examinatorn blir infekterat av Kovid-19 viruset. Då ska gruppen se till att planera så effektivt att arbetet inte ska påverkas på ett negativt sätt under sådana situation. Och att gruppen måste bli bra med att jobba hemifrån och använda de resurser som finns online.
12. **Förändringsplan** – Det finns olika sätt att diskutera och meddela varandra om förändringar i god tid. bl.a. grupp på discord, gruppens sida på Canvas, gruppens rum på Zoom och viktigaste av allt arbetstavlan .
13. **Kostnader** – den enda kostnad som finns för gruppen är arbetskraften dvs de timmar som gruppen lägger för projektet. Andra resurser finns aningen gratis online eller så får man det gratis genom skola(som IBMs cloud services)
14. **Dokumentplan** – det viktigaste dokumentet när arbetstavlan som är gjord med LucidChart, sedan ska gruppen tillsammans skriva kurs rapporten. Man ska skriva en iterationsplanering inför varje iteration. Gantt schema ska fixas. projektets vision över produkten som utvecklas.
15. **Utbildningsplan** – Alla i gruppen ska läsa Scrum häftet, introduktion till agil projekthantering, häfte om Scrum, artikel om Agile Tools, Exjobbsrapport från Finland av Annika Katter, 2015. Plus att det finns några roll specifika material som ska läsas individuellt. Alla i gruppen ska gå på alla föreläsningar och kolla på de olika introduktions videor.
16. **Referenser** – Eklund, Sven. Arbeta i projekt - individen, gruppen, ledaren. 4 ed. Studentlitteratur, 2011.

2 Projektöversikt – bakgrund, syfte och mål

Detta kapitel ger en översikt av projektet.

2.1 Bakgrund

Detta projekt består av två olika delar. En del handlar om en forskning som gruppen ska tillsammans göra och hitta svar till frågan; “vad är en bra projektmetod till små IT-projekt” vilket ska skrivas som en kursrapport.

Den andra delen är att utveckla en internetuppkopplad dörrdisplay vilket använder sig av internet of things. denna display ska visa det meddelandet man skickar från en webbapplikation.

2.2 Syfte

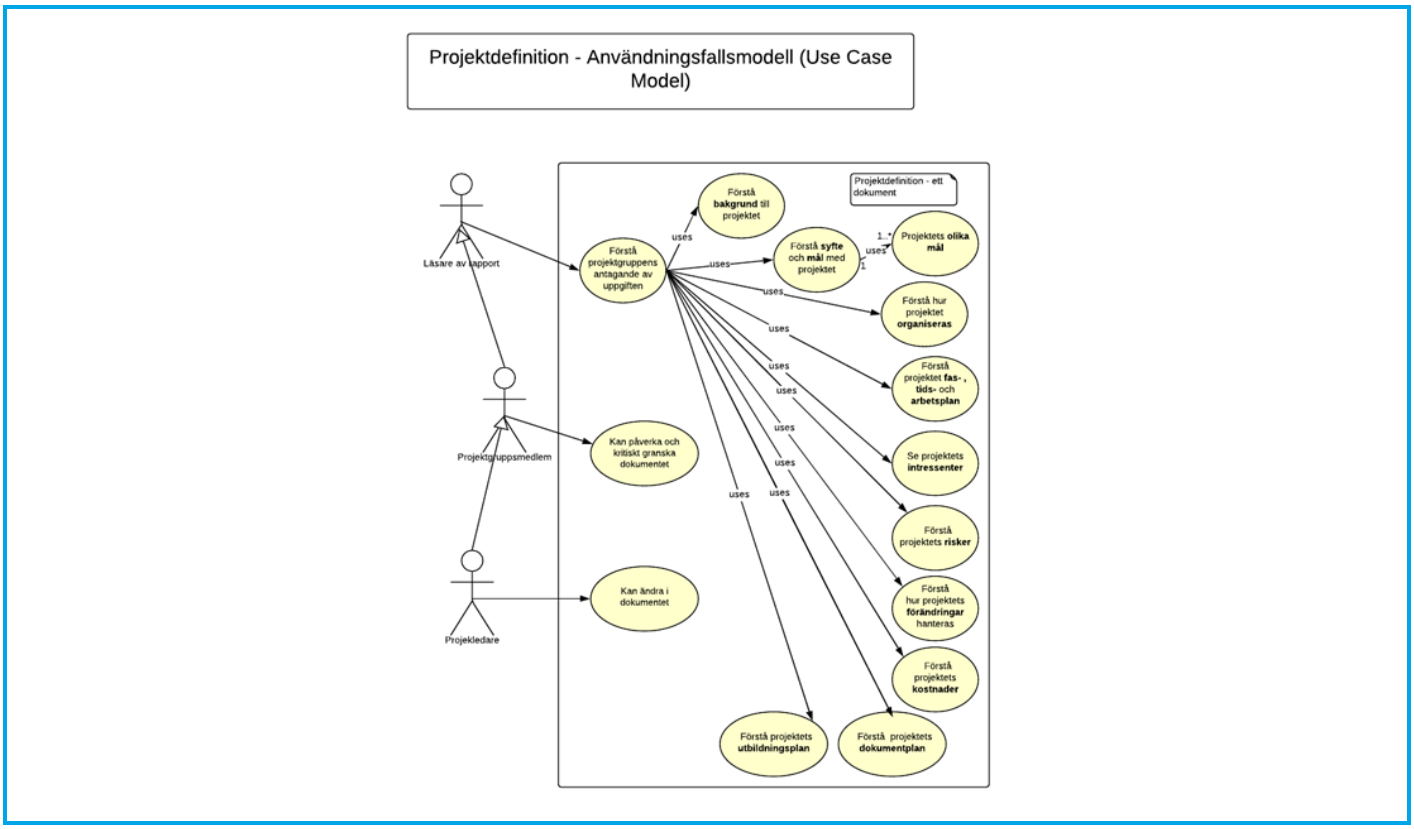
syfte med detta projekt är att komma fram till ett bra projekt metod och använda denna metod att utveckla produkten (dörrdisplayen och webbapplikationen)

2.3 Mål

- Att skriva en kursrapport som besvarar frågeställningen.
- Att utveckla produkten enligt kravspecifikationen

2.4 Funktionella krav - användningsfallsmodell

<Diagram: En användningsfallsmodell för dokumentet projektdefinition. Här skall istället finnas en första användningsfallsmodell för systemet som skall utvecklas.>



3 Organisation

3.1 Personer i projektet

Person	Kontaktinformation och beskrivning (adress, telefon ..)
Mohamed Mahdi	mmahdi@kth.se Tester
Hugo Lindell	hulind@kth.se Arkitekt
Abdirahman Abdullahi	aabdul@kth.se Kund och Kravansvarig
Sadeq Gholami	sadeqg@kth.se Projektledare

3.2 Möten

Gruppen har samlat sig regelbundet för att gå igenom uppgifter och vad som ska göras. Mötets längd bruka variera mellan kort och lång beroende om det ska samarbetande för att lösa en task tillsammans eller att alla gruppmedlemmar tar sin egen task och löser på egen tid separat.

3.3 Arbetsplats

På grund av okontrollerbar situation så måste alla gruppmedlemmar jobba hemifrån. Möten sker på Discord som tillåter gruppsamtal och skärmdelning av flera samtidigt. Detta möjliggör att gruppen kan jobba på en gemensam uppgift eller redovisa något som har blivit genomförd.

3.4 Arbetsutrustning

Majoriteten av arbetet kommer ske på datorn. Gruppen kommer att både planera projektet och skapa självaste produkten på datorn.

3.5 Meddelanden

Det är väldigt viktigt att gruppen ska kunna kontakta varandra väldigt effektivt. Detta görs genom en gruppchatt där alla typer av frågor ställs och även där möten samordnas. Det är även i gruppchatten som alla uppdaterar varandra om deras nuvarande status.

Alla medlemmar tar ansvaret att kolla chatten med jämna mellanrum. För att konfirmera att meddelandet har uppfattats så lämnas ett kort svar. Ifall tid har förflutit utan någon konfirmation brukas kontakt nås på privata nummer.

3.6 Webbplatser

Mycket sker online så de är många webbverktyg som behövs i en sån situation. Första hand så används Github för lagring av all kod som är skriven av gruppen. Gitsidan har även en egen wiki sida som förklarar kortfattat om projektet och alla medlemmars roller.

All dokumentation som är uppdelad är fortfarande sparad på samma plats. Gruppen har skapat en Google Drive som alla har tillgång till och allt är sparat i.

För att projektarbetet ska va tydligt för alla används en arbetstavla. Denna arbetstavla är inte en fysisk tavla utan är skapad på ett webbverktyg som heter ludichart. Där finns en privat mapp för gruppen som alla kommer in i.

4 Projektets olika mål

En Intressent till detta projekt är examinator som kommer att bedöma uppgiften och kursrapportet. denna intressents resultat mål är en kursrapport som besvarar på den nämnda frågeställningen(detta mål kan också vara projektmålet.) Effektmålet är att studenterna ska lära sig olika projektmetoder och komma fram till en lämplig metod för att utveckla den specifika produkten.

En annan intressent är användaren av dörrdiospalyen. Denna intressents resultatmål är ett fungerande dördisplay som visar meddelandet som man skickar via webbapplikationen. Effektmålet är då att göra det enklare att skriva ett meddelande i dörrdisplayen även om man inte är på plats.

4.1 Uppgiftsägaren

Hur färdig måste eventuell produkt bli? Hur skall en inkrementell utveckling ske för att uppgiftsägarens (kundens) förväntningar skall uppfyllas så bra som möjligt och tillräckligt? Vilka är de konkreta **resultatmålen** som skall göra att **effektmålen** uppfylls för uppgiftsägaren.

4.2 Kursmål och examensmål

Hur kopplar projektet till examensarbetets (själv-) bedömande och godkännande? Vilka är (projekt-) målen för att uppfylla akademins krav för en godkänd kurs eller ett godkänt examensarbete?

4.2.1 Vetenskaplighet

Värdera projektets vetenskaplighet/ingenjörsmässighet, använd rapport från LTH (Lunds Tekniska Högskola) som källa/referens (Andersson & Ekholm, 2002).

Hur kan detta projekt bidra till ökad vetenskap och/eller ingenjörserfarenhet? Vad är en vetenskaplig undersökningsmetod? Vilka intressanta frågeställningar kan formuleras ur detta projekt och som kommer att beskrivas i den akademiska rapporten.

Välj vetenskaplig metod och anpassa den för ditt projekt m a p ”projekttriangeln”, se kapitel 5.

Vetenskap kontra ”sunt förnuft”, se citat:

Kerlinger (1973) poängterar på samma sätt vikten av den vetenskapliga metoden och gör en distinktion mellan vetenskap och sunt förnuft utifrån fem punkter, vilka alla kretsar kring begreppen systematik och kontroll. Vetenskap inkluderar, till skillnad från sunt förnuft:

1. Användandet av en enhetlig begreppsapparat och teoretiska strukturer
2. Att systematiskt och empiriskt testa teorier och hypoteser
3. Kontroll, t.ex. att identifiera och styra de variabler som påverkar den studerade företeelsen och samtidigt systematiskt utesluta de variabler som inte berör den aktuella problemställningen
4. Att metvetet och systematiskt identifiera relationer mellan fenomen som berör problemställningen
5. Användningen av förklaringsmodeller.

Av punkterna framgår att vetenskap hanterar observerade fenomen och förklarar relationer mellan fenomen, vilka kan testas och utvärderas.

Figur 1: Tabell citerad ur (Andersson & Ekholm, 2002 som i sin tur citerar filosofen Mario Bunge)

Teknologisk forskning (se Andersson & Ekholm för "generell vetenskaplig metod"

Den generella vetenskapliga metoden kan anpassas för teoretisk, experimentell och teknologisk forskning. För den senare inriktningen, teknologisk forskning, följer den vetenskapliga forskningsprocessen följande steg (Bunge 1983):

1. Hur kan den aktuella problemställningen lösas?
2. Hur kan en teknik/produkt utvecklas för att lösa problemet på ett effektivt sätt?
3. Vilket underlag/information finns och erfordras för att utveckla tekniken/produkten?
4. Utveckla tekniken/produkten utifrån underlaget/informationen i steg 3. Om tekniken/produkten visar sig fullgod, gå till steg 6.
5. Försök med ny teknik/produkt.
6. Skapa en modell/simulering av den föreslagna tekniken/produkten.
7. Vad medför, alltså vilka är konsekvenserna av, modellen/simuleringen i steg 6?
8. Testa tillämpningen av modellen/simuleringen. Om utfallet inte är tillfredsställande gå till steg 9, annars gå till steg 10.
9. Identifiera och korrigera för brister i modellen/simuleringen.
10. Utvärdera hur resultatet i förhållande till befintlig kunskap och praxis, samt identifiera nya problemområden för fortsatt forskning.

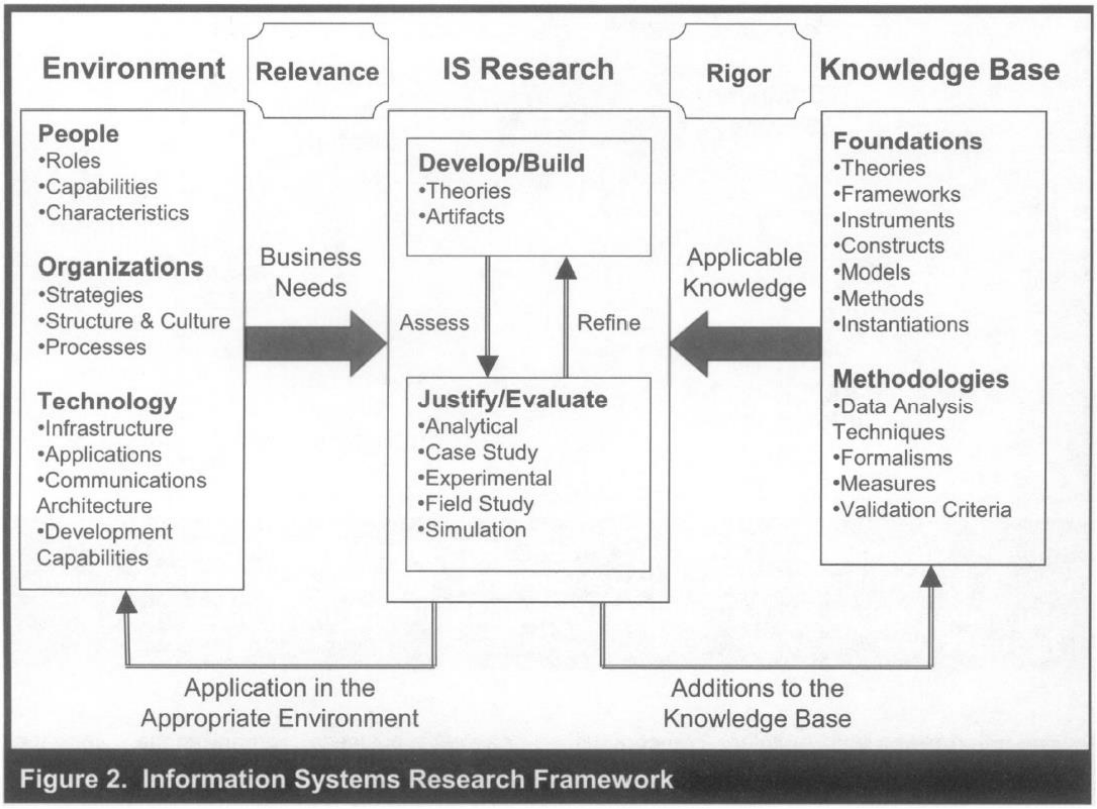
Figur 2: Tabell citerad ur (Andersson & Ekholm, 2002 som i sin tur citerar filosofen Mario Bunge)

Se även **Polyas metod** refererad i *(Eklund, 2010)* – förstå problemet, skapa en plan, genomför planen, kontrollera/utvärdera resultatet.

Vetenskaplig alternativ metod för utveckling av ICT-lösningar

Se också, som alternativ vetenskaplig metod, 7 st ”guidlines” (Hevner et al 2004): A. R. Hevner, S. T. March, J. Park, and S. Ram, “Design science in in-formation systems research,”MIS Q., vol. 28, pp. 75–105, Mar. 2004

(Artikel via KTHB/KTH-lan: https://www.jstor.org/stable/25148625?seq=1&cid=pdf-reference#references_tab_contents (figur ur artikel nedan)



Figur 3. Tabell citerad ur (Hevner et al 2004). Villkor enligt https://about.jstor.org/terms

Table 1. Design-Science Research Guidelines	
Guideline	**Description**
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Figur 4. Tabell citerad ur (Hevner et al 2004). Villkor enligt https://about.jstor.org/terms

Table 2. Design Evaluation Methods	
1. Observational	Case Study: Study artifact in depth in business environment
	Field Study: Monitor use of artifact in multiple projects
2. Analytical	Static Analysis: Examine structure of artifact for static qualities (e.g., complexity)
	Architecture Analysis: Study fit of artifact into technical IS architecture
	Optimization: Demonstrate inherent optimal properties of artifact or provide optimality bounds on artifact behavior
	Dynamic Analysis: Study artifact in use for dynamic qualities (e.g., performance)
3. Experimental	Controlled Experiment: Study artifact in controlled environment for qualities (e.g., usability)
	Simulation – Execute artifact with artificial data
4. Testing	Functional (Black Box) Testing: Execute artifact interfaces to discover failures and identify defects
	Structural (White Box) Testing: Perform coverage testing of some metric (e.g., execution paths) in the artifact implementation
5. Descriptive	Informed Argument: Use information from the knowledge base (e.g., relevant research) to build a convincing argument for the artifact's utility
	Scenarios: Construct detailed scenarios around the artifact to demonstrate its utility

Figur 5. Tabell citerad ur (Hevner et al 2004). Villkor enligt https://about.jstor.org/terms

4.2.2 Kursmål och examensmål

Den slutliga rapporten skall påvisa studentens uppfyllande av kursmålen och examensmålen, se listan nedan. Studenten utvärderar fortlöpande att rapporten kommer att innehålla dessa "bevis"och hela arbetet avslutas med en självvärdering som klargör hur rapporten påvisar måluppfyllelse. Självvärderingen redovisas för examinator.

Kunskap och förståelse

Studenten ska

1. visa kunskap om det valda teknikområdets vetenskapliga grund och dess beprövade erfarenhet samt kännedom om aktuellt forsknings- och utvecklingsarbete, och
2. visa brett kunnande inom det valda teknikområdet och relevant kunskap i matematik och naturvetenskap.

Färdighet och förmåga

Studenten ska

3. visa förmåga att med helhetssyn självständigt och kreativt identifiera, formulera och hantera frågeställningar och analysera och utvärdera olika tekniska lösningar,
4. visa förmåga att kritiskt och systematiskt använda kunskap samt att modellera, simulera, förutsäga och utvärdera skeenden med utgångspunkt i relevant information,
5. visa förmåga att planera och med adekvata metoder genomföra uppgifter inom givna ramar,
6. visa förmåga att utforma och hantera produkter, processer och system med hänsyn till människors förutsättningar och behov och samhällets mål för ekonomiskt, socialt och ekologiskt hållbar utveckling,
7. visa förmåga till lagarbete och samverkan i grupper med olika sammansättning, och
8. visa förmåga att muntligt och skriftligt redogöra för och diskutera information, problem och lösningar i dialog med olika grupper.

Värderingsförmåga och förhållningssätt

Studenten ska

9. visa förmåga att göra bedömningar med hänsyn till relevanta vetenskapliga, samhälleliga och etiska aspekter,
10. visa insikt i teknikens möjligheter och begränsningar, dess roll i samhället och människors ansvar för dess nyttjande, inbegripet sociala och ekonomiska aspekter samt miljö- och arbetsmiljöaspekter, och
11. visa förmåga att identifiera sitt behov av ytterligare kunskap och att fortlöpande utveckla sin kompetens.

4.3 Hållbarhetsaspekter och mål

- Kopplat till projektgenomförande

under hela projektgenomförandet ska gruppen se till att ha koll på olika miljöaspekter dvs använda resurser på ett effektivt sätt.

- Kopplat till produkt/tjänst som utvecklas dess användning och avveckling;

  Ett miljömål med produkten kan vara att använda sig av solenergi eller miljövänlig energi för att driva produkten. Dessutom ska man se till att produkten är inte gjord av miljöskadliga material, och om möjligt skapa det från återvinningsmaterial.

4.4 Etik, jämlikhet och likabehandling (JML) - mål

Gruppmedlemmar är alla killar däremot gruppen skulle gärna ha tagit hänsyn till jämlikhet om det var någon tjej i gruppen. när det gäller likabehandling ska alla i gruppen känna sig trygga. Alla ska få säga sina ideer och alla ideer ska tas på lika allvar.

I grupparbete bär etik, jämlikhet och likabehandling en stor vikt för att skapa en god arbetsmiljö. För att uppnå dessa mål har projektgruppen lagt stor fokus på att alla ska känna sig inkluderade i grupparbetet. Gruppen delar arbetsvikten jämställd så ingen känner att de får bära andras vikt. I denna projektgrupp har medlemmarna sina unika roller som har sina egna ansvar men gruppen samarbetar även ifall arbetet inte faller direkt i ens egna roll.
Det är också viktigt att lyssna på varandra i en grupp då vissa kan ha de svårare att få fram sin röst en andra.

Produkten som utvecklas i denna instans är en vara som är riktad till en stor målgrupp så den inkluderar majoriteten. I en väld där utvecklad teknik kan bryta lagar är det viktigt att behålla god moral. Denna produktanvändning bryter inga regleringar vilket är viktigt.

4.5 Arbetsmiljöaspekter och mål

På grund av utveckling av den nya coronaviruset COVID-19 så kan gruppen inte träffas fysiskt; gruppen använder sig av discords eller zooms gruppsamtal där man kan prata och dela sin skärm om det behövs och för att dela dokument så använder sig gruppen av google drive och github.

Projektet arbetas gemensamt på distans vilket möjliggör att alla har full kontroll av deras arbetsmiljö. Den tid som arbetas gemensamt i gruppsamtal är målet att den ska va effektiv och lugn. Men för att bevara en god arbetsmiljö så är de också viktigt att det är bra stämning i samtalet. Detta kan uppnås med att lära känna varandra vilket kan leda till att man känner sig mer bekväm i omgivningen. Att dela upp arbete mellan medlemmar i gruppen leder till att alla får planera sitt eget arbete. Det är viktigt eftersom alla inte

jobbar i samma takt och ifall det går för fort för någon kan det leda till stress, vilket skapar en dålig arbetsmiljö.

Produkten är riktad generellt till arbetsmiljöer som går i snabb takt. Då är det viktigt att produkten har en bra ergonomi för att användningen ska ske naturligt. Detta är ett viktigt mål för projektgruppen och därför tar en närmare titt på det.

4.6 Juridiska aspekter och mål

Det är viktigt att förlita sig på juridiken just när problem uppstår utan ska va med i projektets start. Projektgruppens har tagit nytta av utvecklingsverktyg som bidrar fria licenser. Dessa verktyg har inga krav på privata information utan skapas endast med mail och lösenord.

När det gäller produkten så kommer gruppen att utveckla medans de tar hänsyn till lagar angående säkerhet. Det ska allmänt finnas stöd att blockera andra från att ha tillgång till ens produkt. Det finns flera företag som erbjuder kryptering och säker lagring vilket är ett bra för ett litet projekt som inte har resurserna att lägga på säkerheten som krävs.

5. Intressenter

Lista vilka som är projektets intressenter, deras förväntningar och ambition att uppfylla dessa förväntningar och hur.

Inressent	Namn	Förväntningar
Examinator	Aners Sjögren	Att studenterna skall hitta en bra projektmetodik till små IT-projekt samt utveckla en produkt med åtminstone minimikrav.
Utvecklarana	Sadeq, Abdirahman, Hugo, Mohamed	Att jämföra de olika projektmetoderna och använda dem i praktiken för att utveckla produkten

6 Riskanalys

Nedan beskrivs identifierade risker.

Risklista

ID	Risk	Förebyggande åtgärd	Åtgärder vid riskutfall
R1	Att någon i gruppen drabbas av chrona	Att fördela arbetet på ett sätt att arbetet inte ska läggas ner om någon drabbas.	Att de uppfifter till andra
R2	Att man missar någon deadline	Alla i gruppen ska ha koll på deadlines	Diskutera med examinatorn och andra intressenter
R3	Att produkten inte är redo i tid	Se till att planera alla uppfigter på ett bra sätt och lära sig teorier	Få hjälp av läraren

	Hög sannolikhet			
Liten påverkan			R1	Stor påverkan
		R3		
			R2	
	Låg sannolikhet			

7 Kostnadsplan

Vilka kostnader finns i projektet? Vem betalar vad? Licenser?

Kostnader som finns i projektet förekommer i form av den tid som varje grupp medlemmer skall lägga på arbetet. Varje medlem skall lägga 180 timmer på arbetet samt undersökningen kring projektet resultat mål ”att komma på ett bra projektmetod för små IT-projekt”.
Utöver tiden kommer det att inte finnas några andra kostnader eftersom verktyg som annvänds i projektet som lucidchart, MS-office, NodeJS, IBM cloud tjänst osv, är alltingen grattis eller bjudas av KTH.

8 Dokumentplan

Vilka dokument skall användas, underhållas, granskas och levereras? När skall detta ske och för vilka?

	Dokument	Platform
1	Alla diagram	Lucid Chart
2	Arbetstavla	Lucid Chart
3	Use Cases	Lucid Chart
4	System diagram	Lucid Chart
5	Alpha-tillstånd	Lucid Chart
6	Tidsraopportering	Google arch
7	Gantt-Schema	MS Excel
8	Teknisk dokumentation databas	MS Word
9	Teknisk dokumentation meddelande applikation	MS Word
10	Teknisk dokumentation display applikation	MS Word
11	Teknisk dokumentation testning	MS Word
12	Testplan	MS Word
13	Utvecklingsplan	MS Word
14	Arkitekturbeskrivning	MS Word
15	Game Score Cards	MS Word

9 Utbildningsplan

Behov av förstudie, inläsning, utbildning.

No	Kunskap	Metod
1	IBM cloud	Föreläsningen om IBM samt IBM-lab
2	Scrum from trenches 2nd edtion	inläsning av teori
3	Agile projekthantering	inläsning av teori
4	Agile tools	inläsning av teori
5	Fallstudie av Annika katter	inläsning av teori
6	Sommerville	inläsning av teori
7	KanBan kompendium,	inläsning av teori
8	Artikel om "Essence"	inläsning av teori
9	Essence-standard	inläsning av teori
10	Exjobbsrapport (Testning och lärmodul)	inläsning av teori
11	GitHub	Miniprojekt
12	Arbetstavla	Föraläsningar

Appendix A - Referenser

Använd denna sektion för att ge fullständiga detaljer om alla referenser (artiklar, böcker url mm) som används i detta dokument.

Andersson, N., & Ekholm, A. (2002). Vetenskaplighet - Utvärdering av tre implementeringsprojekt inom IT Bygg & Fastighet 2002.

Eklund, S. (2010). *Arbeta i projekt: individen, gruppen, ledaren*: Studentlitteratur.

A. R. Hevner, S. T. March, J. Park, and S. Ram, (2004) “Design science in in-formation systems research,”MIS Q., vol. 28, pp. 75–105, Mar. 2004 (KTH länk)

https://developer.ibm.com/technologies/iot/tutorials/cl-mqtt-bluemix-iot-node-red-app/

Bilaga 2

Digital Door Display Project

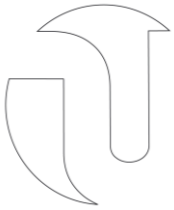
Web Application Component Description

Abstract

This document describes the required behavior and interfaces of the web application for the door display project.

Version History

Date	Version	Author	Description
14/05/2020	1.0	Sadeq Gholami	Initial draft
29/05/2020	1.1	Sadeq Gholami	Final



IVAR JACOBSON
CONSULTING

An Essential Unified Process Document

Table of Contents

- 1 Introduction 35
  - 1.1 Document Purpose 35
  - 1.2 Document Scope 35
  - 1.3 Document Overview 35
- 2 Required Behavior 36
- 3 Implementation Constraints 36
- 4 Provided Interfaces 5
- 5 Required Interfaces 5
- 6 Internal Structure 5
- Appendix A - References 6

Introduction

Document Purpose

The purpose of this document is to describe different components in the web application of the door display. The documents explain user interface, user interaction and the backend functions of the application.

The document is aimed at people who are interested in developing a front-end web application. Prerequisites for getting the full benefit from this document is having some basic understanding of JavaScript, React framework, NodeJS, HTML, CSS and interaction programming.

Document Scope

The scope of this document is limited to consideration of:

- The specification of the required behaviors for this component
- The specification of the interfaces that this component must implement
- Listing the required interfaces that this component depends upon
- Optionally specifying the internal structure and design of the component.

This scope of this document does not include consideration of:

- The unit tests that required to verify that this component conforms and performs to its specification
- The specification or implementation of any other components that depend upon this component or that this component depends upon.

Document Overview

This document contains the following sections:

- **Required Behavior** – behavior that is required from the component in terms of the responsibilities that it must satisfy and related requirements
- **Implementation Constraints** – constraints like implementation environment, implementation language and related patterns, guidelines and standards
- **Provided Interfaces** – the interfaces that the component must provide specified in terms of their signatures and constraints
- **Required Interfaces** – lists the interfaces that the component depends upon
- **Internal Structure** – optionally specifies the internal design structure of the component in terms of its constituent components and their relationships
- **Internal Element Designs** – optionally specifies the internal design of the constituent elements the component.
- **References** – provides full reference details for all documents, white papers and books referenced by this document.

Required Behavior

This section specifies the behavior that is required from the component in terms of:

- The responsibilities that it must satisfy
- The functions or services it must provide for its clients
- The non-functional requirements that relate to each of these
- Any other maintenance, documentation, reuse, control or logging requirements that relate to the component.

Web Application is the front-end component of the product “door display”. Its requirement is to make an easy interface to interact with the user. The user should be able to create an account, and log in safely. Then he/she should be able to create displays with different names, select a display to send a message to and then write a message to be displayed on the door display.

This component interacts with a RESTful-API also developed in the team in order to create account and manage the signing in process. It also interacts with the same RESTful-API to create new displays and send message to a certain display.

Implementation Constraints

This section specifies the constraints acting upon the implementation of the component, such as:

- The deployment environments within which it must operate
- What language(s) it must be written in
- The component frameworks it should operate within
- Design patterns it should conform to
- Other architectural standards and structures to which it must conform
- Guidelines that should be followed, including modelling guidelines and coding guidelines and standards
- Environment constraints such as memory footprint and other resource usage constraints.

The web plication is deployed on IBM cloud. The code written in react should be saved in GitHub. One important point is to make the manifest.yml file for the build process. To get the build version of the app one should run the command “npm run build”. The rest of the process such as deployment and hosting, is taken care of by IBM cloud services. [2]

The webapp is written in JavaScript language with React as framework. React is a component-based framework which has its own html syntax called JSX. In order to make the code more readable, easy to understand and code reuse (not to reinvent the wheel) some other libraries such as “React Bootstrap” and “Styled Components” are also used.

The webapp consists of five webpages; thus, login page, home page, displays page, message page and an about page. [3]

The design pattern used in the webapp is Reacts components-based pattern. Each component is either a container component (i.e. components with state) or consists of one or more other presentation components (i.e. components without state). The pros of this pattern are code reuse of presentation components. [1]

Provided Interfaces

This section specifies the interfaces that the component must provide in terms of their signatures and constraints.

The webapp interacts with the RESTful API (developed in the team) via fetch-API to connect itself to the database. This way the webapp can manage safe login process, saving users, messages and displays in database, which can then be fetched by the displays.

The webapp provides a graphical user interface that is easy to interact with and navigate. This interface contains of several webpages that the user can navigate through to create account, create new displays, and send messages to specific display.

Required Interfaces

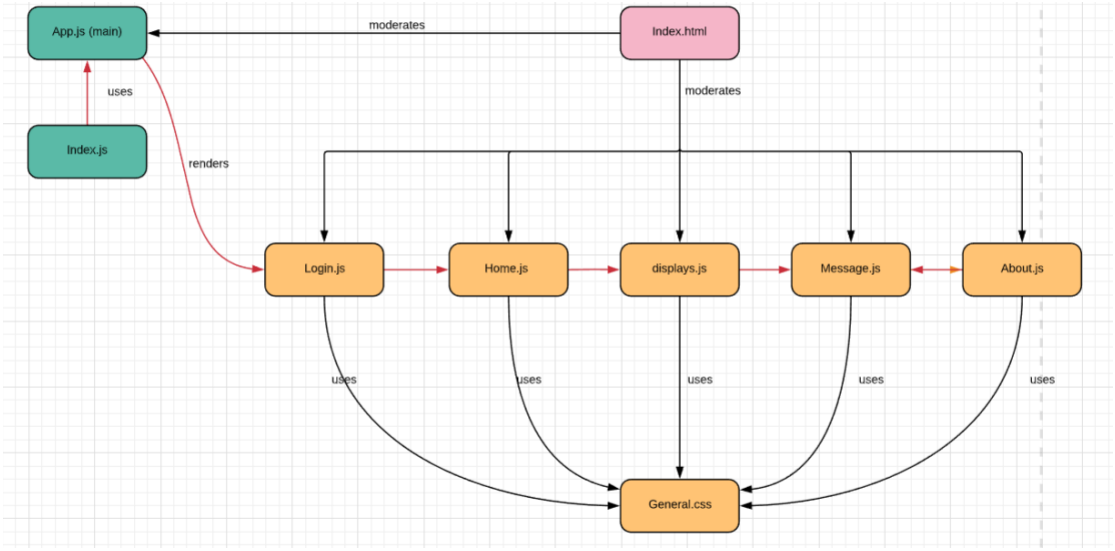
This section lists the interfaces that the component depends upon to fulfill its specified behavior.

Webapp is written in react which is a framework in JavaScript programming language. React uses NodeJS as runtime built, which compiles and runs java script code outside a web browser. For developers NodeJS is a necessity that the webapp depends upon. But once the application is deployed to a cloud service and hosted there, it requires only a browser and internet connection to fulfil its specified behavior. [1]

Internal Structure

This section specifies the internal design structure and the limits of the design structure of one the functionalities that the component should provide.

The following figure is the structure diagram of the application. It shows all the webpages that the application contains and how they interact with each other.

- 

All the components are rendered inside App.js and it is rendered inside index.js which is always run when the webapp is opened. App.js contains router that renders different components according to the URL. It starts by rendering login component when the user enters a valid username and password or if the user creates a new account it then renders the Home page. There the user can create new displays or select an existing one. After selecting a display the message.js component is rendered where user can write the message and send it to the specified display.

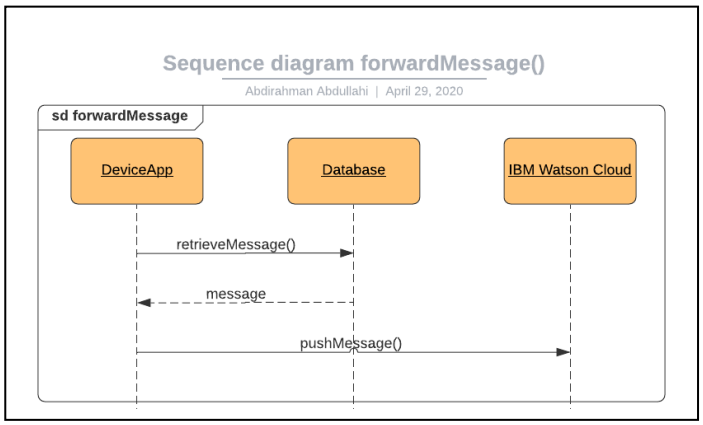
Appendix A - References

Use this section to give full reference details for all documents, white papers and books that are referenced by this document.

[1] https://reactjs.org/

[2] https://developer.ibm.com/technologies/iot/tutorials/cl-mqtt-bluemix-iot-node-red-app/

[3]



Bilaga 3

Group 10
Vision

1. Introduction

This document consists of a primary vision for the project.

2. Positioning

1. Problem Statement

IDEAL:
Ideally all visitors of an office would receive real-time information of the current availability and whereabouts of end-user.

REALITY:
In reality we use sticky notes and whiteboards to convey information to visitors. Whiteboards require special pens and regular upkeep whilst sticky notes require constant purchase and replacement.

CONSEQUENCES:
The consequences are that the user can't convey new relevant information to visitors without being physically present at the office. This leads to visitors rarely receiving up-to-date information. The current solutions also require regular upkeep and maintenance which leads to loss of valuable time and money.

SOLUTION:
A digital door display connected via the cloud to a web application. The web application can be used on any device with an internet platform such as a browser.

2. Product Position Statement

For	Knowledge workers
Who	have an office
The (product name)	is a digital door display
That	provides up-to-date information to visitors
Unlike	whiteboards and sticky notes
Our product	Is connected to the internet and can be updated in real-time from any location, on any device and at any time

3. Stakeholder Descriptions

1. Stakeholder Summary

Name	Description	Responsibilities
Developers	A student group consisting of four members working collaborating to deliver digital door display.	Developers responsibilities include the following: 1. Ensures that customer needs and requirements are accurately reflected in the product. 2. monitors the project's progress 3. Makes sure the project is maintainable (built on maintainable technology) and well-structured such that new features can be added later. 4. To be observant and deductive about the system so that it reaches up to the projects goal. Keeps a track of tests and marks the one who fail and those who pass.
Examiner	Course examiner that takes the role of end-client of the system.	The examiner is responsible for evaluating the developer's solution and each project member contributions.

2. User Environment

An important task for the target user is setting and updating the information displayed on the door display and its corresponding digital application. The task has the following attributes:

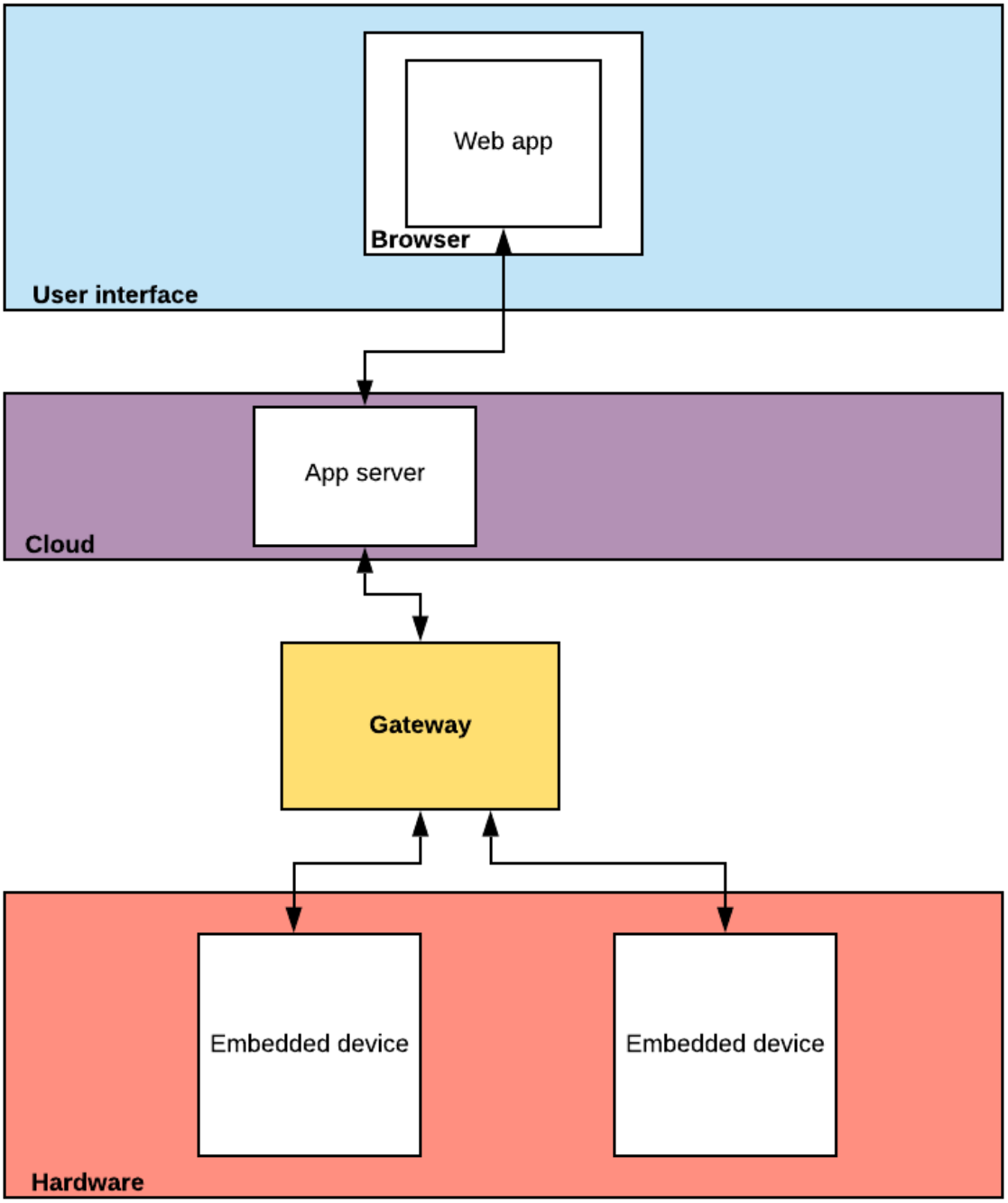
- Requires one person for completion.
- Takes the same amount of time as it would to send an SMS.

There are certain environmental constraints that can have the effect of hindering the target user of using certain features. An example would be if the users was in a location – in-flight or outdoors – where they could not establish an internet connection and therefore not be able to use the core functionalities of the product.

Applications in use that mimic to a lesser degree our products functionality are whiteboards and sticky notes. Our product does not need to integrate with these because it only requires a device with an internet platform such as a web browser.

4. Product Overview

1. Product Perspective



2. Assumptions and Dependencies

Assumptions involved in creating the features of the product are the following:

- User has a device with an internet connection and browser.
- Cloud services will be free to use.

3. Needs and Features

Need	Priority	Features	Planned Release
Secure and private access for users to their door displays.	MEDIUM	Secure Login	Sprint 3-4
Set the information to be displayed on door display in real-time.	VERY HIGH	Set Message	Sprint 1-2
See the information currently displayed on door display	HIGH	Display on device, Display on web application	Sprint 2-3
Convey the information audibly for visitors with disabilities	LOW	Audible message	Sprint 4

4. Alternatives and Competition

The following alternatives available to a user:

1. User builds a home-grown solution.

a. The strengths of this approach are that it can be tailored-fit to meet the user's requirements.

b. The drawbacks of this approach are that it is time costly and requires a financial upfront cost.

2. User maintains the status quo.

. The strengths of this approach are it requires minimal action for the user.

a. The disadvantages are that visitors seldom get up-to-date information. This increase the chances of conflicts occurring between the user and visitors and it also leads to a loss of productivity.

3. User creates a twitter page to keep visitors informed.

. The strengths of this approach are a user can deliver information is real-time to users from any location.

a. The drawbacks of this approach are that visitors are required to create an account on twitter and follow the user.

5. Other Product Requirements

1. Hardware requirements: Device with browser application and internet connection.
2. Performance requirements: The time it takes to complete the basic flow of the web application should be the same as in the basic flow of sending an email.
3. Documentation requirements: A user should have access to online help such as a tutorial to be able to get full benefit from the application both in functionality and usability.
4. Dependability requirements: The web application should always available to users and visitors (24/7) with maximum downtime of 5 minutes per day.
5. Usability requirements: The user should be able to use all system functions after receiving a tutorial or some form of labeling of functionalities. The online help is not to take require more than 3 minutes of effort from the end-user.

Bilaga 4

Digital Door Display Project

IOT Client Component Description

Abstract

This document describes the required behavior and interfaces of the IOT Cleint component.

Version History

Date	Version	Author	Description
30/04/2020	1.0	Abdirahman Abdullahi	Initial draft
13/05/2020	1.1	Abdirahman Abdullahi	Revised
16/06/2020	1.2	Abdirahman Abdullahi	Finalized

IVAR JACOBSON CONSULTING

An Essential Unified Process Document

Table of Contents

1 Introduction 3

1.1 Document Purpose 3

1.2 Document Scope 3

1.3 Document Overview 3
2 Required Behavior 4
3 Implementation Constraints 4
4 Provided Interfaces 4
5 Required Interfaces 4
6 Internal Structure 4
7 Internal Element Designs 4
Appendix A - References 5

1. Introduction

1. Document Purpose

The document is aimed at a person that is developing or is interested in developing a system to manage the dataflow between devices, cloud database and IBM IOT Platform and organisational management of devices.

Prerequisites for getting the full benefit from this document is prior knowledge in JavaScript, HTTP methods and Event-driven programming. The objective and purpose of this technical document is to describe the interfaces and implementation of the IOT Client.

2. Document Scope

The scope of this document is limited to consideration of:

- The specification of the required behaviors for this component
- The specification of the interfaces that this component must implement
- Listing the required interfaces that this component depends upon
- Optionally specifying the internal structure and design of the component.

This scope of this document does not include consideration of:

- The unit tests that required to verify that this component conforms and performs to its specification
- The specification or implementation of any other components that depend upon this component or that this component depends upon.

3. Document Overview

This document contains the following sections:

- **Required Behavior** – behavior that is required from the component in terms of the responsibilities that it must satisfy and related requirements
- **Implementation Constraints** – constraints like implementation environment, implementation language and related patterns, guidelines and standards

- **Provided Interfaces** – the interfaces that the component must provide specified in terms of their signatures and constraints
- **Required Interfaces** – lists the interfaces that the component depends upon
- **Internal Structure** – optionally specifies the internal design structure of the component in terms of its constituent components and their relationships
- **Internal Element Designs** – optionally specifies the internal design of the constituent elements the component.
- **References** – provides full reference details for all documents, white papers and books referenced by this document.

2. Required Behavior

This section specifies the behavior that is required from the component in terms of:

- The responsibilities that it must satisfy
- The functions or services it must provide for its clients
- Any other maintenance, documentation, reuse, control or logging requirements that relate to the component.

IOT Client is a component in the system that simplifies interaction with the IBM Internet of Things Platform. The component provides dual functionality and therefore has dual responsibilities as listed below:

1. The component is responsible for retrieving information from a database and publishing that information in the form of commands to all relevant embedded devices connected to the IBM Internet of Things Platform.
2. The component is responsible for providing a Restful interface such that client-facing web application can add, delete, and retrieve a list of all devices registered to the IBM Internet of Things Platform at any giving time. The Restful interface requires maintenance of an API documentation.

3. Implementation Constraints

This section specifies the constraints acting upon the implementation of the component, such as:

- The deployment environments within which it must operate
- What language(s) it must be written in
- The component frameworks it should operate within

IOT Client is a component that is part of a larger system and therefore requires other components to provide full functionality. These components are a database, the IBM Watson cloud and Device client (explained in section 6). Once these components are in place you can either run the component locally or on the cloud. In this section the procedures to executing the component in a local environment is explained. To start the component in a local environment run the following command:

```
$ npm start
```

The component will output “Connected the application.” which indicates that a connection to the IBM Watson cloud platform has been established.

4. Provided Interfaces

This section specifies the interfaces that the component must provide in terms of their signatures and constraints.

The component provides external servers within the system an interface to retrieve and interact with information from the IBM Watson Platform. The interface is provided through the HTTP protocol and adheres to the REST architectural constraints. As such the interface is defined with the following aspects: it maintains a base Uniform Resource Identifier (URI) and delivers different services with HTTP methods:

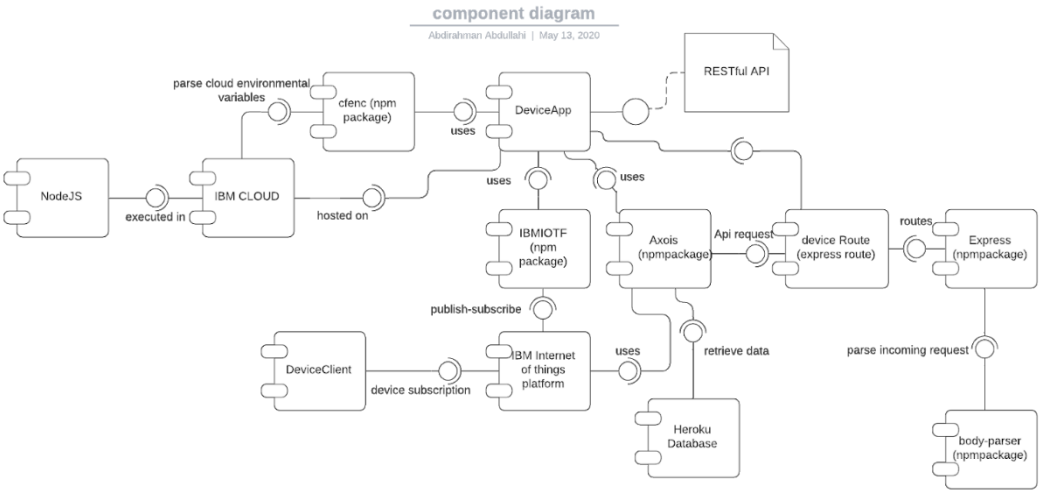
- Base URI: https://display-api.eu-gb.mybluemix.net/devices/
- Services:
    1. Add a device with a unique identifier to the IBM Internet of Things Platform.
    2. Delete a device specified by its unique identifier from the IBM Internet of Things Platform.
    3. Retrieve a list of all devices registered to the IBM Internet of Things Platform.

A use case of the provided interface within the system is for example that the frontend application adds a device through the interface services, which is then retrieved by the database using the interface service. This ensures that neither needs to communicate with each other or with the IBM Internet of Things Platform regarding the management of display devices which reduces coupling in the system.

The constraints of this interface it is always configured to a single IBM Internet of Things organisation. The specific organisation currently used can be changed through the interface corresponding configuration file, but the constraints of a single organisation connected still applies. This constraint has the effect that the interface cannot at present capabilities manage multiple organisations and therefore not be of benefit if the end-product is provide to multiple enterprise clients instead of individual clients.

5. Required Interfaces

This section lists the interfaces that the component depends upon to fulfill its specified behavior. The required interfaces are illustrated in the following diagram:



6. Internal Structure

This section specifies the internal design structure and the limits of the design structure of one the functionalities that the component should provide.

As explained in the section 1.1 one of the objectives of the component is to publish commands – which contain information – to devices. One of the ways the component can fulfil this requirement is through the MQTT protocol. Message Queue Telemetry Transport (MQTT) is a publish-subscribe network protocol built on top of the Internet TCP/IP protocol which transports facilitates the transportation of information [1].

The interface that is needed to facilitate MQTT communication between the component and devices is the IBM Watson cloud. IBM Watson cloud acts as a broker between the component and the Device client. The Device client is separate component that is responsible for connecting devices to the IBM Watson Internet of Things Platform. This interaction between the different constituents is illustrated in the following figure [2]:

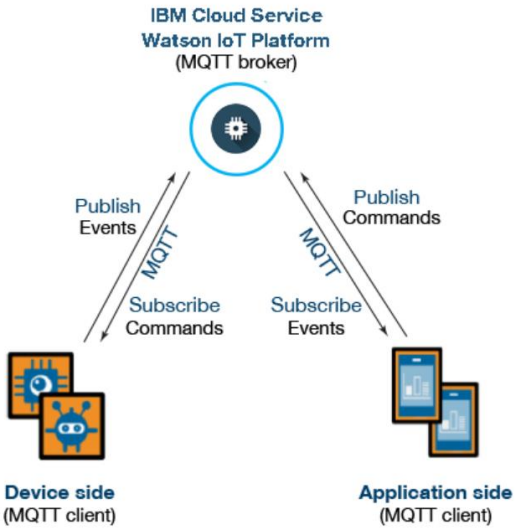


Figure 1

The component (application side in figure 1) subscribes to events from all devices connected to the IBM cloud platform through the MQTT protocol. The specific event type that the component subscribes to is “getCurrentMessage”. In other words, the component will only receive and perform action on events of that specific type.

The component publishes commands only to the specific device that requested a new message. The is achieved by each event containing the following information by which a device can be uniquely identified:

Type	**Description**
DeviceType	The type of the device that requested
DeviceID	The ID of the device that requested a new message

As briefly stated in the beginning of this section this architecture has certain limits. One of which is that it was originally designed for physical hardware and therefore does not translate well to digital door displays developed on the web.

7. Internal Element Designs

This section optionally specifies the internal design of the constituent elements that component.

The implementation is grounded in event-driven programming techniques. This means that the order in which statements are executed is determined by the order in which events are emitted. The IOT Client consist of three main sub-components: App.js, Server.js and Application.js.

1. App.js is a composition root and its main responsibility is to construct, instantiate and set up listeners for the other subcomponents.
2. Application.js consists of constructors, methods and interactions that are explained below:

a. Constructor: Establishes a connection and subscribes to events in the IBM Watson Cloud platform. When the application receives an event from the IBM cloud it emits an event to the listeners set up in App.js.

b. Listeners: Listens for a specific event from devices connected to the platform. Once that event is received the initiates the process of sending information to devices connected to the IBM IOT Platform. This process is illustrated in the following figure [3] and [4] in Appendix A.

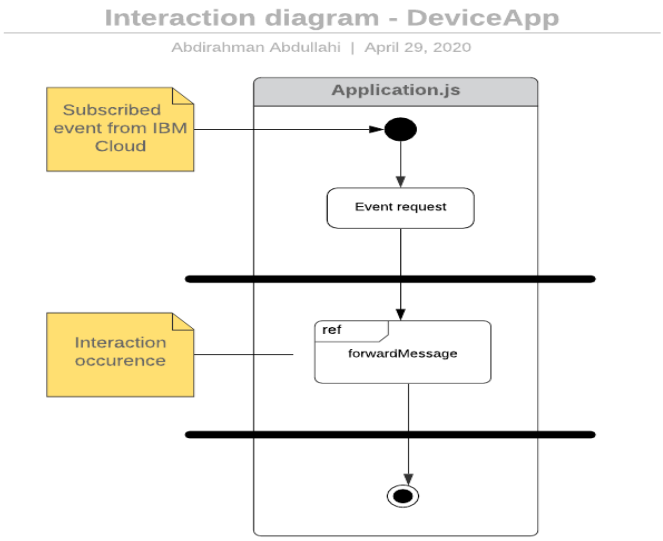
3. Server.js is the implementation of the API with an Express server and its different routes. The API consist of the three endpoints; list of all devices, add a device and delete a device [5].

Appendix A - References

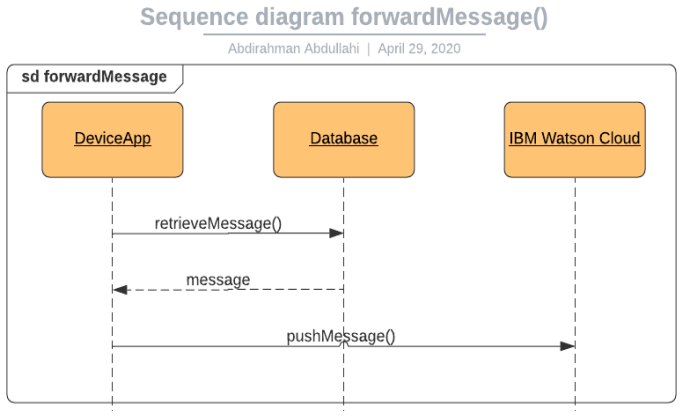
Use this section to give full reference details for all documents, white papers and books that are referenced by this document.

[1] https://en.wikipedia.org/wiki/MQTT

[2] https://developer.ibm.com/technologies/iot/tutorials/cl-mqtt-bluemix-iot-node-red-app/


Interaction diagram - DeviceApp
Abdirahman Abdullahi | April 29, 2020
Application.js
Subscribed event from IBM Cloud
Event request
Interaction occurence
ref
forwardMessage

[3]


Sequence diagram forwardMessage()
Abdirahman Abdullahi | April 29, 2020
sd forwardMessage
DeviceApp
Database
IBM Watson Cloud
retrieveMessage()
message
pushMessage()

[4]

[5]https://docs.google.com/document/d/1xHBOCAjTw56Za9U4bQAksAwbsILwbwB8vT7D-UXURyI/edit#heading=h.gsdl6wmgvfbt

Bilaga 5

Door display Project

Use-Case Model Survey

Abstract

This document provides an overview of the solution context, scope and high-level functional capabilities for the Door Display project. It catalogs the actors that interact with the system and the use cases that together describe all the ways in which they interact with the system. It also documents the use-case diagrams that show the relationships that exist between the actors and the use cases.

Version History

Date	Version	Author	Description
24/03/2020	1.0	Abdirahman Abdullahi	
27/04-2020	1.1	Abdirahman Abdullahi	

IVAR JACOBSON CONSULTING

An Essential Unified Process Document

Table of Contents

1 Introduction 3

1.1 Document Purpose 3

1.2 Document Scope 3

1.3 Document Overview 3
2 Use-Case Diagrams 4
2.1 Door Display 4
3 Actor Catalog 4
4 Use-Case Catalog 5
Appendix A - References 6

1. Introduction

1. Document Purpose

The purpose of this document is to provide an overview of the system use-case model in order to provide a high-level understanding of:

- Context – the people or things that interact with the system (the *Actors*)
- Scope – the things of value that the system performs for its Actors (the *Use Cases*).

2. Document Scope

The scope of this document is limited to:

- Diagramming and cataloging the system actors and use cases for the <Project> project.

The scope of this document does *not* include consideration of:

- Detailed specification of each use case – this is provided separately in a Use-Case Specification document for each use case.

3. Document Overview

This document contains the following sections:

- **Brief Description** – reminder of why the use case is needed
- **Use-Case Model Overview** – overview of the use-case model in the form of one or more use-case diagrams with supporting explanatory text
- **Actor Catalog** – catalog of all system actors
- **Use Case Catalog** – catalog of all system use-cases
- **References** – provides full reference details for all documents, white papers and books that are referenced by this document.

2. Use-Case Diagrams

This section provides an overview of the use-case model in the form of one or more use-case diagrams with supporting explanatory text.

1. Primary Use Cases

Figure 1 shows the primary use cases from the Door Display project use case model. The descriptions of the actors and use cases are given in the actor and use case catalogue that follow the figure.

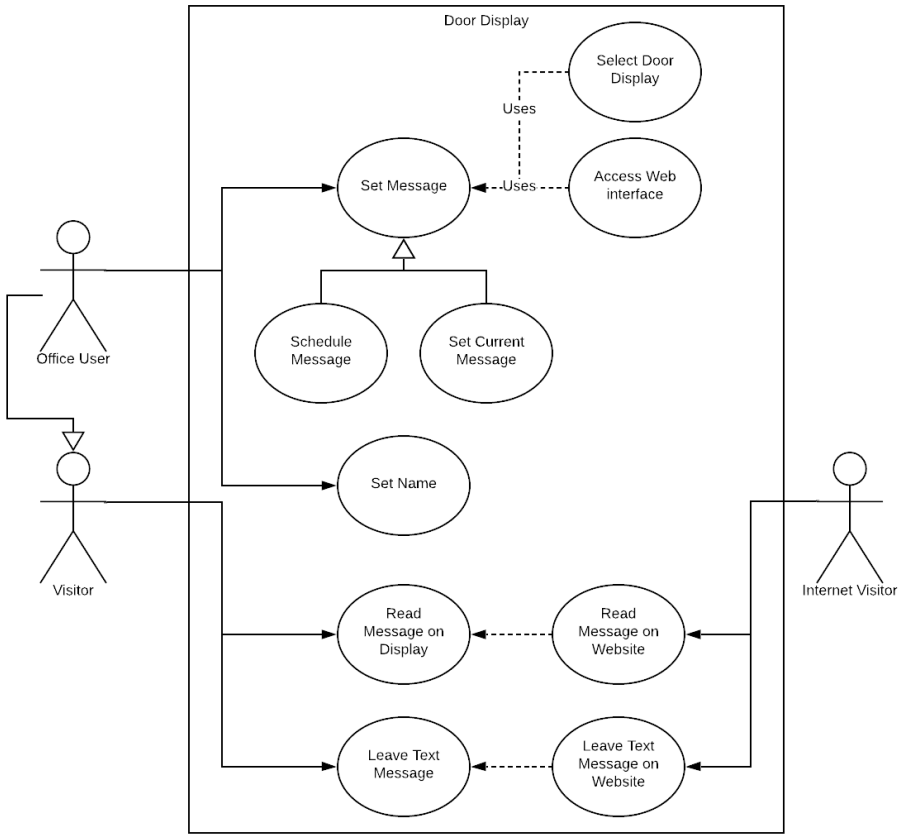


Figure 1

3. Actor Catalog

The table below catalogs the system actors, specifying for each actor:

- Name – unique and meaningful name for the actor
- Brief Description – summarizing the role that the actor plays with respect to the system.

Name	Brief Description
Office user	The Office user manages the web application to designate the information displayed on a specific door display.

Visitor	The visitor represents any individual that visits an office with a door display. The visitor reads or listens to the information displayed.
Internet-visitor	The internet-visitor represents any individual that visits a website that is connected to a door display. The internet-visitor reads or listens that is displayed on a door display through the internet.
Developer	The developer is responsible for the operations of the web application and analyzing the performance of the system.

4. Use-Case Catalog

The table below catalogs the system use cases, specifying for each use case:

- ID – unique identifier for the use case
- Name – unique and meaningful name for the use case
- Type:
  - *Base – end-to end interaction between an actor and the system*
  - *Inclusion – abstracted common part of many Base Use Cases that is explicitly referenced by these use cases*
  - *Extension – extension of one or more referenced Base Use-Cases*
  - *Abstract – use case describing generic aspects of many Base Use Cases which in turn specialize this general case in some way.*
- Brief Description – summarizes the use-case purpose in terms of the value produced for its actors and other stakeholders.

ID	Name	Type	Brief Description
1	Set message	Base	Describes how an office-user can set the information that is displayed on the door display and the visitor web application.
2	Schedule a message	Extension	Describes how an office-user can schedule information that is displayed on the door display at a specified time.
3	Select a door display	Base	Describes how an office-user selects a specific door display to interact with.
4	See message on display	Base	Describes how a visitor interacts with the door display by being able to read a message specified by an office-owner.
5	See message on web application	Extension	Describes how a visitor can interact with information displayed on the door display using the web application.
6	Leave message	Base	Describes how a visitor of an office can leave a text-based message on the door display.
7	Leave message on website	Extension	Describes how an internet-visitor can leave a text-based message on the web application of a linked door display.
8	Set Name	Base	Describes how an office-user can set a name (identifier) of their choosing to a door display on the web application.

<Door Display > Project
<Grupp10 > Test Specification

Abstract

This document provides a specification of the test methodology used in the project, by group 10 in the course II1302 Project and Project Methods.

Version History

Date	Version	Author	Description
25/05/2020	<1.0 >	Mohamed Mahdi	
27/05/2020	<1.1>	Mohamed Mahdi	

An Essential Unified Process Document

Table of Contents

1 Introduction 3
1.1 Document Purpose 3
1.2 Document Overview 3

2 Test Theory 4

3 Test Strategy 5

4 Risk Handeling 7

Appendix A - References 6

1. Introduction

1. Document Purpose

The purpose of this document provides a specification and reflect on the methodology the group studied and experimented to implement testing in the project.

The two parts of the document are theory on general test plans and the group method that was used under the projects run. The point is to compare and discuss them.

2. Document Overview

The document includes test specification that was used for the system that was developed by a team in the course II1302 Project and Project Methods, group 10. The group that developed was a small scrum team that had many different roles, but this document is directed to the test responsible role.

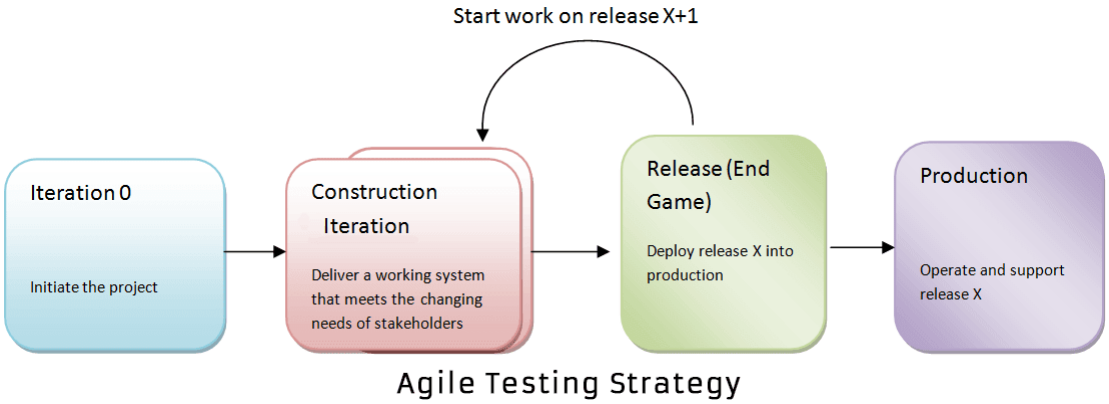
When testing is brought up it usually gets followed by verification and validation. A good test manager will create a good and structures test plan that will handle the verification and validation of the system under developing.

2. Test Theory

Behind every good leader there has to be studying part were as much possible information should be studied and analyzed to see if there is a practical way to implement it to their own project.

The project was following the Agile methodology which meant that testing would be done each sprint. The benefits with that is the testing is more unstructured then the typical waterfall approach. It better suited for small projects because tester and developers work closely and if there too big of personal it would not be practical.

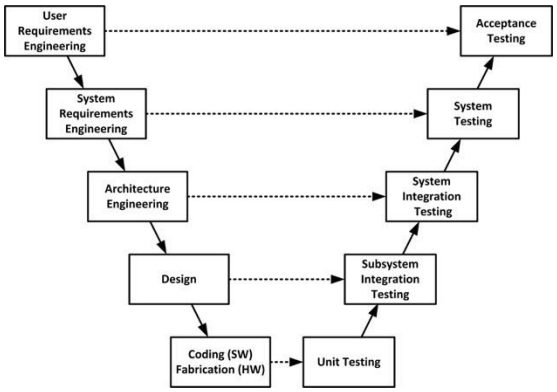
The testing life cycle spans through these four stages:



Agile Testing Strategy

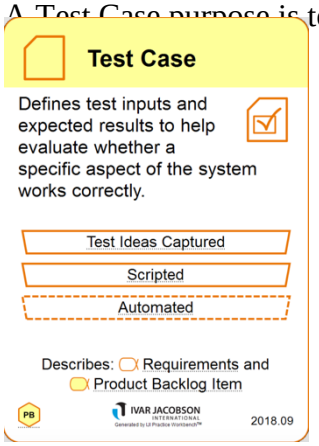
One strategies worth mentioning is the V Models for testing. As mentioned in section 1.2 verification and validation are the central parts for testing. Its typically preformed using the following 4 points analysis, demonstration, inspection and lastly testing.

The V model is a variation from the standard waterfall model. Its main focus is to emphasize verification and validation. How its created is by bending the bottom half of the waterfall model to create a V shape sequence, hence the name V model.



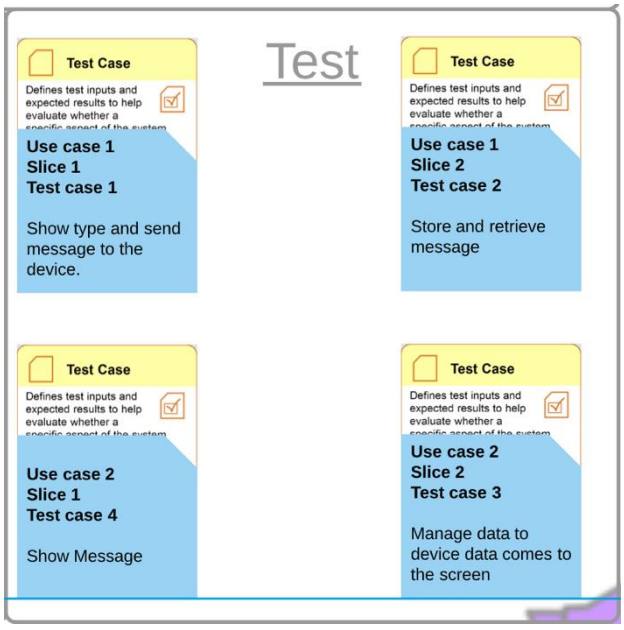
3. Test Strategy

A Test Case purpose is to define each department of testing and define if the system works as intended.



As seen in figure 1, these were used under the project to bind to an existing Use Case slice. Each of the Use Case slices had to do with a particular part of the system and were separate code so it is only logical to do the same with the test. Since the group used Ludichart as the workbench there could not edit the cards, instead a note was placed on top of the Test Case for relevant information.

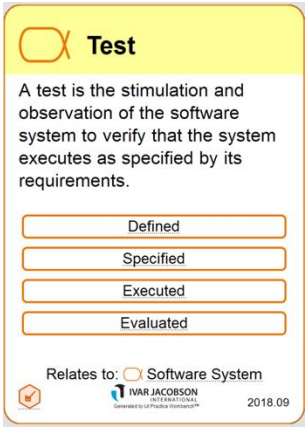
Figure 1, Test Case Slice



As seen in figure 2, The plan was to keep the information short but also have a system where it was easy to fin witch test had to do with which part of the system. The strategy was that the note informs what Use case it belonged to then which slice it handled and then its own identifier to keep track of all the test that belonged to that Test Case.

It included a short text as well to be as informing as possible if the numbers gets hard to keep track of.

Figure 2, Test Case on the work bench

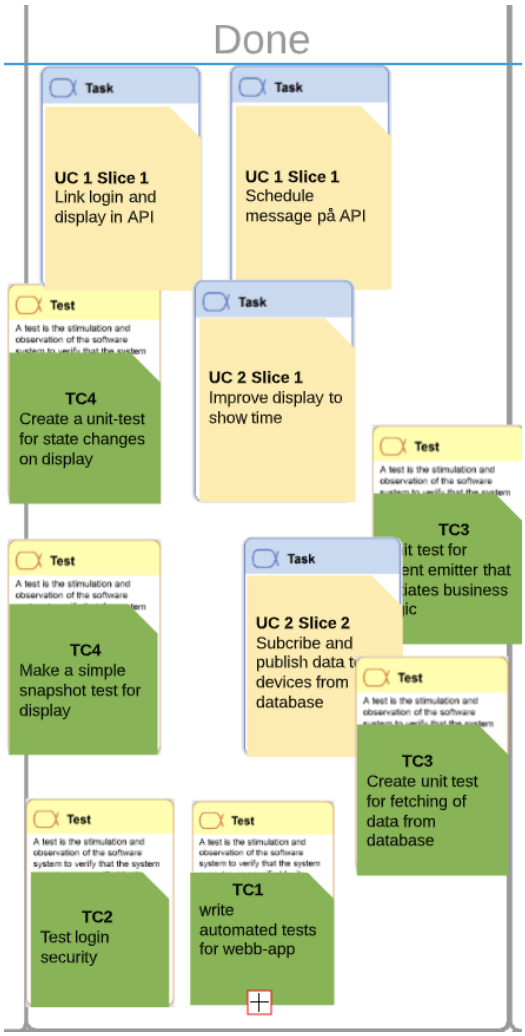


Each Test Case have multiple undercards that are Test cards. Here the leader of the test planning had to inform the other members of the strategy and needed them to follow the laid down structure.

The plan was for each member that handled their own Use Case slice under the development also were responsible to develop test for them. So, to keep track each member had their own Test Case Slice as well and had to add a Test Card (as seen in figure 3) to the work bench as well. They had to add a note on top as well with the information structure:

TC number, short text explaining test.

Figure 3, Test Card



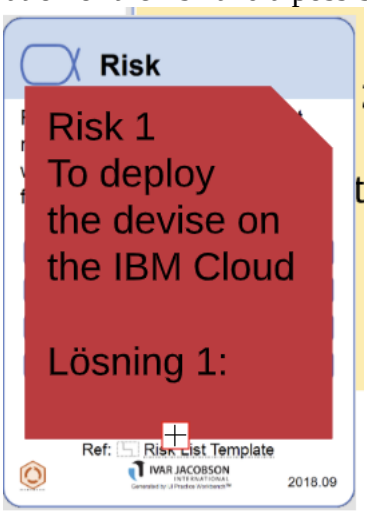
This will make it easy to keep track on who testing what and what Test Case slice it belongs to. To let people, know what they are working on, when they start with a test it's added to checkout with the color red on the note and when it's completed its switched to green and added to Done.

Slowly it all the cards will end up as green as seen in the figure 4. and that's when testing is completed of the project.

Figure 4, Test Card on the work bench

4. Risk handling

The risk handling was not that big of a focus during the projects run. In each sprint the group discussed if there were any risk that would come along the sprint. There was only one time that it the group faced a risk and it was handled by adding a risk card to the risk part of the workbench. It included short information of the risk and a possible solution. When solved it was added to Done.



Appendix A - References

[1] I. Jacobson, (2020, may) Essence kernel. [Online]. Available: https://practicelibrary.ivarjacobson.com/

[2] ReQtest, (2020, may) Agile Testing. [Online]. Available: https://reqtest.com/testing-blog/agile-testing-principles-methods-advantages/

[3] Guru99, (2020, may) Agile Testing. [Online]. Available: https://www.guru99.com/agile-testing-a-beginner-s-guide.html

[4] D. Firesmith, (2020, may) Using V Models for Testing. [Online]. Available: https://insights.sei.cmu.edu/sei_blog/2013/11/using-v-models-for-testing.html

Bilaga 7

Digital Door Display Project

Device Screen Component Description

Abstract

This document describes the required behavior and interfaces of the <Device Screen > component.

Version History

Date	Version	Author	Description
10/05/2020	<1.0 >	Mohamed Mahdi	Initial draft
29/05/2020	<1.1>	Mohamed Mahdi	Complete

An Essential Unified Process Document

Table of Contents

1 Introduction 3
1.1 Document Purpose 3
1.2 Document Scope 3
1.3 Document Overview 3
2 Required Behavior 4
3 Implementation Constraints 4
4 Provided Interfaces 6
5 Required Interfaces 7

6 Internal Structure 8
7 Testing10
7 Secound Display [Fetch] 12
Appendix A - References 13

1. Introduction

1. Document Purpose

This documents purpose is to go into detail of the component Device screen. It should make it clear on all the behavior and interfaces of the component. It should also be clear to explain all the hidden functions and features that is implemented but are naked to the eye in the code.

Upon reading this document it should enable a future developer to take over the component for future developing or maintaining. It also has to make it easy to test the components operations. To be able to do this the document should include all information including the dependencies that exist with other components.

2. Document Scope

What this document scope covers are the specification of the required behavior for this component. It also describes the interfaces that the component needs to implement. It dives deep to explain the required interfaces the component depends on.

As it is a pretty necessary the document also covers the internal structure and the test that have established.

3. Document Overview

This document contains the following sections:

- **Required Behavior** – behavior that is required from the component in terms of the responsibilities that it must satisfy and related requirements
- **Implementation Constraints** – constraints like implementation environment, implementation language and related patterns, guidelines and standards
- **Provided Interfaces** – the interfaces that the component must provide specified in terms of their signatures and constraints
- **Required Interfaces** – lists the interfaces that the component depends upon
- **Internal Structure** – optionally specifies the internal design structure of the component in terms of its constituent components and their relationships

- **Testing** – specifies the test that has been created to make it harder to destroy previews work and the thought behind them.
- **References** – provides full reference details for all documents, white papers and books referenced by this document.

2. Required Behavior

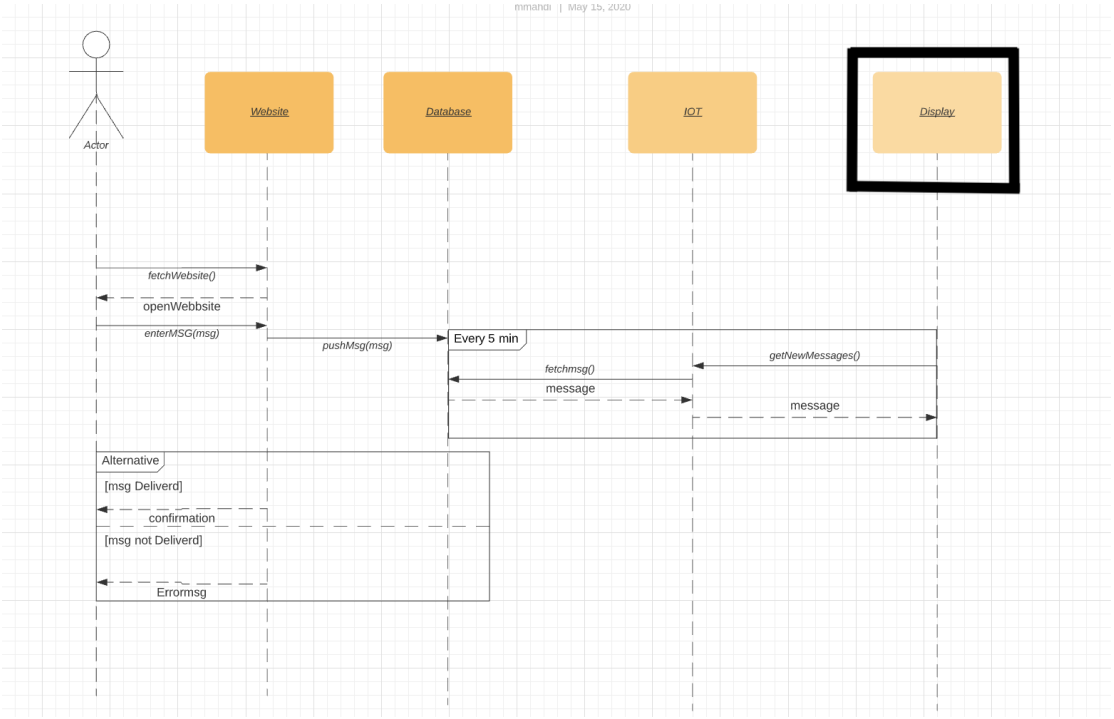
The requirement for the component is to deliver the wanted message and show it on the display for users passing by to see. For this to happen the component needs to parse incoming payload and move the string to the part of the component that is rendering it to the screen.

It's also implemented so that the user should know when the message appeared on the screen, so the timestamp has to be set when the screen is updated with message.

In the backend there has to be a refresher that periodically looks for if there is new message to show. This is done my setting a interval on specified time were the component repeats the call to the IOT where the device app is and responding with the latest message.

3. Implementation Constraints

The Device screen is an endpoint for a project called Door Display. There are total four part in the system and the device screen is only connected to the Device app. In the below picture you can see the Device Screen with the naming (Display).



The goal of the project is to deliver the string that is entered into the display and the display to show the string for the users passing by the display. To implement this the display has to get the message from the IOT part since that is the only component the display is aware of. It does this by a special request that is in the device library.

It starts with a call in “App.js” with the string “getCurrentMessage”

```
device.Push('getCurrentMessage');
```

Then it goes to “device.js” that uses the publish request that goes to the IOT part of the system.

```
Push(id) {
  this.device.publishHTTPS(id, 'json', JSON.stringify('Any new messages?'), 0);
}
```

If the id that is used when publishing is “getCurrentMessage” the IOT will then return the latest message.

To be able to catch the message from the IOT there has to be listeners set up in a specific way or the device won’t catch anything.

“device.js”

```
that.device.on("command", function (commandName,format,payload,topic) {
  if(commandName === "currentMessage") {
```

There should also be a filter that looks in the commandName is equal to currentMessage so that it doesn’t execute the code below on wrong payload.

The component is written in React and should be able to start in a local environment with the command

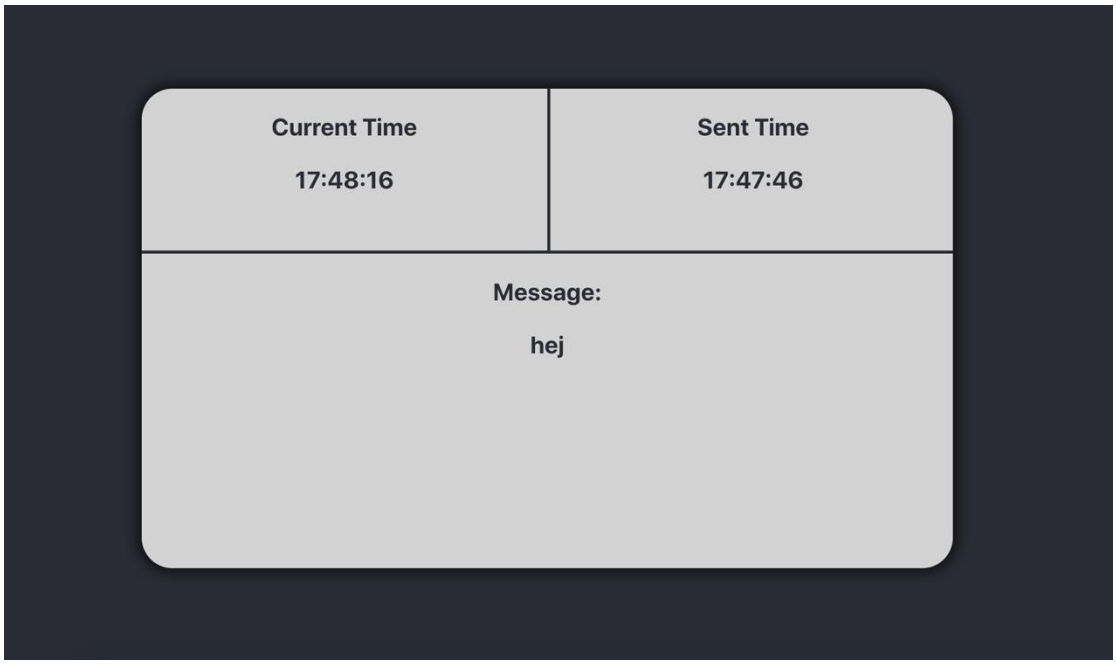
$ npm start

The component should bring up a display to the localhost3000 on the chosen browser

4. Provided Interfaces

The component Device screen acts as interaction part for the end user in the complete system. The meaning behind the system is to deliver message to a door display. We achieve this with a mock website that's acts as the physical door display.

The user is greeted by the picture below.



There are three main part that the user gets informed of. The first is the current time at the user location. The second is the time stamp that the message was sent so that the user know if the message was really

long time ago the message might be false. Lastly the message which is the part that is depended for it to be complete.

The constraint on this is that is only connected to one device on the IOT platform. This make switching only possible if the developer switches device id in the backend. The group developed a second version of the device screen component that can handle switching display. Will be brought up in section 7.

5. Required Interfaces

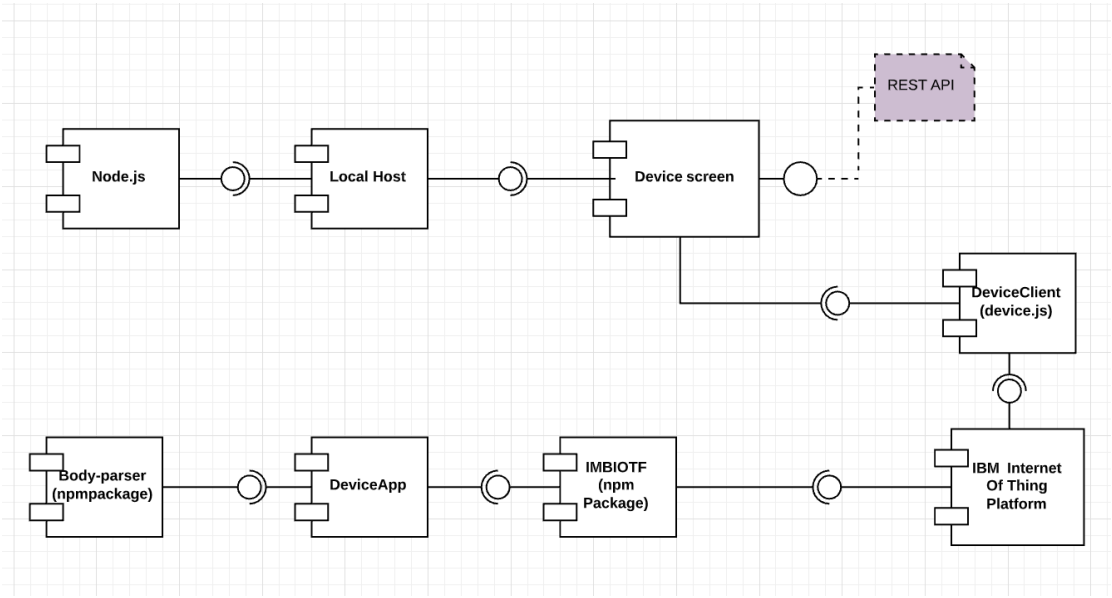
Since the project is created in react is **Node.js** an obvious necessity it is a JavaScript runtime environment that executes JavaScript code outside of a web browser.

These dependencies are needed in the node package:

- Ibmiotf: "^0.2.41"

And for testing the following are needed:

- enzyme: "^3.11.0"
-
- enzyme-adapter-react-16: "^1.15.2"
- react-test-renderer: "^16.13.1"



6. Internal Structure

When the device wants to achieve is to receive data from the IOT platform that is on IBM cloud. It also wants to send information to the platform which it does by publishing events.

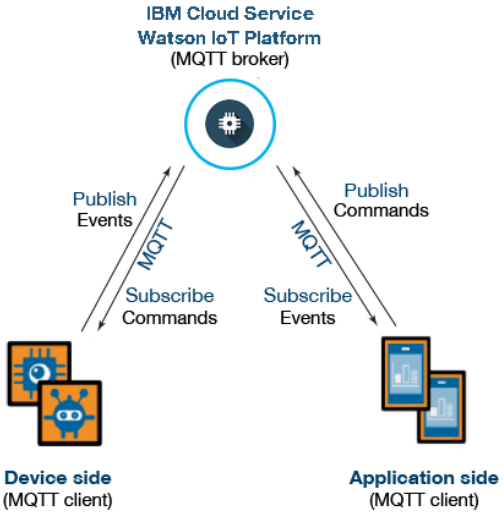
For the IOT devices to be functional they have to be connected to the internet. When an internet connection has been established, they can communicate through backend services.

Usually system work on the normal TCP/IP protocol but what makes this special is that it is using the protocol as an underlying base and running MQTT (Message Queue Telemetry Transport) on top.

Quoting IBM for the reason that MQTT is in use for the IOT developing. **[1]**

> *"The lightweight protocol allows it to be implemented on both heavily constrained device hardware as well as high latency / limited bandwidth networks.*
> *Its flexibility makes it possible to support diverse application scenarios for IoT devices and services."*

The naming might me confusing since it has nothing to do with messages, instead uses publish and subscribe. The picture below describes the chain and we can se that our device screen falls under client.



To be able to communicate to the platform the device has to connect to the device that is sitting in the platform. In the device screen this is established in the following picture:

```
constructor(org, token) {
  const device_config = {
    "org": org,
    "domain": "internetofthings.ibmcloud.com",
    "type": "IBM-KTH",
    "id": "0",
    "auth-method": "token",
    "auth-token": token,
    "use-client-certs": false
  };
  this.device = new iotf.IotfManagedDevice(device_config);
```

The const device_config has to follow the same structure so that the device will be able to connect. The org and token are forwarded from app.js but are the personal information on the chosen IOT platform. After establishing this there can be an setup device connection.

7. Testing

The device screen has a few automated tests that should be tested after any new feathers are implemented. These tests can be run on the command:

npm test

And the result will be a continues test that runs every time a change gets saved.

```
PASS src/App.test.js
  device module
    ✓ should match the snapshot (26ms)
    handleChange
      ✓ should call setState on title (4ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   1 passed, 1 total
Time:        3.874s
Ran all test suites.

Watch Usage: Press w to show more.
```

The tests consist of one that checks that the display variables that are shown on the screen are changed correctly. This is done by making a mock event and putting test data in it. Then that event should be sent to the display and change the display state with the events data. The way it passes the test is by comparing the mock event variable that was created to the variables in the display. If the change happed correctly the two data should match each other exactly. The following code is what test this:

```
describe("handleChange", () => {
  it("should call setState on title", () => {
    const mockEvent = {
      target: {
        name: "text",
        value: "test"
      }
    };
    function formatTime(){
      let now = new Date();
      return (now.getHours()<10?"0"+now.getHours():now.getHours()) + ":"
      + (now.getMinutes()<10?"0"+now.getMinutes():now.getMinutes()) + ":"
      + (now.getSeconds()<10?"0"+now.getSeconds():now.getSeconds())
    }
    const expected = {
      text: "test",
      currenttime: formatTime(),
      senttime:"",
      displaylink:"https://iot-display.herokuapp.com/display/get/",
      displayid:""

    };
    wrapper.instance().handleChange(mockEvent);
    expect(wrapper.state()).toEqual(expected);
  });
});
```

The second tests is shorter in code but serves a good purpose. Since this component is a screen and acts with the user its important that the view doesn't get affected by backend changes. This test takes a snapshot of the view and compares if it looks the same. This will prevent that something gets changed by accident. The following code is the test:

```
describe('device module',()=>{
  let wrapper;
  let mockSubmit;
  beforeEach(() => {
    mockSubmit = jest.fn();
    wrapper = shallow(<App submit={mockSubmit} />);
  });

  it("should match the snapshot", () => {
    expect(wrapper).toMatchSnapshot();
  });
```

Lastly it should be mentioned that these tests is also gets run every time a new code is tried to be deployed to IBM cloud. If the test fail it will stop the code to be push through to the building department of the pipeline. The problem should then manually be fixed retry the push to the pipeline. The following picture is the part of the log that shows if the test are passed.

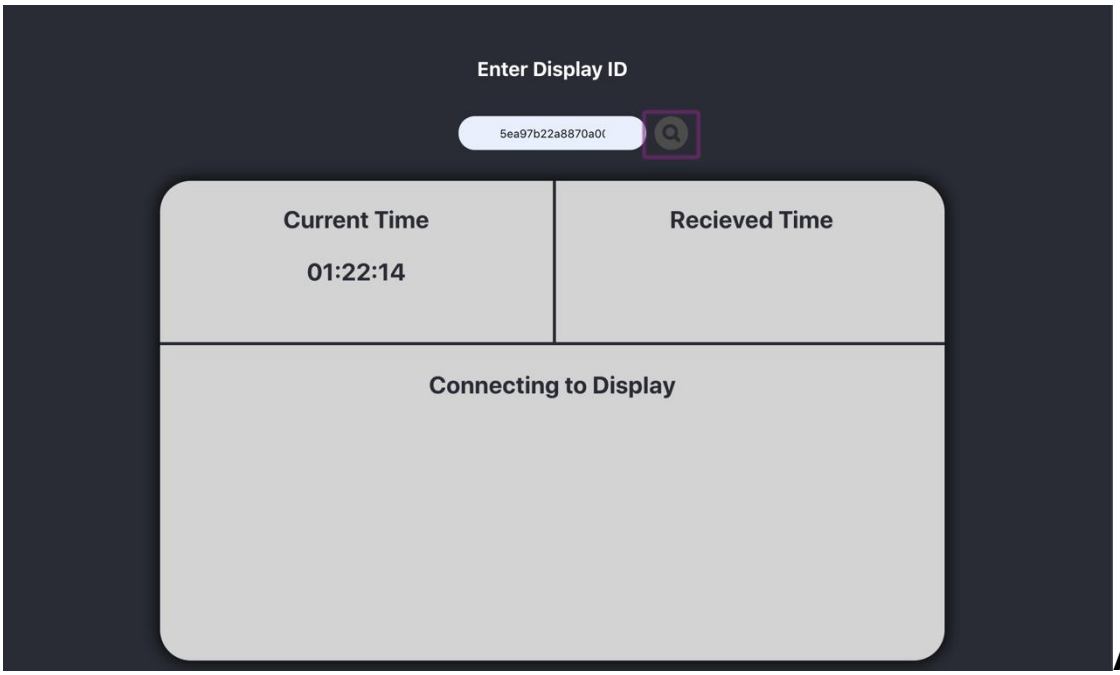
```
PASS src/App.test.js
  device module
    ✓ should match the snapshot (16ms)
    handleChange
      ✓ should call setState on title (4ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   1 passed, 1 total
Time:        2.499s
Ran all test suites.
Preparing the build artifacts...
```

8. Second display [Fetch]

Since having many display using IOT platform would make it a bit harder to implement and problem that occurred when trying to deploy the device the decision to make a display that uses fetch directly to the database was implemented. Mostly everything mentioned before is the same with the other component hence the no need to make a whole new complete document covering it.

The major change is the removed connection to the IOT platform instead uses Axios fetch and directly showing the return. There is a small improvement on this component that is an implemented interaction the user is capable now. The user can switch between the many displays that are created by the website (Website is show in section 3). The user is able to enter an Id that is bounded to a display making it possible to switch back and forth which display that is wanted to be shown. **[2]**



Appendix A - References

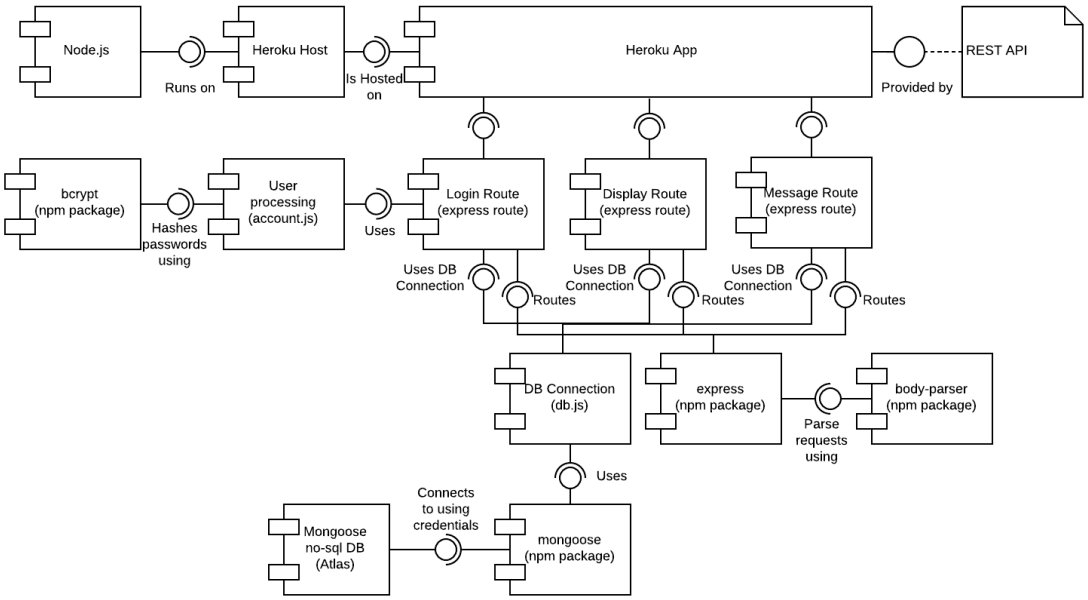
[1] M. Yuan, (Jan 7 2020) Getting to know MQTT, [Online] Available:

https://developer.ibm.com/articles/iot-mqtt-why-good-for-iot/

[2] M. Mahdi (May 28 2020) Display component on IBM cloud,[Online] Available:

https://ii1302-screen.eu-gb.mybluemix.net/

Bilaga 8


Node.js
Runs on
Heroku Host
Is Hosted on
Heroku App
Provided by
REST API
bcrypt (npm package)
Hashes passwords using
User processing (account.js)
Uses
Login Route (express route)
Display Route (express route)
Message Route (express route)
Uses DB Connection
Routes
Uses DB Connection
Routes
Uses DB Connection
Routes
DB Connection (db.js)
express (npm package)
Parse requests using
body-parser (npm package)
Uses
Connects to using credentials
Mongoose no-sql DB (Atlas)
mongoose (npm package)

Door Display Project

Door Display API Component Description

Abstract

This document describes the required behavior and interfaces of the Display API component. The REST API interface to interacting with the Door display system creating display instances, customizing the display and login.

Version History

Date	Version	Author	Description
26/05/2020	1	Hugo Lindell	Initial version

Table of Contents

1 Introduction 3
1.1 Document Purpose 3
1.2 Document Scope 3
1.3 Document Overview 3
2 Required Behavior 4
3 Implementation Constraints 4
4 Provided Interfaces 4
5 Required Interfaces 4
6 Internal Structure 4
7 Internal Element Designs 4
Appendix A - References 5

1. Introduction

1. Document Purpose

The purpose of this document is to describe the behavior and interfaces of the Door Display API component in order to:

- Enable the implementation effort to be sized and managed
- Establish the dependencies that exist with other components
- Support the independent development of this component
- Provide a basis for testing that the component operates as required.

2. Document Scope

The scope of this document is limited to consideration of:

- The specification of the required behaviors for this component
- The specification of the interfaces that this component must implement
- Listing the required interfaces that this component depends upon
- Optionally specifying the internal structure and design of the component.

This scope of this document does not include consideration of:

- The unit tests that required to verify that this component conforms and performs to its specification
- The specification or implementation of any other components that depend upon this component or that this component depends upon.

3. Document Overview

This document contains the following sections:

- **Required Behavior** – behavior that is required from the component in terms of the responsibilities that it must satisfy and related requirements
- **Implementation Constraints** – constraints like implementation environment, implementation language and related patterns, guidelines and standards
- **Provided Interfaces** – the interfaces that the component must provide specified in terms of their signatures and constraints
- **Required Interfaces** – lists the interfaces that the component depends upon
- **Internal Structure** – optionally specifies the internal design structure of the component in terms of its constituent components and their relationships
- **Internal Element Designs** – optionally specifies the internal design of the constituent elements the component.
- **References** – provides full reference details for all documents, white papers and books referenced by this document.

2. Required Behavior

This section specifies the behavior that is required from the component in terms of:

- The responsibilities that it must satisfy
- The functions or services it must provide for its clients
- The non-functional requirements that relate to each of these
- Any other maintenance, documentation, reuse, control or logging requirements that relate to the component.

The API purpose is to provide constrained and secure access to the database. Functions which the Door Display clients require from the API is a s follows:

- Secure login to access and store user information.
- Provide a list of owned displays.
- Set and schedule messages for individual displays.
- Get the current state/displayed info of a display.

While everyone have read access to the display info in the database a user account is required to write data to a display in the database. The API provides a way to authenticate a user using a username and password and give a signed in user access to the write functions of the API.

All REST functions (routes) should have proper documentation on the API wiki where every functions possible request parameters and the structure of the response is documented.

No user credentials should be visible in the server logs.

3. Implementation Constraints

This section specifies the constraints acting upon the implementation of the component, such as:

- The deployment environments within which it must operate
- What language(s) it must be written in
- The component frameworks it should operate within
- Design patterns it should conform to
- Other architectural standards and structures to which it must conform
- Guidelines that should be followed, including modeling guidelines and coding guidelines and standards
- Environment constraints such as memory footprint and other resource usage constraints.

The API is built upon nodejs[1] which constrains the platforms and operating systems the code can run on but not by much since nodejs runs on every major operating system such as Windows, Linux and MacOSX. Nodejs is a JavaScript ES6 runtime framework for server side network applications and was made with applications like this project in mind. The cost of using a very high-level platform like nodejs is that a server instance is very memory intensive because JavaScript is very memory intensive relative to lower-level languages like Java or C.

Express routes should be used to categorize/group different kinds of database functions, each route should be contained in individual source files and should serve no more than one purpose.

The input and output of each REST function should be logical and uniformly named parameters across all other functions such that no two functions use different parameter names for the same parameter type, for example every parameter reference to the session id should be named ‘session’. The response (output) of every function should include a Boolean value ‘accepted’ that determine whether or not the request passed, if not the a ‘error’ value should also be present containing an error message string. All unused paths should lead to a 404 response.

4. Provided Interfaces

This section specifies the interfaces that the component must provide in terms of their signatures and constraints.

The API component as a whole provides the REST API interface to the open web. A REST API uses the HTTP/HTTPS protocol to request and provide strings of JSON formatted text. This is the interface in which the database communicates with all other external components and devices.

5. Required Interfaces

This section lists the interfaces that the component depends upon to fulfill its specified behavior.

Node.js is a JavaScript runtime built using chromes V8 JavaScript engine. *“As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications.”*[1].

The following is a list of required nodejs package dependencies:
- bcrypt: ^4.0.1,
- body-parser: ^1.19.0,
- cors: ^2.8.5,
- express: ^4.17.1,
- mongoose: ^5.8.4,
- morgan: ^1.9.1,
- multer: ^1.4.2,
- nodemon: ^2.0.2

MongooseDB[2] is a no-sql database and in this implementation MongooseDB Atlas[3] was used to host the database but the mongoose node module can be used to connect to any mongoose database service or locally hosted mongoose database.

Heroku[4] is a web hosting service pipeline and is in this project used to build and host the REST API.

6. Internal Structure

This section optionally specifies the internal design structure of the component in terms of its constituent components and how these collaborate to deliver the required component behaviors.

The **Account component** is responsible for all interactions with the database related to account information such as account creation, login and session authentication. This component provides a protected interface for account actions for other components in the application.

The **Router component** is a simple express router that routes http requests to the intended component. This router routes all traffic from /login/ to the Login component, traffic at /display/ to the Display

component, traffic at /message/ to the message component, all other traffic is rejected. The full list of REST API functions are located in the API Documentation[5]

The **Login component** is the public interface for login related actions and exposes them as a rest API. These actions are characterized as user management and authentication. The component includes routes:

- POST at /login/ with {username, pwd} responds as accepted or error.
- POST at /login/create/ with {username, pwd} responds as accepted or error.

The **Message component** is the public interface for message related actions and exposes them as a rest API. These actions are characterized as message sending and scheduling. The component includes routes:

- POST at /message/set/ with {display, session, message} responds as accepted or error.
- POST at /message/schedule/ with {display, session, message, time} responds as accepted or error.

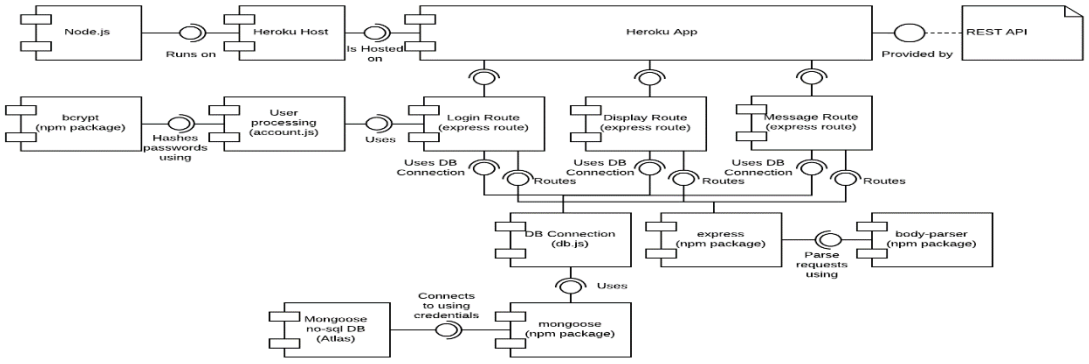
The **Display component** is the public interface for display related actions and exposes them as a rest API. These actions are characterized as display configuration related The component includes routes:

- GET at /display/get/<display> respond with a display object or error.
- GET at /display/get/all/ with {session} respond with a list of display objects owned by the specific user or an error.
- POST at /display/new/ with {session, name} responds with a display id or an error.
- POST at /display/name/set/ with {display, session, name} responds with a display id or an error.

The **Database component** is responsible for establishing a channel connection with MongooseDB Atlas using an authentication key provided by the service. The structure of the database is defined using the high-level mongoose database Schema. The Database Schemas are then provided to dependent components to enable them to interact with the database.

The **API view** is the public interface or REST API accessible via HTTP GET and POST requests. The API view is the consolidation of the Login, Display and Message components.

The following is a component diagram describing the component relations:



Symbol 1

7. Internal Element Designs

This section optionally specifies the internal design of the constituent elements that component.

Appendix A - References

Use this section to give full reference details for all documents, white papers and books that are referenced by this document.

[1] Nodejs https://nodejs.org/en/ 2020-05-26

[2] Mongoose get started: https://mongoosejs.com/docs/ 2020-05-09

[3] MongoDB Atlas https://www.mongodb.com/cloud/atlas 2020-05-26

[4] Heroku get started: https://devcenter.heroku.com/articles/getting-started-with-nodejs 2020-05-09-26

[5] API reference documentation https://github.com/sadeq-gholami/project-arbete-ii1302/wiki/API-Refference 2020-05-26

Bilaga 10

Door Display Project
Architecture Description

Abstract

This document describes the architecture for the KTH Group 10 Door Display project. It describes the approach selected to meet the architectural demands of the specified system. It further describes the process of selecting services and solutions to fulfill the vision and demands of the finished product.

Version History

Date	Version	Author	Description
06/05/2020	1	mrh lll	Initial Completed version

Table of Contents

1 Introduction 3
1.1 Document Purpose 3
1.2 Document Scope 3
1.3 Document Overview 3
2 Architecture Goals and Constraints 4
3 Architecturally Significant Scenarios 4
4 Architectural Views of the Design Model 4
5 Other Architecturally Significant Views 4
Appendix A - References 5

1. Introduction

1. Document Purpose

The purpose of the Architecture Description is to describe the approach selected to meet the architectural demands of the specified system. The architecture description:

- Describes how the architectural demands are met by the approach
- Explains the most important design decisions
- Specifies the aspects of the design that must be consistent across the system.

2. Document Scope

The scope of this document is limited to consideration of:

- Architecturally significant aspects of the system.

This scope of this document does not include consideration of:

- Detailed analysis, design or implementation of non-critical solution elements.

3. Document Overview

This document contains the following sections:

- **Architecture Goals and Constraints** – references the subset of the solution requirements and constraints that are architecturally significant in that they shape the architecture and are used to validate the architecture
- **Architecturally Significant Scenarios** – lists the scenarios that are considered architecturally significant, along with their dependencies and relative priorities
- **Architectural Views of Design Model -** contains a set of architectural views onto to the Design Model that together define the architecturally significant static and dynamic aspects of the system design
- **Other Architecturally Significant Views** – optionally describes other architecturally significant views such as analysis views, deployment views and implementation views
- **References** – provides full reference details for all documents, white papers and books that are referenced by this document.

2. Architecture Goals and Constraints

The architecture goals of the project is to make a flexible, reusable and secure API and database implementation. To make the project as flexible as possible, meaning making it easy to expand upon was to separate the database interaction from the backend logic and the outward interface. This was done by separating out these three components into their own file and designing a public interface for each of the components to communicate with each other. To make the project reusable each component is make as generic as possible by making each interface method only perform one particular action and chaining different actions rather than having one method preform an entire task. To make the project application secure we examine the most possible point of failure; the login component. The Login component is a self

contained component with a simple public interface to avoid unintentional use and is using a known secure, known tested off-the-shelf hashing methods provided by a third-party module called bcrypt[1]. Constraints adhering to the project are privacy protection legislation such as the EU regulatory GDPR[2]. This is a small scale project and is not high traffic enough to need to follow GDPR however would

3. Architecturally Significant Scenarios

This section lists the scenarios that are considered architecturally significant, indicates their dependencies and relative priorities and summarizes the approach that will be taken to implementing and validating these architecturally significant scenarios.

High priority Scenarios:

- The API must not crash when posting invalid data. The REST API will be exposed to the open internet and as a consequence one should never expect the posted data to be the correct format or contain the correct information which in a system that expects certain post parameters to be present or valid would suffer from crashes or bad operations when this is not the case. To solve this each post route validates that the data expected is present and valid and otherwise terminates the request with a error code 500 response to let the user know an error occurred.
- The data stored in the Database must not be corrupted under any circumstances. Inputting invalid data or duplicate keys could permanently corrupt the database and hinder proper operation, it is there for important properly validate user inputs and constrain the types of data that can be stored. Most corruption protection resides inside the database handler in this case MongooseDB[3] but some constraints are needed to be set by the developers such as preventing a user from creating an account with a username that already is present in the database or preventing the user from inputting a null value into any attribute that should not have one.
- Personal information must be kept private and secure. The authentication key to the database is stored inside the secure account section of Heroku and as such does not risk being visible in any released source code of the application. The only way to interact with the database should be using the API and keeping it that way will insure that all data output does not contain any unauthorized private user information.
- User passwords can never be stored in plain text. If a database breach occurs no readable passwords should be present and neither should any staff or database managers have access to a list of user passwords. To solve this a third-party password hashing solution bcrypt is used. bcrypt ensures that all passwords are hashed before stored and also handles passwords comparisons. bcrypt uses a combination of pepper and salt to hash the passwords for optimal password protection.
- A login session cannot last more than one hour. When a user signs in to an account a user session key is generated and stored on the client. This session key is only a valid authentication key for one hour, this ensures that a stolen session key can never within reason access the user account.

Low priority Scenarios:

- Architecture and code must be easy to debug. Solution provided by Heroku[4].
- Architecture must be easy to test. Solution provided by jest.

4. Architectural Views of the Design Model

This section contains a set of architectural views onto to the Design Model that together define the architecturally significant static and dynamic aspects of the system design, including the architecturally significant elements such as subsystems, components, classes and interfaces and the architecturally significant collaborations between these system elements.

The API is built to be modular with mainly 2 types of components. Route components exposes internal functionality to the web with a public REST interface on some URL path. Database components handle interactions with the database and provide interface for high level database actions. The end product mostly consists of high-level database interactions and no part of the application require any deeper computational component. The architecture for serving a request follows this path trough the component types:

Request □ Route Component □ Database Component □ Route Component □ Response

The requests move through the Route Component to the corresponding request route preforming one or more database action which returns a response back to the Route Component which finally responds to the initial request.

The node.js module express provides a layer of abstraction between the HTTP request/response server and the Route Component.

5. Architecture of Testing and Deployment

This section optionally describes other architecturally significant views such as analysis views, deployment views and implementation views.

The importance of testing an application cannot be understated and as such a reliable, easy to implement automated solution for testing a application at scale must be used and as such Jest (unit test) is used in this application since it meets these requirements. Application unit tests are run on a local dev environment using the NPM command ‘npm test’. The test framework used in this implementation was jest which is available through NPM. To get started with writing jest tests the jest reference documentation[6] provides all the necessary instruction on how to write any type of test. Jest tests are written for the most critical parts of the application that are hard to manually verify.

A modern rapid development and deployment practices a concept referred to as DevOps used in this project requires a fast and iterative development and deployment cycle. The deployment solution requirements of a DevOps operation are: Rapid and cooperative development with short windows between releases. Heroku meets these requirements as it uses the industry standard cooperative software solution git when automating its deployments.

Deployment of the production application is done though the Heroku CLI. Deploying a new build is as follows:

- Login to Heroku using the command ‘heroku login’ inside the local Heroku project.
- Commit the changes to the Heroku git using the standard git commands.
- Heroku immediately initiates a new build. Wait for the Heroku build to complete.

To debug eventual errors in the deployed build the Heroku log is used to see the standard JavaScript console log output stream. Access the log using the command: ‘heroku logs --tail’. Details on debugging in Heroku is available on the Heroku devcenter page[7]

Appendix A - References

Use this section to give full reference details for all documents, white papers and books that are referenced by this document

[1] bcrypt https://www.npmjs.com/package/bcrypt 2020-05-09

[2] GDPR tl;dr https://expresswriters.com/tldr-what-is-gdpr/ 2020-05-09

[3] MongooseDB get started: https://mongoosejs.com/docs/ 2020-05-09

[4] Heroku get started: https://devcenter.heroku.com/articles/getting-started-with-nodejs 2020-05-09-26

[5] API Refference https://github.com/sadeq-gholami/project-arbete-ii1302/wiki/API-Refference 2020-05-26

[6] jest documentation: https://jestjs.io/docs/en/getting-started 2020-05-09

[7] Heroku Devcenter: https://devcenter.heroku.com/articles/logging 2020-05-09

API url: https://iot-display.herokuapp.com/

Get information and messages of a display using 'displayId' in url.
```
GET display/get/<displayId>/
RES:
{
  display:{
    id: <ObjectID>,
    name: <String>,
    message: { text: <String>, time: <Date> },
    scheduled: <Array of Messages>,
    now: <currentTime>
  },
  error: <Exception>
}
```

Create a new display owned by a user using 'session' with 'name'.
```
POST display/new/
BODY:
{ name: <String>, session: <Session> }
RES:
{
  display:{
    id: <ObjectID>,
    name: <String>
  },
  error: <Exception>
}
```

Get the name of a display using 'displayId' in url.
```
GET display/name/get/<displayId>/
RES:
{
  display:{
    id: <ObjectID>,
    name: <String>,
  },
  error: <Exception>
}
```

Get all displays owned by user using 'session'.
```
```

```
POST display/get/all/
BODY:
{ session: <Session> }
RES:
{
  displays:[
    {
      _id: <ObjectID>,
      name: <String>
    }, ...
  ],
  error: <Exception>
}
```

Set the name of the display using 'id' and 'session' with 'name'.
```
POST display/name/set/
BODY:
{ id: <ObjectID>, name: <String>, session: <Session> }
RES:
{
  accepted: <Boolean>,
  error: <Exception>,
  echo: {
    id: <ObjectID>,
    name: <String>,
    old: <String>
  }
}
```

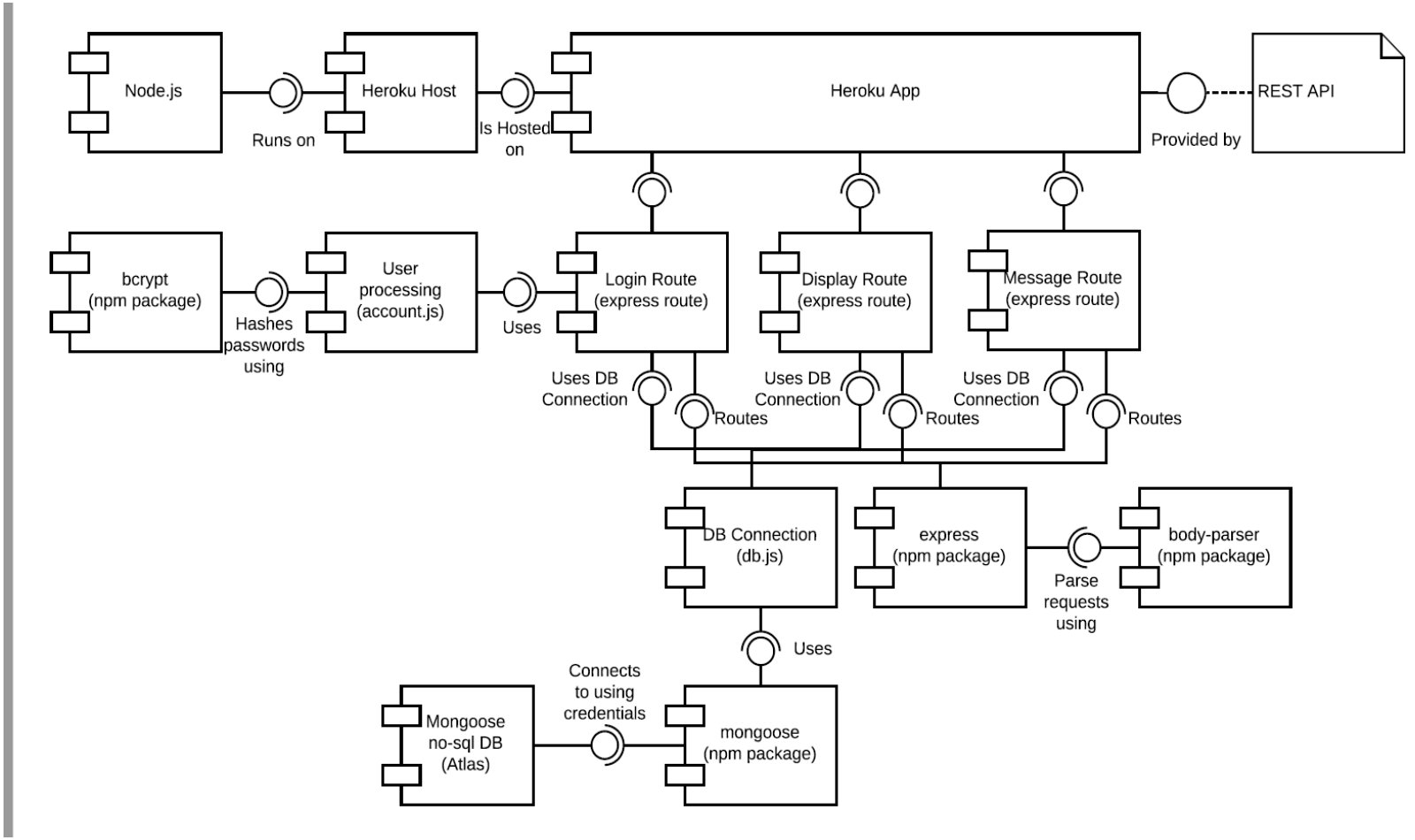
Set the current message using 'display' and 'session' with 'message' and ['overrides'].
```
POST message/set/
BODY:
{ display: <ObjectID>, message: <String>, overrides: <Time/Date>, session: <Session> }
RES:
{
  accepted: <Boolean>,
  error: <Exception>,
  echo:{
    display: <ObjectID>,
    message: <String>
  }
}
```

Schedule a future message using 'display' and 'session' with 'message' and 'time'.
```
POST message/schedule/
BODY:
```

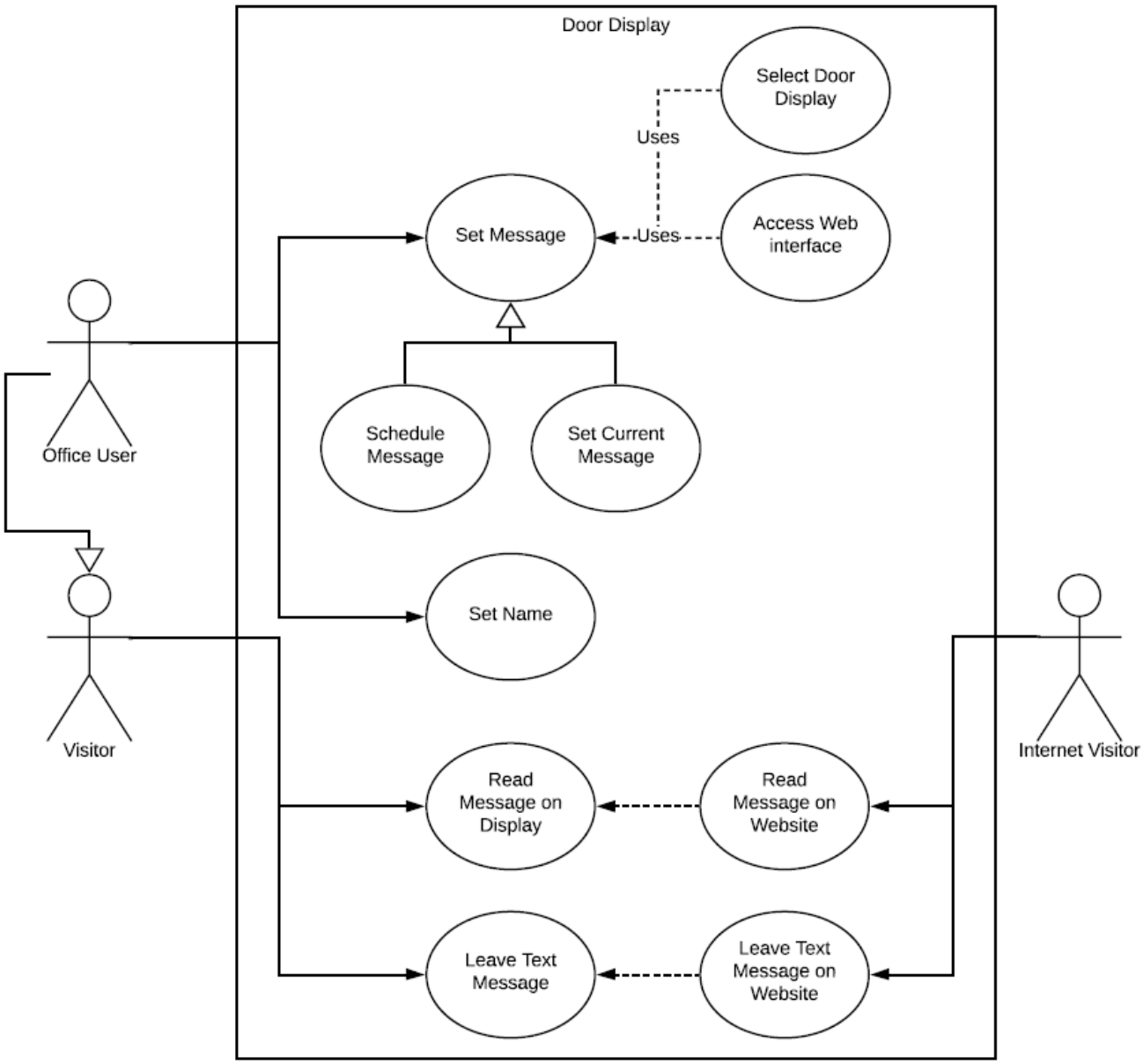
{ display: <ObjectID>, message: <String>, time: <Time/Date>, session: <Session> }
RES:
{
  accepted: <Boolean>,
  error: <Exception>,
  echo:{
    display: <ObjectID>,
    messageid: <ObjectID>
    message: <String>,
    time: <Time/Date>
  }
}
```

Remove scheduled message using 'messageid' and 'session'.
```
POST message/unschedule/ [WIP]
BODY:
{ messageid: <ObjectID>, session: <Session> }
RES:
{
  accepted: <Boolean>,
  error: <Exception>,
  echo:{
    id: <ID>,
    message: <String>,
    time: <Time/Date>
  }
}
```
Login using 'username' and 'password'.```
POST login/
BODY:
{ username: <String>, pwd: <String> }
RES:
{
  accepted: <Boolean>,
  session: <Session>,
  error: <Exception>
}
```
Create a new account with 'username' and 'password'.
```
POST login/create/
BODY:
{ username: <String>, pwd: <String> }
RES:
{
  accepted: <Boolean>,
  username: <Username>,
  error: <Exception>
```

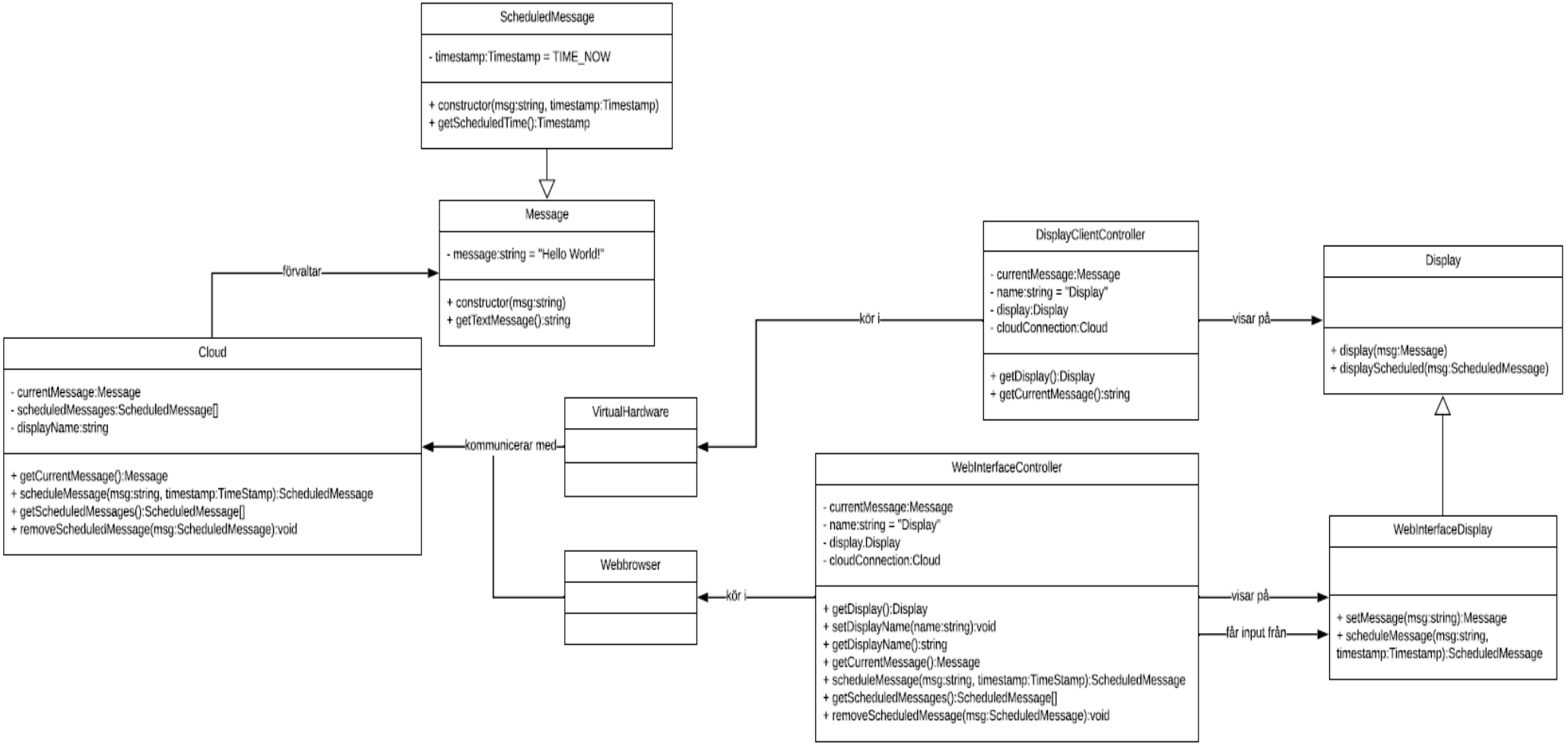
Bilaga 12


Node.js
Runs on
Heroku Host
Is Hosted on
Heroku App
Provided by
REST API
bcrypt (npm package)
Hashes passwords using
User processing (account.js)
Uses
Login Route (express route)
Display Route (express route)
Message Route (express route)
Uses DB Connection
Routes
Uses DB Connection
Routes
Uses DB Connection
Routes
DB Connection (db.js)
express (npm package)
Parse requests using
body-parser (npm package)
Uses
Connects to using credentials
Mongoose no-sql DB (Atlas)
mongoose (npm package)

Bilaga 13


Door Display
Select Door Display
Uses
Set Message
Uses
Access Web interface
Office User
Schedule Message
Set Current Message
Set Name
Visitor
Internet Visitor
Read Message on Display
Read Message on Website
Leave Text Message
Leave Text Message on Website

Bilaga 14


ScheduledMessage
- timestamp:Timestamp = TIME_NOW
+ constructor(msg:string, timestamp:Timestamp)
+ getScheduledTime():Timestamp
Message
- message:string = "Hello World!"
+ constructor(msg:string)
+ getTextMessage():string
förvaltar
Cloud
- currentMessage:Message
- scheduledMessages:ScheduledMessage[]
- displayName:string
+ getCurrentMessage():Message
+ scheduleMessage(msg:string, timestamp:TimeStamp):ScheduledMessage
+ getScheduledMessages():ScheduledMessage[]
+ removeScheduledMessage(msg:ScheduledMessage):void
kommunicerar med
VirtualHardware
kör i
DisplayClientController
- currentMessage:Message
- name:string = "Display"
- display:Display
- cloudConnection:Cloud
+ getDisplay():Display
+ getCurrentMessage():string
visar på
Display
+ display(msg:Message)
+ displayScheduled(msg:ScheduledMessage)
Webbrowser
kör i
WebInterfaceController
- currentMessage:Message
- name:string = "Display"
- display.Display
- cloudConnection:Cloud
+ getDisplay():Display
+ setDisplayName(name:string):void
+ getDisplayName():string
+ getCurrentMessage():Message
+ scheduleMessage(msg:string, timestamp:TimeStamp):ScheduledMessage
+ getScheduledMessages():ScheduledMessage[]
+ removeScheduledMessage(msg:ScheduledMessage):void
visar på
får input från
WebInterfaceDisplay
+ setMessage(msg:string):Message
+ scheduleMessage(msg:string,
timestamp:Timestamp):ScheduledMessage